Disclaimer and Legal Information
Contents

Notices and Disclaimers........................................................................................................ 15
Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference........................................... 16

Part I: Introducing the Intel® oneAPI DPC++/C++ Compiler
Feature Requirements........................................................................................................ 17
Getting Help and Support.................................................................................................. 18
Related Information......................................................................................................... 19
Notational Conventions.................................................................................................... 20

Part II: Compiler Setup
Using the Command Line................................................................................................ 23
  Specifying the Location of Compiler Components....................................................... 23
  Invoking the Compiler................................................................................................... 25
  Using the Command Line on Windows*........................................................................ 27
  Understanding File Extensions...................................................................................... 28
  Using Makefiles to Compile Your Application............................................................ 29
  Using CMake with Data Parallel C++ (DPC++)............................................................ 30
  Using Compiler Options............................................................................................... 31
  Specifying Include Files............................................................................................... 34
  Specifying Object Files................................................................................................. 35
  Specifying Assembly Files............................................................................................ 35
  Converting Projects to Use a Selected Compiler from the Command Line....................... 35
Using Eclipse*.................................................................................................................... 36
  Adding the Compiler to Eclipse*.................................................................................. 37
  Multi-Version Compiler Support.................................................................................. 37
Using Cheat Sheets.......................................................................................................... 37
Creating a Simple Project.................................................................................................. 38
  Creating a New Project................................................................................................. 38
  Adding a C Source File................................................................................................. 38
  Setting Options for a Project or File............................................................................... 39
  Excluding Source Files from a Build............................................................................ 39
  Building a Project......................................................................................................... 40
  Running a Project........................................................................................................... 40
  Error Parser.................................................................................................................. 41
Make Files.......................................................................................................................... 41
  Project Types and Makefiles.......................................................................................... 41
  Exporting Makefiles...................................................................................................... 41
Using Intel Libraries with Eclipse*.................................................................................... 42
Using Microsoft Visual Studio*........................................................................................ 44
  Creating a New Project.................................................................................................. 44
  Using the Intel® oneAPI DPC++/C++ Compiler............................................................. 45
  Build a Project.............................................................................................................. 46
  Selecting the Compiler Version.................................................................................... 46
  Switching Back to the Visual C++* Compiler............................................................... 46
  Selecting a Configuration............................................................................................. 47
  Specifying a Target Platform....................................................................................... 47
  Specifying Directory Paths........................................................................................... 48
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mconditional-branch, Qconditional-branch</td>
<td>109</td>
</tr>
<tr>
<td>mintrinsic-promote, Qintrinsic-promote</td>
<td>111</td>
</tr>
<tr>
<td>momit-leaf-frame-pointer</td>
<td>111</td>
</tr>
<tr>
<td>mregparm</td>
<td>113</td>
</tr>
<tr>
<td>mtune, tune</td>
<td>113</td>
</tr>
<tr>
<td>Qcxx-features</td>
<td>116</td>
</tr>
<tr>
<td>Qpatchable-addresses</td>
<td>117</td>
</tr>
<tr>
<td>Qsafeseh</td>
<td>117</td>
</tr>
<tr>
<td>regcall, Qregcall</td>
<td>118</td>
</tr>
<tr>
<td>x, Qx</td>
<td>119</td>
</tr>
<tr>
<td>xHost, Qxhost</td>
<td>122</td>
</tr>
<tr>
<td>Interprocedural Optimization (IPO) Options</td>
<td>125</td>
</tr>
<tr>
<td>ipo, Qipo</td>
<td>125</td>
</tr>
<tr>
<td>Advanced Optimization Options</td>
<td>126</td>
</tr>
<tr>
<td>ffreestanding, Qfreestanding</td>
<td>126</td>
</tr>
<tr>
<td>fjump-tables</td>
<td>127</td>
</tr>
<tr>
<td>ipp-link, Qipp-link</td>
<td>128</td>
</tr>
<tr>
<td>qactypes, Qactypes</td>
<td>129</td>
</tr>
<tr>
<td>qdaal, Qdaal</td>
<td>130</td>
</tr>
<tr>
<td>qipp, Qipp</td>
<td>131</td>
</tr>
<tr>
<td>qmkl, Qmkl</td>
<td>133</td>
</tr>
<tr>
<td>qopt-assume-no-loop-carried-dep, Qopt-assume-no-loop-carried-dep</td>
<td>134</td>
</tr>
<tr>
<td>qopt-for-throughput, Qopt-for-throughput</td>
<td>136</td>
</tr>
<tr>
<td>qopt-multiple-gather-scatter-by-shuffles, Qopt-multiple-gather-scatter-by-shuffles</td>
<td>136</td>
</tr>
<tr>
<td>qtbb, Qtbb</td>
<td>137</td>
</tr>
<tr>
<td>unroll, Qunroll</td>
<td>138</td>
</tr>
<tr>
<td>use-intel-optimized-headers, Quse-intel-optimized-headers</td>
<td>139</td>
</tr>
<tr>
<td>vec, Qvec</td>
<td>140</td>
</tr>
<tr>
<td>vec-threshold, Qvec-threshold</td>
<td>141</td>
</tr>
<tr>
<td>Optimization Report Options</td>
<td>142</td>
</tr>
<tr>
<td>qopt-report, Qopt-report</td>
<td>142</td>
</tr>
<tr>
<td>Offload Compilation Options, OpenMP* Options, and Parallel Processing Options</td>
<td>143</td>
</tr>
<tr>
<td>Device Offload Compilation Considerations</td>
<td>143</td>
</tr>
<tr>
<td>device-math-lib</td>
<td>143</td>
</tr>
<tr>
<td>fintelfpga</td>
<td>144</td>
</tr>
<tr>
<td>fiopenmp, Qiopenmp</td>
<td>145</td>
</tr>
<tr>
<td>fno-sycl-libspirv</td>
<td>146</td>
</tr>
<tr>
<td>foffload-static-lib</td>
<td>147</td>
</tr>
<tr>
<td>fopenmp</td>
<td>148</td>
</tr>
<tr>
<td>fopenmp-device-lib</td>
<td>148</td>
</tr>
<tr>
<td>fopenmp-targets, Qopenmp-targets</td>
<td>150</td>
</tr>
<tr>
<td>fsyscl</td>
<td>151</td>
</tr>
<tr>
<td>fsyscl-add-targets</td>
<td>152</td>
</tr>
<tr>
<td>fsyscl-dead-args-optimization</td>
<td>153</td>
</tr>
<tr>
<td>fsyscl-device-code-split</td>
<td>154</td>
</tr>
<tr>
<td>fsyscl-device-lib</td>
<td>154</td>
</tr>
<tr>
<td>fsyscl-device-only</td>
<td>155</td>
</tr>
<tr>
<td>fsyscl-early-optimizations</td>
<td>156</td>
</tr>
<tr>
<td>fsyscl-enable-function-pointers</td>
<td>157</td>
</tr>
<tr>
<td>fsyscl-explicit-simd</td>
<td>158</td>
</tr>
<tr>
<td>fsyscl-help</td>
<td>159</td>
</tr>
</tbody>
</table>
## Contents

- fsycl-host-compiler ................................................................. 160
- fsycl-host-compiler-options ................................................ 161
- fsycl-id-queries-fit-in-int .................................................. 161
- fsycl-link ........................................................................... 162
- fsycl-link-targets .............................................................. 163
- fsycl-max-parallel-link-jobs ................................................. 164
- fsycl-targets ...................................................................... 165
- fsycl-unnamed-lambda ....................................................... 167
- fsycl-use-bitcode .............................................................. 167
- nolibsycl ............................................................................ 168
- qopenmp, Qopenmp ............................................................ 169
- qopenmp-lib, Qopenmp-lib ............................................... 170
- qopenmp-link, Qopenmp-link ............................................. 171
- qopenmp-simd, Qopenmp-simd ......................................... 172
- qopenmp-stubs, Qopenmp-stubs ........................................... 174
- qopenmp-threadprivate, Qopenmp-threadprivate .................. 174
- reuse-exe .......................................................................... 175
- Wno-sycl-strict ................................................................... 176
- Xs ..................................................................................... 177
- Xopenmp-target ................................................................. 178
- Xsycl-target ...................................................................... 179

### Floating-Point Options

- ffp-contract ................................................................. 180
- fimf-absolute-error, Qimf-absolute-error ......................... 181
- fimf-accuracy-bits, Qimf-accuracy-bits ............................. 183
- fimf-arch-consistency, Qimf-arch-consistency .................... 184
- fimf-domain-exclusion, Qimf-domain-exclusion .............. 186
- fimf-force-dynamic-target, Qimf-force-dynamic-target ...... 189
- fimf-max-error, Qimf-max-error ...................................... 191
- fimf-precision, Qimf-precision ....................................... 192
- fimf-use-svml, Qimf-use-svml ......................................... 194
- fma, Qfma ....................................................................... 196
- fp-model, fp .................................................................... 197
- fp-speculation, Qfp-speculation ...................................... 199

### Inlining Options

- fgnu89-inline ................................................................. 200
- finline ............................................................................. 201
- finline-functions ............................................................ 201

### Output, Debug, and Precompiled Header (PCH) Options

- c ................................................................................... 202
- debug (Linux*) .............................................................. 203
- debug (Windows*) .......................................................... 205
- Fa .................................................................................. 207
- FA .................................................................................. 208
- fasm-blocks ................................................................. 208
- FC ............................................................................... 209
- Fd ................................................................................. 209
- FD ............................................................................... 210
- Fe .................................................................................. 211
- Fo .................................................................................. 212
- Fp .................................................................................. 212
- ftrapuv, Qtrapuv .......................................................... 213
- fverbose-asm ............................................................... 214
- g ................................................................................... 215
- gdwarf .......................................................................... 216
<table>
<thead>
<tr>
<th>Option</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gm</td>
<td>217</td>
</tr>
<tr>
<td>grecord-gcc-switches</td>
<td>218</td>
</tr>
<tr>
<td>gsplit-dwarf</td>
<td>218</td>
</tr>
<tr>
<td>o</td>
<td>219</td>
</tr>
<tr>
<td>pdbfile</td>
<td>220</td>
</tr>
<tr>
<td>print-multi-lib</td>
<td>221</td>
</tr>
<tr>
<td>RTC</td>
<td>221</td>
</tr>
<tr>
<td>S</td>
<td>222</td>
</tr>
<tr>
<td>use-msasm</td>
<td>223</td>
</tr>
<tr>
<td>Y</td>
<td>224</td>
</tr>
<tr>
<td>Yc</td>
<td>224</td>
</tr>
<tr>
<td>Yu</td>
<td>225</td>
</tr>
<tr>
<td>Zi, Z7, ZI</td>
<td>227</td>
</tr>
<tr>
<td>Preprocessor Options</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>228</td>
</tr>
<tr>
<td>C</td>
<td>229</td>
</tr>
<tr>
<td>D</td>
<td>230</td>
</tr>
<tr>
<td>dD, QdD</td>
<td>231</td>
</tr>
<tr>
<td>dM, QdM</td>
<td>232</td>
</tr>
<tr>
<td>E</td>
<td>232</td>
</tr>
<tr>
<td>EP</td>
<td>233</td>
</tr>
<tr>
<td>FI</td>
<td>234</td>
</tr>
<tr>
<td>H, QH</td>
<td>235</td>
</tr>
<tr>
<td>I</td>
<td>235</td>
</tr>
<tr>
<td>I-</td>
<td>236</td>
</tr>
<tr>
<td>idirafter</td>
<td>237</td>
</tr>
<tr>
<td>imacros</td>
<td>237</td>
</tr>
<tr>
<td>iprefix</td>
<td>238</td>
</tr>
<tr>
<td>iquote</td>
<td>238</td>
</tr>
<tr>
<td>isystem</td>
<td>238</td>
</tr>
<tr>
<td>iwithprefix</td>
<td>239</td>
</tr>
<tr>
<td>iwithprefixbefore</td>
<td>240</td>
</tr>
<tr>
<td>Kc++, TP</td>
<td>241</td>
</tr>
<tr>
<td>M, QM</td>
<td>241</td>
</tr>
<tr>
<td>MD, QMD</td>
<td>242</td>
</tr>
<tr>
<td>MF, QMF</td>
<td>243</td>
</tr>
<tr>
<td>MG, QMG</td>
<td>243</td>
</tr>
<tr>
<td>MM, QMM</td>
<td>244</td>
</tr>
<tr>
<td>MMD, QMMD</td>
<td>245</td>
</tr>
<tr>
<td>MP (Linux* OS)</td>
<td>245</td>
</tr>
<tr>
<td>MQ</td>
<td>246</td>
</tr>
<tr>
<td>MT, QMT</td>
<td>246</td>
</tr>
<tr>
<td>nostdinc++</td>
<td>247</td>
</tr>
<tr>
<td>P</td>
<td>247</td>
</tr>
<tr>
<td>pragma-optimization-level</td>
<td>248</td>
</tr>
<tr>
<td>u (Windows*)</td>
<td>249</td>
</tr>
<tr>
<td>U</td>
<td>250</td>
</tr>
<tr>
<td>undef</td>
<td>251</td>
</tr>
<tr>
<td>X</td>
<td>251</td>
</tr>
<tr>
<td>Component Control Options</td>
<td></td>
</tr>
<tr>
<td>Qinstall</td>
<td>252</td>
</tr>
<tr>
<td>Qlocation</td>
<td>252</td>
</tr>
<tr>
<td>Qoption</td>
<td>253</td>
</tr>
<tr>
<td>Language Options</td>
<td></td>
</tr>
<tr>
<td>ansi</td>
<td>255</td>
</tr>
</tbody>
</table>
Contents

fno-gnu-keywords......................................................... 256
fno-operator-names ...................................................... 256
fno-rtti ........................................................................ 257
fpermissive .................................................................. 257
fshort-enums ................................................................ 258
fsyntax-only .................................................................. 258
funsigned-char ............................................................. 259
GZ .............................................................................. 260
J ................................................................................ 260
std, Qstd ..................................................................... 261
strict-ansi .................................................................... 263
vd .............................................................................. 264
vmg ............................................................................ 265
x (type option)............................................................. 265
Zc .............................................................................. 266
Zg .............................................................................. 268
Zp .............................................................................. 268
Zs .............................................................................. 269

Data Options .................................................................... 270
align ........................................................................... 270
fcommon..................................................................... 271
fkeep-static-consts, Qkeep-static-consts ......................... 271
fmath-errno ................................................................. 272
fpack-struct ................................................................. 273
fpascal-strings.............................................................. 274
fpic ............................................................................. 274
fpie ............................................................................. 275
freg-struct-return ......................................................... 276
fstack-protector............................................................ 276
fstack-security-check .................................................... 278
fvisibility ................................................................. 278
fzero-initialized-in-bss, Qzero-initialized-in-bss ................. 280
GA .............................................................................. 281
Gs .............................................................................. 282
GS .............................................................................. 282
malign-double.............................................................. 283
mcmodel ..................................................................... 284
Qlong-double ............................................................... 285

Compiler Diagnostic Options ........................................... 287
w ............................................................................... 287
w0...w5, W0...W5......................................................... 288
Wabi ........................................................................... 289
Wall ............................................................................ 290
Wcomment .................................................................. 291
Wdeprecated ................................................................. 291
Weffc++, Qeffc++........................................................ 292
Werror, WX ................................................................. 293
Werror-all ................................................................... 294
Wextra-tokens .............................................................. 294
Wformat ................................................................. 295
Wformat-security .......................................................... 295
Wmain ................................................................. 296
Wmissing-declarations ................................................... 297
Wmissing-prototypes ................................................... 297
Wpointer-arith ............................................................. 298
dumpmachine .............................................................. 333
dumpversion .............................................................. 334
Gy .............................................................................. 334
help ............................................................................ 335
intel-freestanding ................................................................ 335
intel-freestanding-target-os ........................................... 337
multibyte-chars, Qmultibyte-chars .................................. 337
multiple-processes ........................................................ 338
nologo ........................................................................ 339
save-temps, Qsave-temps ............................................. 339
showIncludes ............................................................... 340
sox ............................................................................. 341
sysroot........................................................................ 342
Tc............................................................................... 343
TC .............................................................................. 343
Tp .............................................................................. 344
version ........................................................................ 345
watch .......................................................................... 345
Alternate Compiler Options ............................................................. 346
Related Options ............................................................................ 347
Portability Options ............................................................................ 347
GCC*-Compatible Warning Options.......................................... 353
Floating-Point Operations ................................................................. 354
Understanding Floating-Point Operations ................................................................. 354
Programming Tradeoffs in Floating-point Applications ...................... 354
Using the -fp-model (/fp) Option ............................................. 356
Denormal Numbers ........................................................................... 356
Setting the FTZ and DAZ Flags ............................................... 357
Overview: Tuning Performance ........................................................ 357
Handling Floating-point Array Operations in a Loop Body .................... 357
Reducing the Impact of Denormal Exceptions ..................................... 358
Avoiding Mixed Data Type Arithmetic Expressions ......................... 358
Using Efficient Data Types ...................................................... 359
Understanding IEEE Floating-Point Operations ...................................... 359
Special Values ................................................................. 359
Attributes ............................................................................................. 360
align .................................................................................. 361
align_value ........................................................................ 361
allow_cpu_features ....................................................................... 362
concurrency_safe ........................................................................ 364
const .............................................................................. 364
cpu_dispatch, cpu_specific ............................................................. 365
mpx ............................................................................. 366
Intrinsics.............................................................................................. 367
Libraries ............................................................................................... 368
Creating Libraries ........................................................................ 368
Using Intel Shared Libraries ......................................................... 369
Managing Libraries ........................................................................ 370
Redistributing Libraries When Deploying Applications ...................... 371
Resolving References to Shared Libraries Provided with Intel® oneAPI... 372
Intel's Memory Allocator Library ...................................................... 373
Introduction to the SIMD Data Layout Templates ................................ 374
Usage Guidelines: Function Calls and Containers ................................ 376
Constructing an n_container ......................................................... 377
Bounds................................................................................................. 379
Intel’s C++ Asynchronous I/O Extensions for Windows* Operating Systems .......................................................... 489

Intel’s C++ Asynchronous I/O Library for Windows* Operating Systems .............................................................. 490

aio_read ........................................................................... 490
aio_write ....................................................................... 491
Example for aio_read and aio_write Functions ................. 492
aio_suspend ..................................................................... 495
Example for aio_suspend Function .................................. 495
aio_error ........................................................................ 496
aio_return ....................................................................... 497
Example for aio_error and aio_return Functions ................ 497
aio_fsync ....................................................................... 498
aio_cancel ....................................................................... 499
Example for aio_cancel Function .................................... 499
lio_listio ......................................................................... 501
Example for lio_listio Function ...................................... 502
Handling Errors Caused by Asynchronous I/O Functions ..... 503

Intel’s C++ Asynchronous I/O Class for Windows* Operating Systems .................................................................... 504

Template Class async_class ............................................ 504
get_last_operation_id .................................................... 505
wait .............................................................................. 505
get_status ...................................................................... 506
get_last_error .............................................................. 506
get_error_operation_id .................................................. 506
stop_queue ..................................................................... 507
resume_queue ............................................................. 507
clear_queue .................................................................... 507
Example for Using async_class Template Class................. 508

IEEE 754-2008 Binary Floating-Point Conformance Library .......................................................... 509
Overview: Intel® IEEE 754-2008 Binary Floating-Point Conformance Library ......................................................... 509
Using the Intel® IEEE 754-2008 Binary Floating-Point Conformance Library .............................................................. 511
Function List ........................................................................ 512

Homogeneous General-Computational Operations Functions 516
General-Computational Operations Functions .................... 518
Quiet-Computational Operations Functions ....................... 524
Signaling-Computational Operations Functions ................. 525
Non-Computational Operations Functions ......................... 530

Intel’s Numeric String Conversion Library .......................................................... 535
Overview: Intel’s Numeric String Conversion Library ........... 535
Function List ........................................................................ 537

Macros......................................................................................... 542

ISO Standard Predefined Macros ........................................ 542
Additional Predefined Macros ........................................... 542
Use Predefined Macros for Intel® Compilers ......................... 548

Pragmas................................................................................. 550

Intel-SpecificPragma Reference ............................................ 550
alloc_section ...................................................................... 551
block_loop/noblock_loop.................................................. 552
code_align ........................................................................ 554
distribute_point ............................................................ 554
inline, noinline, forceinline.............................................. 556
Part IV: Compilation

Supported Environment Variables .......................................................... 589
Compilation Phases ........................................................................... 615
Passing Options to the Linker ............................................................... 615
Specifying Alternate Tools and Paths .................................................. 617
Using Configuration Files .................................................................. 618
Using Response Files ......................................................................... 619
Global Symbols and Visibility Attributes (Linux*) .............................. 620
Specifying Symbol Visibility Explicitly (Linux*) ................................. 621
Saving Compiler Information in Your Executable ............................... 621
Linking Debug Information ................................................................ 622
Ahead of Time Compilation ................................................................ 622
Use a Third-Party Compiler as a Host Compiler for DPC++ Code ........ 625

Part V: Optimization and Programming Guide

Extensions .................................................................................................. 627
OpenMP* Support .................................................................................. 627
  Adding OpenMP* Support to your Application ................................ 628
  Parallel Processing Model ................................................................. 629
  Worksharing Using OpenMP* ............................................................ 632
  Controlling Thread Allocation .......................................................... 641
  OpenMP* Pragmas Summary ............................................................. 642
  OpenMP* Library Support ................................................................. 648
    OpenMP* Run-time Library Routines ............................................. 648
    Intel® Compiler Extension Routines to OpenMP* ......................... 658
  OpenMP* Support Libraries .............................................................. 662
    Using the OpenMP* Libraries ......................................................... 664
  Thread Affinity Interface (Linux* and Windows*) ............................ 668
  OpenMP* Memory Spaces and Allocators ....................................... 686
  OpenMP* Advanced Issues ............................................................... 689
  OpenMP* Implementation-Defined Behaviors .................................. 691
  OpenMP* Examples ......................................................................... 693
Intel® oneAPI Level Zero Introduction ................................................ 694
Intel® oneAPI Level Zero Switch ...................................................... 695
Intel® oneAPI Level Zero Backend Specification ................................. 697
Vectorization........................................................................................................... 700
Automatic Vectorization .................................................................................. 700
  Programming Guidelines for Vectorization........................................... 700
  Using Automatic Vectorization........................................................... 705
  Vectorization and Loops ................................................................. 711
  Loop Constructs.................................................................................. 713
Explicit Vector Programming ........................................................................... 718
  User-Mandated or SIMD Vectorization........................................ 718
  SIMD-Enabled Functions ........................................................................ 720
  SIMD-Enabled Function Pointers..................................................... 730
  Vectorizing a Loop Using the _Simd Keyword .................................. 736
  Function Annotations and the SIMD Directive for Vectorization .... 737
  Explicit SIMD SYCL* Extension.......................................................... 739
High-Level Optimization (HLO) ............................................................... 741
  Interprocedural Optimization (IPO) ....................................................... 741
    Using IPO.................................................................................. 743
    IPO-Related Performance Issues................................................. 744
    IPO for Large Programs ............................................................... 745
  Understanding Code Layout and Multi-Object IPO ...................... 746
  Creating a Library from IPO Objects................................................. 746
  Inline Expansion of Functions.............................................................. 747
    Compiler Directed Inline Expansion of Functions..................... 748
    Developer Directed Inline Expansion of User Functions........... 749
Methods to Optimize Code Size ................................................................. 749
  Disable or Decrease the Amount of Inlining.................................... 750
  Strip Symbols from Your Binaries ..................................................... 751
  Dynamically Link Intel-Provided Libraries ..................................... 751
  Exclude Unused Code and Data from the Executable .................... 752
  Disable Recognition and Expansion of Intrinsic Functions .......... 752
  Optimize Exception Handling Data (Linux*) ................................ 753
  Disable Loop Unrolling ................................................................. 753
  Disable Automatic Vectorization .................................................... 754
  Avoid References to Compiler-Specific Libraries ......................... 754
  Use Inter-Procedural Optimization (IPO) ........................................ 755
Intel® Math Library ............................................................................................ 755
  Using the Intel® Math Library .............................................................. 756
  Math Functions ............................................................................. 760
    Function List .............................................................................. 760
    Trigonometric Functions ............................................................. 765
    Hyperbolic Functions ..................................................................... 769
    Exponential Functions ..................................................................... 771
    Special Functions .......................................................................... 775
    Nearest Integer Functions ......................................................... 778
    Remainder Functions ..................................................................... 780
    Miscellaneous Functions ............................................................... 781
    Complex Functions ......................................................................... 786
    C99 Macros .................................................................................. 790

Part VI: Compatibility and Portability
  Conformance to the C/C++/DPC++ Standards ........................................ 792
  GCC* Compatibility and Interoperability ............................................ 792
  Microsoft* Compatibility ................................................................. 794
  Precompiled Header Support.............................................................. 795
Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.
Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference

This document is for version 2022.0 of the compilers.

This guide provides information about the Intel® oneAPI DPC++/C++ Compiler (icx, icpx, dpcpp, dpcpp-cl) and runtime environment.

The Intel® oneAPI DPC++/C++ Compiler is available as part of the Intel® oneAPI Base Toolkit, Intel® oneAPI HPC Toolkit, Intel® oneAPI IoT Toolkit, or as a stand-alone compiler.

Refer to the Intel® oneAPI DPC++/C++ Compiler main page for more information about features, specifications, and downloads.

Use this guide to learn about:

- **Compiler Setup**: How to invoke the compiler on the command line or from within an IDE.
- **Compiler Options**: Information about options you can use to affect optimization, code generation, and more.
- **Pragmas**: Information about directives to provide the compiler with instructions for specific tasks, including splitting large loops into smaller ones, enabling or disabling optimization for code, or offloading computation to the target.
- **OpenMP® Support**: Compiler support for OpenMP 5.0 Version TR4 features and some OpenMP Version 5.1 features.

For more information, refer to Introducing the Intel® oneAPI DPC++/C++ Compiler.

For information about Intel intrinsics, visit Intel® Intrinsics Guide.

Notices and Important Information

- To use Microsoft Visual C++ (MSVC) compatible options with DPC++, use dpcpp-cl.
- In this document, you may see features labeled as experimental. An experimental feature is one that requires further testing and possible refinement. Depending on testing results, such features may be fully defined and implemented or they may be removed in a future release.
- The Intel® oneAPI DPC++/C++ Compiler (icx, icpx, dpcpp, dpcpp-cl) does not support macOS*. For macOS or Xcode* support use Intel® C++ Compiler Classic. For more information, visit the Intel® C++ Compiler Classic Developer Guide and Reference.

Using the Compiler Documentation

- **Context Sensitive/F1 Help**: To use the Context Sensitive/F1 Help feature, visit the Download Documentation: Intel® Compiler (Current and Previous) page and follow the provided instructions.
- **Previous Versions of the Developer Guide and Reference**: Visit the Download Documentation: Intel® Compiler (Current and Previous) page to download PDF or FAR HTML versions of previous compiler documentation.

**NOTE** For the best search experience, use a Google Chrome® or Internet Explorer® browser to view your downloaded copy of the Developer Guide and Reference.

If you use Mozilla Firefox®, you may encounter an issue where the Search tab does not work. As a workaround, you can use the Contents and Index tabs or a third-party search tool to find your content.
Introducing the Intel® oneAPI DPC++/C++ Compiler

Using the Intel® oneAPI DPC++/C++ Compiler, you can compile and generate applications that can run on Intel® 64 architecture. You can also create programs for the IA-32 architecture on Windows* and Linux*.

NOTE IA-32 architecture is specific to C++; it does not apply to DPC++.

Intel® 64 architecture applications can run on the following:

- Windows operating systems for Intel® 64 architecture-based systems.
- Linux operating systems for Intel® 64 architecture-based systems.

IA-32 architecture applications can run on the following:

- Supported Windows operating systems
- Supported Linux operating systems

Unless specified otherwise, assume the information in this document applies to all supported architectures and all operating systems.

You can use the compiler in the command-line or in a supported Integrated Development Environment (IDE):

- Microsoft Visual Studio* (Windows only)
- Eclipse*/CDT (Linux only)

See the Release Notes for complete information on supported architectures, operating systems, and IDEs for this release.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

Feature Requirements

To use these tools and features, you need licensed versions of the tools and you must have an appropriately supported version of the product edition. For more information, check the product release notes.

NOTE Some features may require additional product installation.

The following table shows components (tools) and where to find additional information on them.

<table>
<thead>
<tr>
<th>Component</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® oneAPI DPC++/C++ Compiler</td>
<td>More information on tools and features can be found on the Intel® Developer Zone and the Software Development Tools pages.</td>
</tr>
<tr>
<td>Intel® Advisor</td>
<td></td>
</tr>
</tbody>
</table>
The following table lists dependent features and their corresponding required products. For certain compiler options, the compilation may fail if the option is specified but the required product is not installed. In this case, remove the option from the command line and recompile.

### Feature Requirements

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® oneAPI Threading Building Blocks (oneTBB)</td>
<td>The -tbb, -qtbb, and /Qtbb options require a oneTBB install.</td>
</tr>
<tr>
<td>Intel® oneAPI Math Kernel Library (oneMKL)</td>
<td>The -qmkl, -mkl, and /Qmkl options require a oneMKL install.</td>
</tr>
<tr>
<td>Intel® oneAPI Data Analytics Library (oneDAL)</td>
<td>The -daal, -qdaal, and /Qdaal options require a oneDAL install.</td>
</tr>
<tr>
<td>Intel® Integrated Performance Primitives Cryptography (Intel® IPP Cryptography)</td>
<td>Use crypto to link to the Intel® IPP Cryptography library.</td>
</tr>
<tr>
<td>Thread Checking</td>
<td>Intel® Inspector</td>
</tr>
<tr>
<td>Trace Analyzing and Collecting</td>
<td>Intel® Trace Analyzer and Collector</td>
</tr>
</tbody>
</table>

Compiler options related to this feature may require a set-up script. For further information, see the product documentation.

Refer to the Release Notes for detailed information about system requirements, late changes to the products, supported architectures, operating systems, and Integrated Development Environments (IDEs).

## Getting Help and Support

### Intel® Software Documentation

You can find product documentation for many released products at: https://software.intel.com/content/www/us/en/develop/documentation.html

### Product Website and Support

To find product information, register your product, or contact Intel, visit: https://software.intel.com/content/www/us/en/develop/support.html

At this site, you will find comprehensive product information, including:

- Links to Get Started, Documentation, Individual Support, and Registration
- Links to information such as white papers, articles, and user forums
- Links to product information
- Links to news and events
Online Service Center
For more information about the Online Service Center visit: https://supporttickets.intel.com/servicecenter

NOTE To access support, you must register your product at the Intel® Registration Center: https://registrationcenter.intel.com/en/products/

Release Notes
For detailed information on system requirements, late changes to the products, supported architectures, operating systems, and Integrated Development Environments (IDE) see the Release Notes for the product.

Forums
You can find helpful information in the Intel Software user forums. You can also submit questions to the forums. To see the list of the available forums, go to https://community.intel.com/t5/Software-Development-Tools/ct-p/software-dev-tools

Related Information

Recommended Additional Reading
You are strongly encouraged to read the following books for in-depth understanding of threading. Each book discusses general concepts of parallel programming by explaining a particular programming technology:

• For information on OpenMP® technology: Chapman, Barbara, Gabriele Jost, Ruud van der Pas, and David J. Kuck (foreword). Using OpenMP: Portable Shared Memory Parallel Programming. MIT Press, October 2007
• For information on Microsoft Win32® Threading (for Windows® users): Akhter, Shameem, and Jason Roberts. Multi-Core Programming: Increasing Performance through Software Multithreading, Intel Press, April 2006

Intel does not endorse these books or recommend them over other books on the same subjects.

Additional Product Information
For additional technical product information including white papers, forums, and documentation, visit https://software.intel.com/content/www/us/en/develop/tools.html

Additional Language Information
• For information on Data Parallel C++ (DPC++) and mastering DPC++ for programming of heterogeneous systems using C++ and SYCL, visit https://link.springer.com/book/10.1007%2F978-1-4842-5574-2 for a book download.
• For information about the C++ standards, visit the C++ website: http://www.isocpp.org/
• For information about the C standards, visit the C website: http://www.open-std.org/jtc1/sc22/wg14/
• For information about the OpenMP® standards, visit the OpenMP website: http://www.openmp.org/
Notational Conventions

Information in this documentation applies to all supported operating systems and architectures unless otherwise specified. This documentation uses the following conventions:

<table>
<thead>
<tr>
<th>Notational Conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THIS TYPE</td>
<td>Indicates language keywords.</td>
</tr>
<tr>
<td>this type</td>
<td>Indicates command-line or option arguments.</td>
</tr>
<tr>
<td>This type</td>
<td>Indicates a code example.</td>
</tr>
<tr>
<td>This type</td>
<td>Indicates what you type as input.</td>
</tr>
<tr>
<td>This type</td>
<td>Indicates menu names, menu items, button names, dialog window names, and other user-interface items.</td>
</tr>
<tr>
<td>File &gt; Open</td>
<td>Menu names and menu items joined by a greater than (&gt;) sign to indicate a sequence of actions. For example, <strong>Click File &gt; Open</strong> indicates that in the File menu, you would click <strong>Open</strong> to perform this action.</td>
</tr>
<tr>
<td>{value</td>
<td>value}</td>
</tr>
<tr>
<td>[item]</td>
<td>Indicates items that are optional.</td>
</tr>
<tr>
<td>item [, item ]...</td>
<td>Indicates that the item preceding the ellipsis (…) can be repeated.</td>
</tr>
<tr>
<td>Intel® C++</td>
<td>This term refers to the name of the common compiler language supported by the Intel® oneAPI DPC++/C++ Compiler.</td>
</tr>
<tr>
<td>compiler or the compiler</td>
<td>These terms are used when information is not limited to only one specific compiler, or when it is not necessary to indicate a specific compiler.</td>
</tr>
<tr>
<td>Windows* or Windows operating system</td>
<td>These terms refer to all supported Microsoft Windows operating systems.</td>
</tr>
<tr>
<td>Linux or Linux operating system</td>
<td>These terms refer to all supported Linux operating systems.</td>
</tr>
<tr>
<td>Microsoft Visual Studio*</td>
<td>An asterisk at the end of a word or name indicates it is a third-party product trademark.</td>
</tr>
<tr>
<td>compiler option</td>
<td>This term refers to Linux or Windows options, which are used by the compiler to compile applications.</td>
</tr>
<tr>
<td>The following conventions are used as shortcuts when referencing compiler option names in text:</td>
<td></td>
</tr>
</tbody>
</table>
• Many options have names that are the same on Linux and Windows, except that the Windows form starts with an initial / and the Linux form starts with an initial -. Within text, such option names are shown without the initial character. For example, check.

• Many options have names that are the same on Linux and Windows, except that the Windows form starts with an initial Q. Within text, such option names are shown as [Q]option-name.

  For example, if you see a reference to [Q]ipo, the Linux form of the option is -ipo and the Windows form of the option is /Qipo.

• This content is specific to C++; it does not apply to DPC++.

Several compiler options have similar names except that the Linux forms start with an initial q and the Windows form starts with an initial Q. Within text, such option names are shown as [q or Q]option-name.

  For example, if you see a reference to [q or Q]opt-report, the Linux form of the option is -qopt-report and the Windows form of the option is /Qopt-report.

Other dissimilar compiler option names are shown in full.

**Conventions Used in Compiler Options**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/option or -option</td>
<td>A slash before an option name indicates the option is available on Windows. A dash before an option name indicates the option is available on Linux systems. For example:</td>
</tr>
<tr>
<td></td>
<td>• Windows option: /help</td>
</tr>
<tr>
<td></td>
<td>• Linux option: -help</td>
</tr>
<tr>
<td>/option:argument or -option=argument</td>
<td>Indicates that an option requires an argument (parameter).</td>
</tr>
<tr>
<td>/option:keyword or -option=keyword</td>
<td>Indicates that an option requires one of the keyword values.</td>
</tr>
<tr>
<td>/option[:keyword ] or</td>
<td>Indicates that the option can be used alone or with an optional keyword.</td>
</tr>
</tbody>
</table>
-option[=keyword ]

option[=n] or option[=n] or option[=n]

Indicates that the option can be used alone or with an optional value. For example, in -unroll[=n], the n can be omitted or a valid value can be specified for n.

option[-] or option[-]

Indicates that a trailing hyphen disables the option. For example, /Qglobal_hoist- disables the Windows option /Qglobal_hoist.

[no]option or [no-]option or [no-]option

Indicates that no or no- preceding an option disables the option.

In the Linux option -[no-]global_hoist, -global_hoist enables the option, while -no-global_hoist disables it.

In some options, the no appears later in the option name. For example, -fno-common disables the -fcommon option.
You can use the Intel® oneAPI DPC++/C++ Compiler from the command line, or from the IDEs listed below. These IDEs are described in further detail in their corresponding sections.

Using the Command Line

This section provides information about the Command Line Interface (CLI).

Specifying the Location of Compiler Components

Before you invoke the compiler, you may need to set certain environment variables that define the location of compiler-related components. The Intel® oneAPI DPC++/C++ Compiler includes environment configuration scripts to configure your build and development environment variables:

- On Linux*, the file is a shell script called `setvars.sh`.
- On Windows*, the file is a batch file called `setvars.bat`.

The following information is operating system dependent.

**NOTE** The Intel oneAPI DPC++/C++ Compiler is designed and tested only for use on 64-bit Linux and Windows operating systems, 32-bit operating systems are not supported. Additionally, the macOS* operating system is not supported by the compiler.

**Linux:**

Set the environment variables before using the compiler by sourcing the shell script `setvars.sh`. Depending on the shell, you can use the `source` command or a . (dot) to source the shell script, according to the following rule for a .sh script:

```
source /<install-dir>/setvars.sh <arg1> <arg2> ... <argn>
```

- examples: (assuming <install-dir> is /opt/intel/oneapi)
  ```
prompt> source /opt/intel/oneapi/setvars.sh intel64
  prompt> . /opt/intel/oneapi/setvars.sh intel64
  ```

  **NOTE** Type: `source /<install-dir>/setvars.sh --help` for more setvars usage information.

The compiler environment script file accepts an optional target architecture argument `<arg>`:

- `intel64`: Generate code and use libraries for Intel® 64 architecture-based targets.
- `ia32`: Generate code and use libraries for IA-32 architecture-based targets.
If you want the `setvars.sh` script to run automatically in all of your terminal sessions, add the source `setvars.sh` command to your startup file. For example, inside your `.bash_profile` entry for Intel® 64 architecture targets:

```
# set environment vars for Intel® oneAPI
DPC++/C++ Compiler
source <install-dir>/setvars.sh intel64
```

If the proper environment variables are not set, errors similar to the following may appear when attempting to execute a compiled program:

```
# for C++
./a.out: error while loading shared libraries:
libimf.so: cannot open shared object file: No such file or directory

# for DPC++
./a.out: error while loading shared libraries:
libsycl.so: cannot open shared object file: No such file or directory
```

**Windows:**

Under normal circumstances, you do not need to run the `setvars.bat` batch file. The terminal shortcuts in the Windows **Start** menu, **Intel oneAPI command prompt for <target architecture> for Visual Studio <year>**, set these variables automatically.

For additional information, see [Using the Command Line on Windows](#).

**NOTE** You need to run the `setvars` batch file if a command line is opened without using one of the provided **Command Prompt** menu items in the **Start** menu, or if you want to use the compiler from a script of your own.

The `setvars` batch file inserts DLL directories used by the compiler and libraries at the beginning of the existing **Path**. Because these directories appear first, they are searched before any directories that were part of the original **Path** provided by Windows (and other applications). This is especially important if the original **Path** includes directories with files that have the same names as those added by the compiler and libraries.

The `setvars` batch file takes multiple optional arguments; the following two arguments are recognized for compiler and library initialization:

```
<install-dir>\setvars.bat [<arg1>] [<arg2>]
```

Where `<arg1>` is optional and can be one of the following:

- `intel64`: Generate code and use libraries for Intel® 64 architecture (host and target).
- `ia32`: Generate code and use libraries for IA-32 architecture (host and target).

With the `dpcpp-cl` compiler, `<arg1>` is `intel64` by default.

The `<arg2>` is optional. If specified, it is one of the following:

- `vs2019`: Microsoft Visual Studio® 2019
- `vs2017`: Microsoft Visual Studio 2017

**NOTE** If `<arg1>` is not specified, the script uses the `intel64` argument by default. If `<arg2>` is not specified, the script uses the highest installed version of Microsoft Visual Studio detected during the installation procedure.
NOTE Support for Microsoft Visual Studio 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

See Also
oneAPI Development Environment Setup
Configure Your CPU or GPU System

Invoking the Compiler

Requirements Before Using the Command Line
You may need to set certain environment variables before using the command line. For more information, see Specifying the Location of Compiler Components.

Different Compilers and Drivers
The table below provides the different compiler front-end and driver information.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Notes</th>
<th>Linux* Driver</th>
<th>Windows* Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® DPC++ Compiler</td>
<td>A C++ and Khronos SYCL* compiler with a Clang front-end.</td>
<td>dpcpp</td>
<td>dpcpp (clang compatible)</td>
</tr>
<tr>
<td>Intel® C++ Compiler</td>
<td>A C++ compiler with a Clang front-end, supporting OpenMP* offload.</td>
<td>icx for C</td>
<td>icx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>icpx for C++</td>
<td></td>
</tr>
</tbody>
</table>

NOTE To use Microsoft Visual C++* (MSVC) compatible options, use dpcpp-cl.

Using the Compiler from the Command Line
Use the compiler with the OS/language specific invocations below.

NOTE You can also use the compiler from within the IDE. For more information on using Microsoft Visual Studio*, see Using Microsoft Visual Studio. For information on using Eclipse*, see Using Eclipse.

Linux:
Invoke the compiler using icx/icpx (for C/C++) or dpcpp (for DPC++) to compile LLVM C/C++/DPC++ source files.

- When you invoke the compiler with dpcpp the compiler builds DPC++ source files using DPC++ libraries and DPC++ include files. If you use dpcpp with a C source file, it is compiled as a DPC++ file. Use dpcpp to link DPC++ object files.
- When you invoke the compiler with icx the compiler builds LLVM C source files using LLVM C libraries and LLVM C include files. If you use icx with a C++ source file, it is compiled as an LLVM C file. Use icx to link LLVM C object files.
• When you invoke the compiler with `icpx` the compiler builds LLVM C++ source files using LLVM C++ libraries and LLVM C++ include files. If you use `icpx` with a C source file, it is compiled as an LLVM C++ file. Use `icpx` to link LLVM C++ object files.

**NOTE** If you are using `icpx` as your main compiler, you must add `-fsycl` at the link stage, otherwise the offload kernels are not available in the main binary.

The `icx/icpx` (for C/C++) or `dpcpp` (for DPC++) command does the following:

• Compiles and links the input source file(s).

• Produces one executable file, `a.out`, in the current directory.

**Windows:**

You can invoke the compiler on the command line using the `icx` (for C/C++) or `dpcpp-cl` (for DPC++) command. This command:

• Compiles and links the input source file(s).

• Produces object file(s) and assigns the names of the respective source file(s), but with a `.obj` extension.

• Produces one executable file and assigns it the name of the first input file on the command line, but with a `.exe` extension.

• Places all the files in the current directory.

When compilation occurs with the compiler, many tools may be called to complete the task that may reproduce diagnostics unique to the given tool. For instance, the linker may return a message if it cannot resolve a global reference. The `watch` option can help clarify which component is generating the error.

**Command Line Syntax**

When you invoke the compiler, the syntax is: For C/C++ projects:

// (Linux)
`{icx/icpx} [options] file1 [file2...]`

// (Windows)
`icx [options] file1 [file2...] [/link link_options]`

For DPC++ projects:

// (Linux)
`dpcpp [options] file1 [file2...]`

// (Windows)
`dpcpp-cl [options] file1 [file2...] [/link link_options]`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Indicates one or more command line options. On Linux systems, the compiler recognizes one or more letters preceded by a hyphen (-). On Windows, options are preceded by a slash (/). This includes linker options. Options are not required when invoking the compiler. The default behavior of the compiler implies that some options are ON by default when invoking compiler.</td>
</tr>
<tr>
<td>file1, file2...</td>
<td>Indicates one or more files to be processed by the compiler. You can specify more than one file, using space as a delimiter for multiple files.</td>
</tr>
<tr>
<td>/link (Windows)</td>
<td>All options following <code>/link</code> are passed to the linker. Compiler options must precede <code>link</code> if they are not to be passed to the linker.</td>
</tr>
</tbody>
</table>
Other Methods for Using the Command Line to Invoke the Compiler

- **Using makefiles from the Command Line:** Use makefiles to specify a number of files with various paths and to save this information for multiple compilations. For more information on using makefiles, see [Using Makefiles to Compile Your Application](#).

- **Using a Batch File from the Command Line:** Create and use a `.bat` file to consistently execute the compiler with a desired set of options instead of retyping the command each time you need to recompile.

Enabling OpenMP* Offloading

To enable OpenMP* offloading for C++ applications, invoke the compiler with:

- `icpx -fiopenmp -fopenmp-targets=<arch>` (Linux)
- `icx /Qiopenmp /Qopenmp-targets:<arch>` (Windows).

To enable OpenMP offloading for DPC++ applications, invoke the compiler with:

- `dpcpp -fiopenmp -fopenmp-targets=<arch>` (Linux)
- `dpcpp-cl /Qiopenmp /Qopenmp-targets:<arch>` (Windows)

See Also

- Specifying the Location of Compiler Components
- Understanding File Extensions
- Using Makefiles to Compile Your Application
- `watch` compiler option

**Using the Command Line on Windows***

The compiler provides a shortcut to access the command line with the appropriate environment variables already set.

**NOTE** Instructions and menu options may vary by Windows* version.

To invoke the compiler from the command line:

1. Open the Windows **Start** menu.
2. Scroll down the list of apps (programs) in the **Start** menu and find the **Intel oneAPI 2021** folder.
3. Left click on the folder name and select your component.

**NOTE** The command prompts shown are dependent on the versions of Microsoft Visual Studio* you have installed on your machine.

4. Right click on the command prompt icon to pin it to your taskbar.

**NOTE** This step is optional.

The command line opens.

You can use any command recognized by the Windows command prompt, plus some additional commands. Because the command line runs within the context of Windows, you can easily switch between the command line and other applications for Windows or have multiple instances of the command line open simultaneously.

When you are finished working in a command line, use the **exit** command to close and end the session.
Understanding File Extensions

Input File Extensions
The Intel® oneAPI DPC++/C++ Compiler recognizes input files with the extensions listed in the following table:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Interpretation</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.c</td>
<td>C source file</td>
<td>Passed to compiler</td>
</tr>
<tr>
<td>file.C</td>
<td>C++ source file</td>
<td>Passed to compiler</td>
</tr>
<tr>
<td>file.CC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.cc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.cpp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.cxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.lib</td>
<td>Library file</td>
<td>Passed to linker</td>
</tr>
<tr>
<td></td>
<td>(Windows*)</td>
<td></td>
</tr>
<tr>
<td>file.a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.so</td>
<td>(Linux*)</td>
<td></td>
</tr>
<tr>
<td>file.i</td>
<td>Preprocessed file</td>
<td>Passed to compiler</td>
</tr>
<tr>
<td>file.obj</td>
<td>Object file</td>
<td>Passed to linker</td>
</tr>
<tr>
<td></td>
<td>(Windows)</td>
<td></td>
</tr>
<tr>
<td>file.o</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Linux)</td>
<td></td>
</tr>
<tr>
<td>file.asm</td>
<td>Assembly file</td>
<td>Passed to assembler</td>
</tr>
<tr>
<td></td>
<td>(Windows)</td>
<td></td>
</tr>
<tr>
<td>file.s</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Linux)</td>
<td></td>
</tr>
<tr>
<td>file.S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output File Extensions
The Intel® oneAPI DPC++/C++ Compiler produces output files with the extensions listed in the following table:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.i</td>
<td>Preprocessed file: Produced with the -P option.</td>
</tr>
<tr>
<td>file.o</td>
<td>Object file: Produced with the -c (Linux and Windows) object. The /Fo (Windows) or -o (Linux) option allows you to rename the output object file.</td>
</tr>
<tr>
<td>file.obj</td>
<td>(Windows)</td>
</tr>
<tr>
<td>file.s</td>
<td>Assembly language file: Produced with the -S option. The /Fa (Windows) or -s (Linux) option allows you to rename the output assembly file.</td>
</tr>
<tr>
<td></td>
<td>(Linux)</td>
</tr>
<tr>
<td>file.asm</td>
<td>(Windows)</td>
</tr>
<tr>
<td>a.out</td>
<td>Executable file: Produced by the default compilation.</td>
</tr>
<tr>
<td></td>
<td>(Linux)</td>
</tr>
<tr>
<td>File Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>file.exe</td>
<td>The /Fe (Windows) or -o (Linux) option allows you to rename the output executable file.</td>
</tr>
</tbody>
</table>

See Also
Invoking the Compiler
Specifying Object Files
Specifying Assembly Files

Using Makefiles to Compile Your Application

This topic describes the use of makefiles to compile your application. You can use makefiles to specify a number of files with various paths, and to save this information for multiple compilations.

Using Makefiles to Store Information for Compilation on Linux*

To run `make` from the command line using the Intel® oneAPI DPC++/C++ Compiler, make sure that `/usr/bin` and `/usr/local/bin` are in your `PATH` environment variable.

If you use the C shell, you can edit your `.cshrc` file and add the following:

```
setenv PATH /usr/bin:/usr/local/bin:$$PATH
```

To use the Intel oneAPI DPC++/C++ Compiler, your makefile must include the setting `CC=icx` (for C), `CC=icpx` (for C++), or `CC=dpcpp` (for DPC++). Use the same setting on the command line to instruct the makefile to use the compiler. If your makefile is written for GCC*, you need to change the command line options that are not recognized by the compiler. Run `make`, using the following syntax:

```
make -f yourmakefile
```

Where `-f` is the `make` command option to specify a particular makefile name.

Using Makefiles to Store Information for Compilation on Windows*

To use a makefile to compile your source files, use the `nmake` command with the following syntax:

```
nmake /f [makefile_name.mak] CPP=[compiler_name] [LINK32=[linker_name]]
```

For C/C++ projects:

```
prompt> nmake /f your_project.mak CPP=icx LINK32=link
-OR-
prompt> nmake /f your_project.mak CPP=icx LINK32=xilink
```

For DPC++ projects:

```
prompt> nmake /f your_project.mak CPP=dpcpp-cl LINK32=dpcpp-cl
```

**NOTE**

If you have `link/xilink` specific options that are not accepted by `dpcpp-cl`, ensure any linker specific options are placed after the `/link` option. For example: `dpcpp test.obj <compiler options> /link <linker options>`
The `nmake` command creates object files (.obj) and executable files () from the information specified in the `your_project.mak` makefile.

**See Also**
- Modifying Your makefile (Linux)
- Modifying Your makefile (Windows*)

## Using CMake with Data Parallel C++ (DPC++)

The following content is OS specific.

### Linux*

Using CMake with the Intel® oneAPI DPC++/C++ Compiler on Linux is supported. When you are using CMake, the compiler is enabled using the `icx` (variant) binary. You may need to set your `CC/CXX` or `CMAKE_C_COMPILER /CMAKE_CXX_COMPILER` string to `icx/icpx`. For example:

```
$ cmake -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx ...

-OR-

$ export CC=icx; export CXX=icpx; cmake ...
```

### Windows*

Using CMake with the Intel® oneAPI DPC++/C++ Compiler on Windows is supported. When you are using CMake, the compiler is enabled using the `icx` (variant) binary. You may need to set your `CC/CXX` or `CMAKE_C_COMPILER /CMAKE_CXX_COMPILER` string to `icx`. The supported generator in the Windows environment is Ninja*.

```
$ cmake -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icx -GNinja ...

-OR-

$ CC=icx CXX=icx cmake -GNinja ...
```

### Support

Use the following steps to enable the compiler for your project:

1. Add the following snippet to your project’s `CMakeLists.txt`:

   ```cmake
   cmake_minimum_required(VERSION 3.21.0)
   
   NOTE The minimum required version for using CMake with the Intel® oneAPI DPC++/C++ Compiler is 3.21.
   ```
2. Add the following snippet to your project’s CMakeLists.txt:

```cmake
find_package(IntelDPCPP REQUIRED)
```

This snippet enables the dpcpp compiler. The DPC++ configuration package
(IntelDPCPPConfig.cmake) is shipped with the Intel® oneAPI DPC++/C++ Compiler. The package
directory is found in the parent directory of icx’s bin directory. Example: /deploy/linux_prod/
IntelDPCPP/IntelDPCPPConfig.cmake.

3. Select the appropriate compilers for C and C++; see the Linux and Windows sections above for specific
settings.

4. Run CMake and build your applications as normal.

5. The DPC++ configuration package exposes other variables that may be required. Refer to the package
for more information.

### Using Compiler Options

A compiler option is a case-sensitive, command line expression used to change the compiler’s default
operation. Compiler options are not required to compile your program, but they can control different aspects
of your application, such as:

- Code generation
- Optimization
- Output file (type, name, location)
- Linking properties
- Size of the executable
- Speed of the executable

See the Option Categories section for the option capabilities included with the Intel® oneAPI DPC++/C++
Compiler.

#### Command Line Syntax (Linux*)

When you specify compiler options on the command line, the following syntax applies:

```bash
// Linux
[invocation] [options] [@response_file] file1 [file2...]
```

The *invocation* is icx for C, icpx for C++, or dpcpp for DPC++.

The *options* represents zero or more compiler options and the *file* is any of the following:

- C or C++ source file (.c, .cc, .cpp, .cxx, .c++, .i, .ii)
- Assembly file (.s, .S)
- Object file (.o)
- Static library (.a)

When compiling C language sources, invoke the compiler with icx. When compiling C++ language sources or
a combination of C and C++, invoke the compiler with icpx. When compiling DPC++ sources, invoke the
compiler with dpcpp.

#### Command Line Syntax (Windows*)

When you specify compiler options on the command line, the following syntax applies:

```bash
[invocation] [options] [@response_file] file1 [file2 ...] [/link linker_options]
```

The *invocation* is icx for C/C++, or dpcpp-cl for DPC++.

The *options* represents zero or more compiler options, the *linker_options* represents zero or more linker
options, and the *file* is any of the following:

- C or C++ source file (.c, .cc, .cpp, .cxx, .i)
• Assembly file (.asm)
• Object (.obj)
• Static library (.lib)

The optional response_file is a text file that lists the compiler options you want to include during compilation. See Using Response Files for additional information.

Default Operation
The compiler invokes many options by default. In this example, the compiler includes the option O2 (and other default options) in the compilation:

```c
// Linux
[invocation] main.c
```

```c
// Windows
[invocation] main.c
```

The invocation is icx for C, icpx for C++, or dpcpp for DPC++ for Linux projects and icx for C/C++, or dpcpp-cl for DPC++ for Windows projects.

Each time you invoke the compiler, options listed in the corresponding configuration file override any competing default options. For example, if your configuration file includes the O3 option, the compiler uses O3 rather than the default O2 option. Use the configuration file to list the options for the compiler to use for every compilation. See Using Configuration Files.

**NOTE** The default .cfg files are not valid for the Intel oneAPI DPC++/C++ Compiler. You can use the -config<name> option instead of a default .cfg file. <name> can be a configuration file that is in the bin directory, or you can use the full path your selected .cfg file.

Options specified in the command line environment variable override any competing default options and options listed in the configuration file.

Finally, options used on the command line override any competing options that may be specified elsewhere (default options, options in the configuration file, and options specified in the command line environment variable). If you specify the option O1 on the command line, this option setting takes precedence over competing option defaults and competing options in the configuration files, in addition to the competing options in the command line environment variable.

Certain #pragma statements in your source code can override competing options specified on the command line. For example, if a function in your code is preceded by #pragma optimize("", off), then optimization for that function is turned off, even though O2 optimization is on by default, the O3 is listed in the configuration file, and the O1 is specified on the command line for the rest of the program.

Using Competing Options
The compiler reads command line options from left to right. If your compilation includes competing options, then the compiler uses the one furthest to the right. For example:

For C:

```c
// Linux icx -xSSSE3 main.c file1.c -xSSE4.2 file2.c
```

```c
// Windows icx /QxSSSE3 main.c file1.c /QxSSE4.2 file2.c
```

For C++:

```c
// Linux icpx -xSSSE3 main.c file1.c -xSSE4.2 file2.c
```

```c
// Windows icx /QxSSSE3 main.c file1.c /QxSSE4.2 file2.c
```
For DPC++:

```
// Linux
dpcpp -O1 main.c file1.c -O2 file2.c
// Windows
dpcpp-cl /O1 main.c file1.c /O2 file2.c
```

The compiler sees \[Q\]xSSSE3 (for C/C++) or 01 (for DPC++) and \[Q\]xSSE4.2 (for C/C++) or 02 (for DPC++) as two forms of the same option, where only one form can be used. Since \[Q\]xSSE4.2 (for C/C++) or 02 (for DPC++) are last (furthest to the right), they are used.

All options specified on the command line are used to compile each file. The compiler does not compile individual files with specific options.

A rare exception to this rule is the \(-x\) `type` option:

For C:

```
// Linux
icx -x c file1 -x c++ file2 -x assembler file3
```

For C++:

```
// Linux
icpx -x c file1 -x c++ file2 -x assembler file3
```

For DPC++:

```
// Linux
dpcpp -x c++-header file1 -x c++ file2
```

The `type` argument identifies each file type for the compiler.

**Using Options with Arguments**

Compiler options can be as simple as a single letter, such as the option `E`. Many options accept or require arguments. The `O` option, for example, accepts a single-value argument that the compiler uses to determine the degree of optimization. Other options require at least one argument and can accept multiple arguments. For most options that accept arguments, the compiler warns you if your option and argument are not recognized. If you specify `O9`, the compiler issues a warning, ignores the unrecognized option `O9`, and proceeds with the compilation.

The `O` option does not require an argument, but there are other options that must include an argument. The `I` option requires an argument that identifies the directory to add to the include file search path. If you use this option without an argument, the compiler will not finish its compilation.

**Other Forms of Options**

You can toggle some options on or off by using the negation convention. For example, the `[Q]ipo` option (and many others) includes negation forms, `-no-ipo` (Linux) and `/Qipo-` (Windows), to change the state of the option.

**Option Categories**

When you invoke the Intel oneAPI DPC++/C++ Compiler and specify a compiler option, you have a wide range of choices to influence the compiler's default operation. Intel oneAPI DPC++/C++ Compiler options typically correspond to one or more of the following categories:

- Advanced Optimization
- Code Generation
- Compatibility
- Compiler Diagnostics
- Component Control This content is specific to C++; it does not apply to DPC++.
• Data
• Floating Point This content is specific to C++; it does not apply to DPC++.
• Inlining
• Interprocedural Optimizations (IPO)
• Language
• Linking/Linker
• Miscellaneous
• OpenMP® and Parallel Processing
• Optimization
• Output
• Preprocessor

This content is specific to C++; it does not apply to DPC++.

To see the included options in each category, invoke the compiler from the command line with the `help` category option. For example:

```bash
//Linux
icx -help codegen

//Windows
icx /help codegen
```

The help option prints to `stdout` with the names and syntax of the options found in the Code Generation category.

**See Also**
Using Configuration files

### Specifying Include Files

The Intel® oneAPI DPC++/C++ Compiler searches the default system areas for include files and items specified by the `-I` compiler option. The compiler searches directories for include files in the following order:

1. Directories specified by the `-I` option
2. Directories specified in the environment variables
3. Default include directories

Use the `X` (Windows*) or `-nostdinc` (Linux*) option to remove the default directories from the include file search path.

For example, to direct the compiler to search the path `/alt/include` instead of the default path, do the following:

For C/C++ projects:

```bash
// Linux
icpx -X -I/alt/include prog1.cpp

// Windows
icx /X /I\alt\include prog1.cpp
```

For DPC++ projects:

```bash
// Linux
dpcpp -nostdinc -I/alt/include prog1.cpp

// Windows
dpcpp-cl /X /I\alt\include prog1.cpp
```

**See Also**

- `I` compiler option
- `X` compiler option
- Supported Environment Variables
Specifying Object Files

You can use the /Fo option (Windows*) or -c and -o options (Linux*) to specify an alternate name for an object file. In this example, the compiler generates an object file name myobj.obj (Windows) or myobj.o (Linux).

For C/C++ projects:

// Linux icpx -c -omyobj.o x.cpp
// Windows icx /Fomyobj x.cpp

For DPC++ projects:

// Linux
dpcpp -c -omyobj.o x.cpp
// Windows
dpcpp-cl /Fomyobj x.cpp

See Also

/Fo  compiler option
-c  compiler option
-o  compiler option

Specifying Assembly Files

You can use the /Fa option (Windows*) or -S and -o options (Linux*) to specify an alternate name for an assembly file. The compiler generates an assembly file named myasm.asm (Windows) or myasm.s (Linux).

For C/C++ projects:

// Linux icpx -S -omyasm.s x.cpp
// Windows icx /Fomyasm x.cpp

For DPC++ projects:

// Linux
dpcpp -S -omyasm.s x.cpp
// Windows
dpcpp-cl /Famyasm x.cpp

See Also

-S  compiler option
-o  compiler option
/Fa  compiler option

Converting Projects to Use a Selected Compiler from the Command Line

You can use the command-line interface ICProjConvert<version>.exe to transform your Intel® C++ projects into Microsoft Visual C++* projects, or vice versa. The syntax is:

ICProjConvert<version>.exe <sln_file | prj_files> </VC[:"VCtoolset name"] | /IC[:"ICtoolset name"]> [/q] [/nologo] [/msvc] [/s] [/f]

Where:
### Parameter | Description
--- | ---
version | The ICProjConvert version number. Values are: 191 or 192.
sln_file | A path to the solution file, which should be modified to use a specified project system.
prj_files | A space separated list of project files (or wildcard), which should be modified to use specified project system.
/VC | Convert to use the Microsoft Visual C++ project system.
VCtoolset name | The possible values are v141 (Microsoft Visual Studio* 2017), v142 (Microsoft Visual Studio 2019), or v143 (Microsoft Visual Studio 2022).

**NOTE** Support for Microsoft Visual Studio 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

/IC | Convert to use the Intel® C++ project system.
ICtoolset name | Such as Intel C++ Compiler 2021.1
Depending on the integration version, the supported name values may be different.
/q | Starts quiet mode, all information messages (except errors) are hidden.
/nologo | Suppresses the startup banner.
/msvc | Sets the compiler to Microsoft Visual C++.
/s | Searches the project files through all subdirectories.
/f | Forces an update to the project even if it has an unsupported type or unsupported properties.
/? or /h | Shows help.

### Example
Issue the command ICProjConvert<version>.exe *.icproj /s /VC to convert all Intel® C++ project files in the current directory and its subdirectories to use Microsoft Visual C++.

**NOTE** If you uninstall the Intel® oneAPI DPC++/C++ Compiler, ICProjConvert<version>.exe remains in the folder Program Files (x86)\Common Files\Intel\shared files\ia32\Bin and you can use it to transform Intel® C++ projects back into Microsoft Visual C++.

### Using Eclipse*

The Intel® oneAPI DPC++/C++ Compiler for Linux* provides integrations for the compiler to Eclipse* and C/C++ Development Tooling* (CDT) that let you develop, build, and debug your Intel oneAPI DPC++/C++ Compiler projects in an integrated development environment (IDE).
Eclipse is an open source software development project dedicated to providing a robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools. It is an extensible, open source integrated development environment (IDE). CDT is a complete C/C++ IDE for the Eclipse platform, which allows you to develop, build, and run projects in a visual, interactive environment. CDT is layered on Eclipse and provides a C/C++ development environment perspective.

**NOTE** Eclipse and CDT are not bundled with DPC++. They must be obtained separately.

### Adding the Compiler to Eclipse*

This step is needed only if you are manually installing the Intel® oneAPI DPC++/C++ Compiler plug-in for Eclipse*.

To add the Intel oneAPI DPC++/C++ Compiler product extension to your Eclipse configuration:

1. Start Eclipse.
2. Select **Help > Install New Software**.
3. Next to the **Work with** field, click the **Add** button. The **Add Repository** dialog box opens.
4. Click the **Archive** button and browse to the `<install_dir>/compiler/<version>/linux/ide_support` directory. Select the `.zip` file that starts with `com.intel.compiler` for C/C++ or `com.intel.dpcpp.compiler` for DPC++, then click **OK**.
5. Select **Intel® Software Development Tools > Intel® C++ Compiler Integration** for C/C++ or **Intel® oneAPI DPC++ Compiler Integration > Intel® oneAPI DPC++ Compiler Integration** for DPC++, then click **OK**.
6. Follow the installation instructions.
7. When asked if you want to restart Eclipse, select **Yes**.

When Eclipse restarts, you can create and work with CDT projects that use the Intel oneAPI DPC++/C++ Compiler.

### Multi-Version Compiler Support

You can select different versions of the Intel® oneAPI DPC++/C++ Compiler for compiling projects with the Eclipse* Integrated Development Environment (IDE). For a list of the currently supported compiler versions by platform, refer to the Release Notes.

If multiple versions of the compiler are installed on the system, Eclipse uses the latest version by default. To select the version of the compiler to build your project:

1. Right click the project and open **Properties**.
2. In the properties dialog box, select **C/C++ Build > Settings**.
3. Select the **Intel(R) oneAPI DPC++ Compiler** for a DPC++ project, or the **Intel® C++ Compiler** for a C++ project tab.
4. Select the row with the desired compiler version.
5. Click **Use Selected**. Alternatively, click **Use Latest** to select the latest version of compiler.
6. Click **Apply**.

The corresponding compiler environment is configured automatically for your project.

Use **Settings** and **Tool Chain Editor** to select tools to be used within the toolchain, or set distinct project properties, like compiler options, to be used with different versions of the compiler.

For any project, you can set the compiler environment by specifying it within Eclipse; this overrides any other environment specifications for the compiler.

### Using Cheat Sheets

The following topic applies to Eclipse* for C/C++.
The Intel® oneAPI DPC++/C++ Compiler integration includes several Eclipse* cheat sheets that can guide you through various compilation and debugging tasks.

To view a list of available cheat sheets and select one:

2. Select a cheat sheet. Cheat sheets located outside of the Eclipse* integration can be entered in the Select a cheat sheet from a file or Enter the URL of a cheat sheet. Intel cheat sheets are located under Intel(R) C++ Compiler. A description of the cheat sheet appears in the lower pane.
3. To open a cheat sheet, click OK.

The Cheat Sheets view opens in the Eclipse window.

Creating a Simple Project

The topics in this section will show you how to create a simple project using Eclipse*.

Creating a New Project

To create a new Eclipse* project:

1. Select File > New > Project... The New Project wizard opens.
2. Expand the C/C++ Project tab and select the appropriate project type. Click Next to continue.
3. For Project name, enter hello_world. Deselect the Use default location to specify a directory for the new project.
4. In the Project Type list, expand the Executable project type and select Hello World C++ Project for C++ or Hello World DPC++ Project for DPC++.
5. In the Toolchains list, select Intel(R) oneAPI DPC++ Compiler for a DPC++ project, or the Intel C++ Compiler for a C++ project. Click Next.

NOTE
- If you need to see the toolchains for the compilers that are not locally installed, uncheck Show project types and toolchains only if they are supported on the platform. You are only able to view and configure these toolchains if the proper compilers are installed.
- If you have multiple versions of the compiler installed, they appear in the project’s properties under C/C++ Build > Settings on the Intel(R) oneAPI DPC++ Compiler tab for a DPC++ project, or the Intel C++ Compiler tab for a C++ project.

6. The Basic Settings page allows specifying template information, including Author and Copyright notice, which appear as a comment at the top of the generated source file. After entering desired fields, click Next.
7. The Select Configurations page allows specifying deployment platforms and configurations. By default, a Debug and Release configuration is created for the selected toolchain. Select no (Deselect all), multiple, or all (Select all) configurations. To edit project properties, click the Advanced settings button. Click Finish to create the hello_world project.

NOTE Configurations can be created after the project is created by selecting Project > Properties.

8. If the view is not the C/C++ Development Perspective (default), an Open Associated Perspective dialog box opens. In the C/C++ Perspective, click Yes to proceed.

An entry for your hello_world project appears in the Project Explorer view.

Adding a C Source File

The following topic applies to Eclipse* for C/C++.
To add a source file to the hello_world project:

1. Select the hello_world project in the Project Explorer view.

   NOTE The dialog box automatically populates the source folder for the source file to be created. You can change this by entering a new location or selecting Browse.

4. Select a Template from the drop-down list or Configure a new template.
5. Click Finish to add the file to the hello_world project.
6. In the Editor view, add your code for new_source_file.c.
7. When your code is complete, Save your file.

See Also
Building a Project

Setting Options for a Project or File
You can specify compiler, linker, and archiver options at the project and source file level. Follow these steps to set options for a project or file:

1. Right-click a project or source file in the Project Explorer.
2. Select Properties. The property pages dialog box opens.
3. Select C/C++ Build > Settings.
4. Select the Tool Settings tab and click an option category for Intel C Compiler, Intel C++ Compiler, or Intel C++ Linker for a C++ project, or select Intel(R) oneAPI DPC++ Compiler or Intel(R) oneAPI DPC++ Linker for a DPC+++ project.
5. Set the options to apply to the project or file.

   NOTE
   • Some properties use check boxes, drop-down boxes, or dialog boxes to specify compiler options. For a description on options properties, hover over the option to display a tooltip. After setting the desired options in command line syntax, select Apply.
   • To specify an option that is not available from the Properties dialog, use C/C++ Build Settings > Settings > <Compiler> > Command Line. The Compiler entry has one of the following values: or Intel(R) oneAPI DPC++ Compiler. Enter the command line options in the Additional Options field using command-line syntax and select Apply.
   • You can specify option settings for one or more configurations by using the Configuration drop-down menu.

6. Click Apply and Close.

The compiler applies the selected options, using the selected configurations, when building.

Tip To restore default settings to all properties for the selected configuration, click the Restore Defaults button on any property page.

Excluding Source Files from a Build
The following topic applies to Eclipse* for C/C++.

To exclude a source file from a build:

1. Right-click a file or folder in the Project Explorer.
2. Select Resource Configurations > Exclude from build. The Exclude from build dialog box opens.
3. Select one or more build configurations to exclude the file or folder from.
4. Click OK.

The compiler excludes that file or folder when it builds using the selected project configuration.

**Building a Project**

To build your project:

1. Select the hello_world project in the Project Explorer view.
2. Select Project > Build Project.

See the Build results in the Console view.

For a C/C++ project, use:

```plaintext
**** Build of configuration Debug for project hello_world ****
make all
Building file: ../src/hello_world.cpp
Invoking: Intel C++ Compiler
icpx -g -O0 -MMD -MP -MF"src/hello_world.d" -MT"src/hello_world.d" -c -o "src/hello_world.o"
"../src/hello_world.cpp"
Finished building: ../src/hello_world.cpp

Building target: hello_world
Invoking: Intel C++ Linker
icpx -O0 -o "hello_world" ./src/hello_world.o
Finished building target: hello_world

Build Finished. 0 errors, 0 warnings.
```

For a DPC++ project, use:

```plaintext
**** Build of configuration Debug for project DPCPPhelloworld ****
make all
Building file: ../main.cpp
Invoking: Intel(R) oneAPI DPC++ Compiler
dcpp -g -Wall -O0 -I/home/sys_idebuilder/eclipse-workspace/DPCPPhelloworld -MMD -MP -c -o "main.o" "../main.cpp"
Finished building: ../main.cpp

Building target: DPCPPhelloworld
Invoking: Linker
dcpp -o "DPCPPhelloworld" ./main.o -lsycl -lOpenCL
Finished building target: DPCPPhelloworld

Build Finished. 0 errors, 0 warnings.
```

Detailed descriptions of errors, warnings, and other output can be viewed by selecting the Problems tab.

**Running a Project**

After building a project, you can run your project by following these steps:

1. Select the hello_world project in the Project Explorer view.
2. Select Run As > Local C/C++ Application.

When the executable runs, the output appears in the Console view.
Error Parser

The Error Parser (selected by default) lets you track compile-time errors in Eclipse*/CDT. To confirm that the Error Parser is active:

1. Select the hello_world project in the Project Explorer view.
2. Select Project > Properties.
3. In the Properties dialog box, select C/C++ Build > Settings.
4. Click the Error Parsers tab. Make sure that Intel C++ Error Parser is checked, and CDT Visual C Error Parser or Microsoft Visual C Error Parser are not checked.
5. Click OK to update your choices, if you have changed any settings.

Using the Error Parser

The Error Parser automatically detects and manages the diagnostics generated by the Intel® oneAPI DPC++/C++ Compiler.

If an error occurring in the hello_world.c program is compiled, for example, #include <xstdio.h>, the error is reported in the Problems view next to an error marker.

You can double-click on each error in the Problems view to highlight the source line, which causes the error in the Editor view.

Correct the error, then rebuild your project.

Make Files

This section provides information about makefile project types and exporting makefiles.

Project Types and Makefiles

When creating a new Intel® C++ project in Eclipse*, Executable, Shared Library, Static Library, or Makefile project types are available for selection.

- Select Makefile Project if the project already includes a makefile.
- Use Executable, Shared Library, or Static Library Project to build a makefile using options assigned from property pages specific to the Intel® oneAPI DPC++/C++ Compiler.

Exporting Makefiles

Eclipse* can build a makefile that includes Intel® oneAPI DPC++/C++ Compiler options for created Executables, Shared Libraries, or Static Library Projects. When the project is complete, export the makefile and project source files to another directory, and then build the project from the command line using make.

Exporting makefiles

To export the makefile:

1. Select the project in the Eclipse Project Explorer view.
2. Select File > Export to launch the Export Wizard. The Export dialog box opens, showing the Select screen.
3. Select General > File system, then click Next. The File System screen opens.
4. Check both the hello_world and Release directories in the left-hand pane. Ensure all project sources are checked in the right-hand pane.
NOTE Some files in the right-hand pane may be deselected, such as the **hello_world.o** object file and the **hello_world** executable. **Create directory structure for files** in the **Options** section must be selected to successfully create the export directory. This process applies to project files in the **hello_world** directory.

5. Use the **Browse** button to target the export to an existing directory. Eclipse can create a directory for full paths entered in the **To directory** text box. For example, if the `/code/makefile` is specified as the export directory, Eclipse creates two new subdirectories:

- `/code/makefile/hello_world`
- `/code/makefile/hello_world/Release`

6. Click **Finish** to complete the export.

**Running make**

In a terminal window, change to the `/cpp/hello_world/Release` directory, then run `make` by typing: `make clean all`.

You should see the following output:

For C++:

```
rm -rf  ./new_source_file.o  ./new_source_file.d  hello_world
Building file: ../new_source_file.c
Invoking: Intel C++ Compiler
new_source_file.c"
Finished building: ../new_source_file.c

Building target: hello_world
Invoking: Intel C++ compiler
icx  -o "hello_world"  ./new_source_file.o
Finished building target: hello_world
```

For DPC++:

```
rm -rf  ./new_source_file.o  ./new_source_file.d  hello_world
Building file: ../new_source_file.c
Invoking: Intel(R) oneAPI DPC++ Compiler
new_source_file.c"
Finished building: ../new_source_file.c

Building target: hello_world
Invoking: Linker
dpcpp  -o "hello_world"  ./new_source_file.o
Finished building target: hello_world
```

This process generates the **hello_world** executable in the same directory.

**See Also**

*Setting Options for a Project or File*

**Using Intel Libraries with Eclipse**

You can use the compiler with the following Intel Libraries, which that may be included as a part of the product:
To access these libraries in Eclipse®, use the property pages:

1. Select your project.
2. Open the property pages from Project > Properties and select C/C++ Build > Settings.
3. Select Intel C/C++ Compiler > Performance Library Build Components for C++ projects, or Intel® oneAPI DPC++ Compiler > Performance Library Build Components for DPC++ projects.

The Use Intel® oneAPI Data Analytics Library (oneDAL) property allows enabling the library and bringing in the associated headers.

- **None**: Disable Use of oneDAL.
- **Use threaded Intel® oneDAL**: Link using the threaded version of the library.
- **Use non-threaded Intel® oneDAL**: Link using the non-threaded version of the library.

The Use Intel® Integrated Performance Primitives Libraries property provides the following options in a drop-down menu:

- **None**: Disable use of Intel® IPP.
- **Use main libraries set**: Link using the main libraries set.
- **Use non-pic version of libraries**: Link using the version of the libraries that do not have position-independent code.
- **Use main libraries and cryptography library**: Link using main or cryptography libraries.

The Use Intel® oneAPI Math Kernel Library property provides the following options in a drop-down menu:

- **None**: Disables the use of the oneMKL.
- **Use threaded Intel® oneMKL library**: Link using the threaded version of the library.
- **Use non-threaded Intel® oneMKL library**: Link using the non-threaded version of the library.
- **Use Intel® oneMKL Cluster and sequential Intel® oneMKL libraries**: Link using the oneMKL Cluster Edition libraries and the sequential oneMKL libraries.

**NOTE** The option value **Use Intel® oneMKL Cluster and sequential Intel® oneMKL libraries** is only available for Intel C Compiler or Intel C++ Compiler.

The Use Intel® oneAPI Threading Building Blocks on the Property page allows enabling the library and bringing in the associated headers.

For more information, see the oneDAL, Intel® IPP, oneMKL, and oneTBB documentation.

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Notice revision #20201201
Using Microsoft Visual Studio*

You can use the Intel® oneAPI DPC++/C++ Compiler within the Microsoft Visual Studio* integrated development environment (IDE) to develop C++ or DPC++ applications, including static library (.LIB), dynamic link library (.DLL), and main executable (.EXE) applications. This environment makes it easy to create, debug, and execute programs. You can build your source code into several types of programs and libraries, using the IDE or from the command line.

The IDE offers these major advantages:

- Makes application development quicker and easier by providing a visual development environment.
- Provides integration with the native Microsoft Visual Studio debugger.
- Makes other IDE tools available.

Find Product Information

To access the product information for the Intel® oneAPI DPC++ Compiler:

1. Open Help > About Microsoft Visual Studio
2. In the left pane, select Intel® oneAPI DPC++ Compiler - Package ID: [package ID].
3. In the bottom pane, product details will show: Intel® oneAPI DPC++ Compiler: toolkit version [toolkit version], extension version [extension version], Package ID: [package ID], Copyright © [copyright year] Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

To access the product information for the Intel® C++ Compiler:

1. Open Help > About Microsoft Visual Studio
2. In the left pane, select Intel® C++ Compiler - Package ID: [package ID].
3. In the bottom pane, product details will show: Intel® C++ Compiler: toolkit version [toolkit version], extension version [extension version], Package ID: [package ID], Copyright © [copyright year] Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

To access the product information for the Intel Libraries for oneAPI:

1. Open Help > About Microsoft Visual Studio
2. In the left pane, select Intel Libraries for oneAPI - Package ID: [package ID].
3. In the bottom pane, product details will show: Intel Libraries for oneAPI: toolkit version [toolkit version], extension version [extension version], Package ID: [package ID], Copyright © [copyright year] Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Creating a New Project

Creating a New Project

When you create a project, Microsoft Visual Studio* automatically creates a corresponding solution to contain it. To create a new Intel® oneAPI DPC++/C++ project using Microsoft Visual Studio:

NOTE Exact steps may vary depending on the version of Microsoft Visual Studio in use.

This content is specific to C++; it does not apply to DPC++.

1. Select File > New > Project.
2. In the left pane, expand **Visual C++** and select **Windows Desktop**.
3. In the right pane, select **Windows Console Application**.
4. Accept or specify a project name in the **Name** field. For this example, use hello32 as the project name.
5. Accept or specify the Location for the project directory. Click **OK**.

This content is specific to DPC++.

1. Select **File > New > Project**.
2. In the left pane, expand **DPC++** and select **Console Application**.
3. In the right pane, select **DPC++ Console Application**.
4. Accept or specify a project name in the **Name** field. For this example, use hello_dpcpp as the project name.
5. Accept or specify the Location for the project directory. Click **OK**.

The hello32 (for C++) or hello_dpcpp (for DPC++) project assumes focus in the **Solution Explorer** view. The default Microsoft Visual Studio* solution is also named hello32 (for C++) or hello_dpcpp (for DPC++).

**Using the Intel® oneAPI DPC++/C++ Compiler**

**Using the Intel® oneAPI DPC++/C++ Compiler within Microsoft Visual Studio**

1. Create a Microsoft Visual C++* (MSVC) project, or open an existing project.
2. In **Solution Explorer**, select the project(s) to build with Intel® oneAPI DPC++/C++ Compiler.
3. Open **Project > Properties**.
4. In the left pane, expand the **Configuration Properties** category and select the **General** property page.
5. In the right pane, change the Platform Toolset to `<compiler selection>`.

**NOTE** For DPC++, select **Intel(R) oneAPI DPC++ Compiler** to invoke dpcpp-cl. For C/C++, there are two toolsets: Select **Intel C++ Compiler <major version>** (example 2021) to invoke icx, or select **Intel C++ Compiler <major.minor>** (example 19.2) to invoke icl.

Alternatively, you can change the toolset by selecting **Project > Intel Compiler > Use Intel oneAPI DPC++/C++ Compiler**. This sets whichever version of the compiler that you specify as the toolset for all supported platforms and configurations.

6. To add options, go to **Project > Properties > C/C++ > Command Line** and add new options to the **Additional Options** field.

   Alternatively, you can select options from Intel specific properties. Refer to complete list of options in the Compiler Options section in this documentation.

7. Rebuild, using either **Build > Project only > Rebuild** for a single project, or **Build > Rebuild Solution** for a solution.

**Verify Use of the Intel® oneAPI DPC++/C++ Compiler**

To verify the use of the Intel® oneAPI DPC++/C++ Compiler:

1. Go to **Project > Properties > C/C++ > General**.
2. Set **Suppress Startup Banner** to **No**. Click **OK**.
3. Rebuild your application.
4. Look at the **Output** window.

You should see a message similar to the following when using the Intel® oneAPI DPC++/C++ Compiler:

Intel(R) oneAPI DPC++/C++ Compiler for applications running on XXXX, Version XX.X.X
Unsupported MSVC Project Types
The following project types are not supported:

- Class Library
- CLR Console Application
- CLR Empty Project
- Windows Forms Application
- Windows Forms Control Library

Tips for Ease of Use

- Create a separate configuration for building with Intel® oneAPI DPC++/C++ Compiler:
  - After you have created your project and specified it as an Intel project, create a new configuration (for example, rel_intelc based on Release configuration or debug_intelc based on Debug configuration).
  - Add any special optimization options offered by Intel® oneAPI DPC++/C++ Compiler only to this new configuration (for example, rel_intelc or debug_intelc) through the project property page.
  - Build with Intel® oneAPI DPC++/C++ Compiler.

Build a Project

To build your Intel® oneAPI DPC++/C++ Compiler project:

1. Select your project in the Solution Explorer view.
2. Right-click and select Build Solution or Rebuild Solution to build the solution.

NOTE
If you select Build Project instead of Build Solution (with the value hello32 for C++, or hello_dpcpp for DPC++) in Solution Explorer, you can select Project Only > Build or Project Only > Rebuild to build a single project.

The results of the compilation are displayed in the Output window.

Selecting the Compiler Version

If you have multiple versions of the Intel® oneAPI DPC++/C++ Compiler installed, you can select which version you want from the Compiler Selection dialog box:

1. Select a project, then go to Tools > Options > Intel Compilers and Libraries > <compiler> > Compilers.

   NOTE The <compiler> values are C++ or DPC++.

2. Use the Selected Compiler drop-down menu to select the appropriate version of the compiler.
3. Click OK.

See Also
Use Intel® C++ dialog box

Switching Back to the Visual C++* Compiler

If your project is using the Intel® oneAPI DPC++/C++ Compiler, you can choose to switch back to Microsoft* Visual C++* by doing the following:
1. Select your project.
2. Right-click and select Intel Compiler > Use Visual C++ from the context menu.

Selecting a Configuration

A configuration contains settings that define the final binary output file that you create within a project. It specifies the type of application to build, the platform on which it is to run, and the tool settings to use when building.

Debug and Release Configurations

When you create a new project, Visual Studio* automatically creates the following configurations:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug configuration</td>
<td>By default, the debug configuration sets project options to include the debug symbol information in the debug configuration. It also turns off optimizations. Before you can debug an application, you must build a debug configuration for the project.</td>
</tr>
<tr>
<td>Release (Retail) configuration</td>
<td>The release configuration does <em>not</em> include the debug symbol information, and it uses any optimizations that you have chosen.</td>
</tr>
</tbody>
</table>

Use the Visual Studio* Configuration Manager to select:

1. Release or Debug configuration for the active solution.
2. Release or Debug configuration for any project within the active solution.
3. Target platform for each project.

To make configuration changes for your project:

1. Choose an active solution in the Solution Explorer.
2. Go to Build > Configuration Manager.
3. Select a configuration.

New Configurations

In addition to the default Debug and Release configurations, you can also define new configurations within your project. These configurations may use the existing source files in your project, the existing project settings, or other characteristics of existing configurations.

Specifying a Target Platform

You can specify a target platform for a Microsoft Visual Studio* solution or an individual project within a solution.

1. Select a solution or project in the Solution Explorer.
2. Select Build > Configuration Manager.
   - Use the Active solution platform drop-down list to specify the target platform for the whole solution.
   - Use the Platform column to specify the target platform for an individual project within a solution.
3. Select <New...> on the Active solution platform drop-down menu to add a platform to the current list of active solution platforms.
4. Select or type a new platform in the New Solution Platform dialog box.
5. In Copy settings from choose a platform to use as a template or choose <Empty>.
6. If you want the platform to be set for the projects within the solution, select Create new project platforms. Click OK.
7. Close the **Configuration Manager**.

**NOTE** You can also change target platforms from the Microsoft Visual Studio* toolbar by selecting a platform from the **Solution Platforms** drop-down selection.

### Removing Target Platforms

You can remove a target platform from the **Configuration Manager**:

1. Select `<Edit...>` from the **Active solution platform** list of options.
2. In the **Edit Solutions Platforms** dialog that opens, select a platform and click **Remove**.
3. Click **Yes** in the confirmation box.

### Specifying Directory Paths

To change path locations in Microsoft Visual Studio*:

1. Go to **Project > Properties**. Expand the **Configuration Properties** category and select **VC++ Directories**.
2. In the left pane, select **Configuration Properties > VC++ Directories**.
3. In the right pane, edit the directory paths.

### Specifying a Base Platform Toolset with the Intel® oneAPI DPC++/C++ Compiler

By default, when your project uses the Intel® oneAPI DPC++/C++ Compiler, the Base Platform Toolset property is set to use that compiler with the build environment. This environment includes paths, libraries, included files, etc., of the toolset specific to the version of Microsoft Visual Studio* you are using.

You can set the general project level property **Base Platform Toolset** to use one of the non-Intel installed platform toolsets as the base.

For example, if you are using Microsoft Visual Studio 2019, and you selected the Intel® oneAPI DPC++/C++ Compiler in the Platform Toolset property, then the Base Platform Toolset uses the Microsoft Visual Studio 2019 environment (**v142**). If you want to use other environments from Microsoft Visual Studio along with the Intel® oneAPI DPC++/C++ Compiler, set the **Base Platform Toolset** property to:

- **v141** for Microsoft Visual Studio 2017
- **v142** for Microsoft Visual Studio 2019
- **v143** for Microsoft Visual Studio 2022

**NOTE**

Support for Microsoft Visual Studio 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

This property displays all installed toolsets, not including Intel toolsets.

To set the Base Platform Toolset:

- Using property pages:
  1. Select the project and open **Project > Properties**.
  2. In the left pane, select **Configuration Properties > General**.
  3. In the right pane, find **Intel Specific > Base Platform Toolset**.
  4. Select a value from the pop-up menu.
• Using the **msbuild.exe** command line tool, use the `/p:PlatformToolset` and `/p:BasePlatformToolset` options.

  Example: When the Platform Toolset property is already set to use the Intel® oneAPI DPC++/C++ Compiler, to build a project using the Microsoft Visual Studio 2019 environment use the following command:

  ```
  Msbuild.exe myproject.vcxproj /p:BasePlatformToolset=v142
  ```

  Example: To set the Platform Toolset property to use the Intel® oneAPI DPC++/C++ Compiler and build a project using the Microsoft Visual Studio 2019 environment use the following command:

  ```
  Msbuild.exe myproject.vcxproj /p:PlatformToolset="Intel C++ Compiler 2021" /p:BasePlatformToolset=v142
  ```

  For possible values for the `/p:BasePlatformToolset` property, see the properties described above.

The next time you build your project with the Intel® oneAPI DPC++/C++ Compiler, the option you selected is used to specify the build environment.

### Using Property Pages

The Intel® oneAPI DPC++/C++ Compiler includes support for Property Pages to manage both Intel-specific and general compiler options.

To set compiler options in Microsoft Visual Studio*:

1. Right-click on a project or source file in the **Solution Explorer** view.
2. Select **Properties** from the pop-up menu.
3. In the **Property Pages** dialog box, expand the C/C++ (for C++), or DPC++ (for DPC++) section to view the categories of compiler options.
4. Click **OK** to complete your selection.

The option you selected is used in the compilation the next time you build your project.

### Using Intel Libraries with Microsoft Visual Studio*

You can use the compiler with the following Intel Libraries, which may be included as a part of the product:

- Intel® oneAPI Data Analytics Library (oneDAL)
- Intel® Integrated Performance Primitives (Intel® IPP)
- Intel® oneAPI Threading Building Blocks (oneTBB)
- Intel® oneAPI Math Kernel Library (oneMKL)

Use the property pages to specify Intel Libraries to use with the selected project configuration. The functionality supports Intel® C++, Intel® oneAPI DPC++, and Microsoft Visual C++* project types.

To specify Intel Libraries, select **Project > Properties**. In **Configuration Properties**, select **Intel Libraries for oneAPI**, then do the following:

1. To use **oneDAL** change the **Use oneDAL** settings as follows:
   - **No**: Disable Use of oneDAL.
   - **Default Linking Method**: Use parallel dynamic oneDAL libraries.
   - **Multi-threaded Static Library**: Use parallel static oneDAL libraries.
   - **Single-threaded Static Library**: Use sequential static oneDAL libraries.
   - **Multi-threaded DLL**: Use parallel dynamic oneDAL libraries.
   - **Single-threaded DLL**: Use sequential dynamic oneDAL libraries.

2. To use **Intel® Integrated Performance Primitives**, change the **Use Intel® IPP** settings as follows:
   - **No**: Disable use of Intel® IPP libraries.
   - **Default Linking Method**: Use dynamic Intel® IPP libraries.
   - **Static Library**: Use static Intel® IPP libraries.
   - **Dynamic Library**: Use dynamic Intel® IPP libraries.

3. To use **oneTBB** in your project, change the **Use oneTBB** settings as follows:
   - **No**: Disable use of oneTBB libraries.
• **Use oneTBB**: Set to Yes to use oneTBB in the application.
• **Instrument for use with Analysis Tools**: Set to Yes to analyze your release mode application (not required for debug mode).

4. To use oneMKL in your project, change the **Use oneMKL** property settings as follows:
   • **No**: Disable use of oneMKL libraries.
   • **Parallel**: Use parallel oneMKL libraries.
   • **Sequential**: Use sequential oneMKL libraries.
   • **Cluster**: Use cluster libraries.

   The target platform of an Intel® oneAPI DCP++ project is set to **x64**, so a final selection appears: **Use ILP64 interfaces**. If selected, the corresponding ilp oneMKL libraries are added to the linker command line. Additionally, the MKL_ILP64 preprocessor definition is added to the compiler command line. If you do not make this selection, the Ip oneMKL libraries are used.

Additional settings for use with the Microsoft Visual C++ Platform Toolset are available on the **Intel Libraries for oneAPI** category, found at **Tools > Options**.

For more information, see the Intel® oneAPI Data Analytics Library, Intel® Integrated Performance Primitives, Intel® oneAPI Threading Building Blocks, and Intel® oneAPI Math Kernel Library documentation.

### Product and Performance Information

| Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex). |
| Notice revision #20201201 |

### Changing the Selected Intel Libraries for oneAPI

If you have installed multiple versions of the Intel Libraries for oneAPI, you can specify which version to use with the Microsoft Visual C++* compiler. To do this:

1. Select **Tools > Options**.
2. In the left pane, select **Intel Compilers and Libraries > Intel Libraries for oneAPI**.
3. Select the desired library version from the drop-down boxes in the right pane.

For more information, see the Intel® oneAPI Data Analytics Library (oneDAL), Intel® Integrated Performance Primitives (Intel® IPP), Intel® oneAPI Threading Building Blocks (oneTBB), and Intel® oneAPI Math Kernel Library (oneMKL) documentation.

### Including MPI Support

To specify the type of MPI support you want:

1. Open the project’s property pages and select **Configuration Properties > Intel Libraries for oneAPI**.
2. Set the property **Use oneMKL** to **Cluster**.
3. Set the property **Use MPI Library** to one of the following values:
   • **Intel® MPI Library**
   • **MS-MPI**
4. Build the project.

The next time you build your project with the Intel® oneAPI DPC++/C++ Compiler or Microsoft Visual C++* compiler, it will include support for the version of MPI that you specified.

### Performing Parallel Project Builds

Visual Studio* provides a parallel project build feature, allowing you to build multiple projects within a solution simultaneously, using separate threads. The Visual Studio* IDE initially sets the number of parallel project builds to equal the number of CPUs. To change this setting, do the following:

1. Choose **Tools > Options > Projects and Solutions**.
2. In the **Build and Run** property page, change the number in maximum number of parallel project builds and click **OK**.

For more information on using this feature, see the Microsoft MSDN* documentation.

**Dialog Box Help**

This section provides information about access to dialog boxes and information about compilers, libraries, and converter dialog boxes.

**Options: Compilers dialog box**

To access the **Compilers** page:

1. Open **Tools > Options**.
2. In the left pane, select **Intel Compilers and Libraries > C++ > Compilers for icx** or **Intel Compilers and Libraries > DPC++ > Compilers for dpcpp-cl**.

**Compiler Selection for C++**

**Target Platform:** Select your target platform.

**Platform Toolset/Selected Compiler:** Select your compiler for your platform toolset. The left column lists the platform toolset names. The right column lists combo boxes, which are used to select a compiler. The default value for all combo boxes in current table is **<Latest>**.

| NOTE | The left column contains Intel® C++ Compiler Classic and Intel® oneAPI DPC++/C++ Compiler options. The **Intel C++ Compiler <major.minor>** (example 19.2) selects the Intel® C++ Compiler Classic (icc). The **Intel C++ Compiler <major>** (example 2021) selects the Intel® oneAPI DPC++/C++ Compiler (icx). |

**Default Options:** Sets the default options for a selected compiler. You may specify this setting for each selected compiler.

**Environment:** Sets custom environment variables. You may specify this setting for each selected compiler.

| NOTE | The Environment selection is only available for icx. |

**Compiler Information:** Shows the detail description of the selected compiler.

**Reset All:** Sets all contents back to the default value on the dialog.

**Compiler Selection for DPC++**

**Platform Toolset/Selected Compiler:** Select your compiler for your platform toolset. The left column lists the platform toolset names. The right column lists combo boxes, which are used to select a compiler. The default value for all combo boxes in current table is **<Latest>**.

**Default Options:** Sets the default options for a selected compiler. You may specify this setting for each selected compiler.

**Environment:** Sets custom environment variables. You may specify this setting for each selected compiler.

| NOTE | The Environment selection is only available for icx. |
**Compiler Information:** Shows the detail description of the selected compiler.

**Reset All:** Sets all contents back to the default value on the dialog.

**See Also**
Specifying Directory Paths

**Options: Intel Libraries for oneAPI dialog box**
Use the Intel Libraries for oneAPI page to specify standalone library versions to use with the Microsoft Visual C++* compiler.

To access the Intel Libraries for oneAPI page:
1. Open **Tools > Options**.
2. Select **Intel Compilers and Libraries > Intel Libraries for oneAPI**.
3. Select the desired library version from the drop-down box with one of the following values:
   - oneDAL
   - Intel IPP
   - oneTBB
   - oneMKL
   - Reset All: Use the latest libraries (default)

**NOTE** To enable or disable the Intel Libraries for oneAPI, use the property pages located in the Configuration Properties category.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

**See Also**
Changing the Selected Intel® Performance Libraries
Using Intel® Performance Libraries

**Use Intel® oneAPI DPC++/C++ Compiler dialog box**
To access the Use Intel oneAPI DPC++/C++ Compiler dialog box, select one or more files in the Solution Explorer, right-click and select **Intel Compiler > Use Intel oneAPI DPC++/C++ Compiler for selected file(s)...**

Use this dialog box to change the compiler for one or more selected files to the Intel® oneAPI DPC++/C++ Compiler.

**Select the configuration(s) to update:** Select the desired configuration. Choose from Active configuration or All configurations. If you select All configurations, the compiler is changed in all configurations for the currently selected file(s).

**Select the Platform Toolset:** Select the desired toolset, if multiple platform toolsets are installed.

**See Also**
Using the Intel® oneAPI DPC++/C++ Compiler
Options: Converter dialog box

To access the Converter page, click Tools > Options and then select Intel Compilers and Libraries > C++ > Converter.

Use the Converter page to specify which platform toolset to use when upgrading an Intel® C++ solution (.icproj) from an older version of Microsoft Visual Studio® to a C++ project supported by Microsoft Visual Studio 2017 or later (.vcxproj). Once a solution is upgraded, the .icproj file is not used and can be deleted.

NOTE Support for Microsoft Visual Studio 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

Win32: Select the desired compiler version to be used when converting projects based on IA-32 architecture.

X64: Select the desired compiler version to be used when converting projects based on x64 architecture.

Reset All: Click this button to use the default platform toolsets.
This section contains compiler reference information. For example, it contains information about compiler options, compiler limits, and libraries.

C/C++/DPC++ Calling Conventions

There are a number of calling conventions that set the rules on how arguments are passed to a function and how the values are returned from the function.

Calling Conventions on Windows*

The following table summarizes the supported calling conventions on Windows:

<table>
<thead>
<tr>
<th>Calling Convention</th>
<th>Compiler Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__cdecl</td>
<td>/Gd</td>
<td>This is the default calling convention for C/C++/DPC++ programs. It can be specified on a function with variable arguments.</td>
</tr>
<tr>
<td>__stdcall</td>
<td>/Gz</td>
<td>Standard calling convention used for Win32 API functions. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>__fastcall</td>
<td>/Gr</td>
<td>Fast calling convention that specifies that arguments are passed in registers rather than on the stack. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>__regcall</td>
<td>/Qregcall</td>
<td>Intel® oneAPI DPC++/C++ Compiler calling convention that specifies that as many arguments as possible are passed in registers; likewise, __regcall uses registers whenever possible to return values. This calling convention is ignored if specified on a function with variable arguments.</td>
</tr>
</tbody>
</table>
### Calling Conventions on Linux*

The following table summarizes the supported calling conventions on Linux:

<table>
<thead>
<tr>
<th>Calling Convention</th>
<th>Compiler Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attribute</strong>((cdecl))</td>
<td>none</td>
<td>Default calling convention for C/C++/DPC++ programs. Can be specified on a function with variable arguments.</td>
</tr>
<tr>
<td><strong>attribute</strong>((stdcall))</td>
<td>none</td>
<td>Calling convention that specifies the arguments are passed on the stack. Cannot be specified on a function with variable arguments.</td>
</tr>
<tr>
<td><strong>attribute</strong>((regparm (number)))</td>
<td>none</td>
<td>On systems based on IA-32 architecture, the regparm attribute causes the compiler to pass up to number arguments in registers EAX, EDX, and ECX instead of on the stack. Functions that take a variable number of arguments will continue to pass all of their arguments on the stack.</td>
</tr>
<tr>
<td><strong>attribute</strong>((regcall))</td>
<td>-regcall</td>
<td>Intel oneAPI DPC++/C++ Compiler calling convention that specifies that as many arguments as possible are passed in registers; likewise, __regcall uses registers whenever possible to return values. This calling convention is ignored if specified on a function with variable arguments.</td>
</tr>
</tbody>
</table>

---

For more information about the Intel-compatible vector functions ABI, download the Vector Function Application Binary Interface PDF.

For more information about the GCC vector functions ABI, see the item Libmvec - vector math library document in the GLIBC wiki at sourceware.org.
### Calling Convention

<table>
<thead>
<tr>
<th>Calling Convention</th>
<th>Compiler Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__attribute__((vectorcall))</code></td>
<td>none</td>
<td>Calling convention that specifies that a function passing vector type arguments should utilize vector registers.</td>
</tr>
</tbody>
</table>

### The `__regcall` Calling Convention

The `__regcall` calling convention is unique to the Intel oneAPI DPC++/C++ Compiler and requires some additional explanation.

To use `__regcall`, place the keyword before a function declaration. For example:

**Example**

```c
__regcall int foo (int i, int j);
```

// Linux
```
__attribute__((regcall)) foo (int I, int j);
```

### Available `__regcall` Registers

All registers in a `__regcall` function can be used for parameter passing/returning a value, except those that are reserved by the compiler. The following table lists the registers that are available in each register class depending on the default ABI for the compilation. The registers are used in the order shown below.

This content is specific to C++; it does not apply to DPC++.

<table>
<thead>
<tr>
<th>Register Class/Architecture</th>
<th>IA-32 for Linux</th>
<th>IA-32 for Windows</th>
<th>Intel® 64 for Linux</th>
<th>Intel® 64 for Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPR</td>
<td>EAX, ECX, EDX, EDI, EDI</td>
<td>ECX, EDX, EDI, EDI</td>
<td>RAX, RCX, RDX, RDI, RSI, R8, R9, R10, R11, R12, R14, R15</td>
<td>RAX, RCX, RDX, RDI, RSI, R8, R9, R11, R12, R14, R15</td>
</tr>
<tr>
<td>FP</td>
<td>ST0</td>
<td>ST0</td>
<td>ST0</td>
<td>ST0</td>
</tr>
<tr>
<td>MMX</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>XMM</td>
<td>XMM0 - XMM7</td>
<td>XMM0 - XMM7</td>
<td>XMM0 - XMM15</td>
<td>XMM0 - XMM15</td>
</tr>
<tr>
<td>YMM</td>
<td>YMM0 - YMM7</td>
<td>YMM0 - YMM7</td>
<td>YMM0 - YMM15</td>
<td>YMM0 - YMM15</td>
</tr>
<tr>
<td>ZMM</td>
<td>ZMM0 - ZMM7</td>
<td>ZMM0 - ZMM7</td>
<td>ZMM0 - YMM15</td>
<td>ZMM0 - YMM15</td>
</tr>
</tbody>
</table>

This content is specific to DPC++.

<table>
<thead>
<tr>
<th>Register Class/Architecture</th>
<th>Intel® 64 for Linux</th>
<th>Intel® 64 for Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPR</td>
<td>RAX, RCX, RDX, RDI, RSI, R8, R9, R10, R11, R12, R14, R15</td>
<td>RAX, RCX, RDX, RDI, RSI, R8, R9, R11, R12, R14, R15</td>
</tr>
<tr>
<td>FP</td>
<td>ST0</td>
<td>ST0</td>
</tr>
<tr>
<td>MMX</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>XMM</td>
<td>XMM0 - XMM15</td>
<td>XMM0 - XMM15</td>
</tr>
</tbody>
</table>
Register Class/Architecture | Intel® 64 for Linux | Intel® 64 for Windows
---|---|---
YMM | YMM0 - YMM15 | YMM0 - YMM15
ZMM | ZMM0 - YMM15 | ZMM0 - YMM15

__regcall Data Type Classification
Parameters and return values for __regcall are classified by data type and are passed in the registers of the classes shown in the following table.

**NOTE** All types assigned to XMM, YMM, or ZMM in a non-SSE target are passed in the stack.

This content is specific to C++; it does not apply to DPC++.

<table>
<thead>
<tr>
<th>Type (for both unsigned and signed types)</th>
<th>IA-32</th>
<th>Intel® 64</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool, char, int, enum, _Decimal32, long, pointer</td>
<td>GPR</td>
<td>GPR</td>
</tr>
<tr>
<td>short, __mmask{8,16,32,64}</td>
<td>GPR</td>
<td>GPR</td>
</tr>
<tr>
<td>long long, __int64</td>
<td>See Structured Data Type Classification Rules</td>
<td>GPR</td>
</tr>
<tr>
<td>_Decimal64</td>
<td>XMM</td>
<td>GPR</td>
</tr>
<tr>
<td>long double</td>
<td>FP</td>
<td>FP</td>
</tr>
<tr>
<td>float, double, float128, _Decimal128</td>
<td>XMM</td>
<td>XMM</td>
</tr>
<tr>
<td>__m128, __m128i, __m128d</td>
<td>XMM</td>
<td>XMM</td>
</tr>
<tr>
<td>__m256, __m256i, __m256d</td>
<td>YMM</td>
<td>YMM</td>
</tr>
<tr>
<td>__m512, __m512i, __m512d</td>
<td>ZMM</td>
<td>ZMM</td>
</tr>
<tr>
<td>complex type, struct, union</td>
<td>See Structured Data Type Classification Rules</td>
<td>See Structured Data Type Classification Rules</td>
</tr>
</tbody>
</table>

**NOTE** For the purpose of structured types, the classification of GPR class is used.

**NOTE** On systems based on IA-32 architecture, these 64-bit integer types (long long, __int64) get classified to the GPR class and are passed in two registers, as if they were implemented as a structure of two 32-bit integer fields.

This content is specific to DPC++.
### Type (for both unsigned and signed types)

<table>
<thead>
<tr>
<th>Type (for both unsigned and signed types)</th>
<th>Intel® 64</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool, char, int, enum, _Decimal32, long, pointer</td>
<td>GPR</td>
</tr>
<tr>
<td>short, __mmask{8,16,32,64}</td>
<td>GPR</td>
</tr>
<tr>
<td>long long, __int64</td>
<td>GPR</td>
</tr>
<tr>
<td>_Decimal64</td>
<td>GPR</td>
</tr>
<tr>
<td>long double</td>
<td>FP</td>
</tr>
<tr>
<td>float, double, float128, _Decimal128</td>
<td>XMM</td>
</tr>
<tr>
<td>__m128, __m128i, __m128d</td>
<td>XMM</td>
</tr>
<tr>
<td>__m256, __m256i, __m256d</td>
<td>YMM</td>
</tr>
<tr>
<td>__m512, __m512i, __m512d</td>
<td>ZMM</td>
</tr>
<tr>
<td>complex type, struct, union</td>
<td>See Structured Data Type Classification Rules</td>
</tr>
</tbody>
</table>

NOTE: For the purpose of structured types, the classification of GPR class is used.

Types that are smaller in size than registers than registers of their associated class are passed in the lower part of those registers; for example, float is passed in the lower four bytes of an XMM register.

---

### __regcall Structured Data Type Classification Rules

Structures/unions and complex types are classified similarly to what is described in the x86_64 ABI, with the following exceptions:

- There is no limitation on the overall size of a structure.
- The register classes for basic types are given in Data Type Classifications.
- For systems based on the IA-32 architecture, classification is performed on four-bytes. For systems based on other architectures, classification is performed on eight-bytes.

**NOTE** This content is specific to C++; it does not apply to DPC++.

- Classification is performed on eight-bytes.

**NOTE** This content is specific to DPC++.

---

### __regcall Placement in Registers or on the Stack

After the classification described in Data Type Classifications and Structured Data Type Classification Rules, __regcall__ parameters and return values are either put into registers specified in Available Registers or placed in memory, according to the following:

- Each chunk (eight bytes on systems based on Intel® 64 architecture or four-bytes on systems based on IA-32 architecture (IA-32 is for C++ only) of a value of Data Type is assigned a register class. If enough registers from Available Registers are available, the whole value is passed in registers, otherwise the value is passed using the stack.
• If the classification were to use one or more register classes, then the registers of these classes from the table in Available Registers are used, in the order given there.
• If no more registers are available in one of the required register classes, then the whole value is put on the stack.

__regcall Registers that Preserve Their Values

The following registers preserve their values across a __regcall call, as long as they were not used for passing a parameter or returning a value:

This content is specific to C++; it does not apply to DPC++.

<table>
<thead>
<tr>
<th>Register Class/ABI</th>
<th>IA-32</th>
<th>Intel® 64 for Linux</th>
<th>Intel® 64 for Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPR</td>
<td>ESI, EDI, EBX, EBP, ESP</td>
<td>R12 - R15, RBX, RBP, RSP</td>
<td>R12 - R15, RBX, RBP, RSP</td>
</tr>
<tr>
<td>FP</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>MMX</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>XMM</td>
<td>XMM4 - XMM7</td>
<td>XMM8 - XMM15</td>
<td>XMM8 - XMM15</td>
</tr>
<tr>
<td>YMM</td>
<td>XMM4 - XMM7</td>
<td>XMM8 - XMM15</td>
<td>XMM8 - XMM15</td>
</tr>
<tr>
<td>ZMM</td>
<td>XMM4 - XMM7</td>
<td>XMM8 - XMM15</td>
<td>XMM8 - XMM15</td>
</tr>
</tbody>
</table>

This content is specific to DPC++.

<table>
<thead>
<tr>
<th>Register Class/ABI</th>
<th>Intel® 64 for Linux</th>
<th>Intel® 64 for Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPR</td>
<td>R12 - R15, RBX, RBP, RSP</td>
<td>R12 - R15, RBX, RBP, RSP</td>
</tr>
<tr>
<td>FP</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>MMX</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>XMM</td>
<td>XMM8 - XMM15</td>
<td>XMM8 - XMM15</td>
</tr>
<tr>
<td>YMM</td>
<td>XMM8 - XMM15</td>
<td>XMM8 - XMM15</td>
</tr>
<tr>
<td>ZMM</td>
<td>XMM8 - XMM15</td>
<td>XMM8 - XMM15</td>
</tr>
</tbody>
</table>

All other registers do not preserve their values across this call.

See Also
Structured Data Type Classification Rules
Data Type Classifications
Available Registers

Compiler Options

This compiler supports many compiler options you can use in your applications.

In this section, we provide the following:
• An alphabetical list of compiler options that includes their short descriptions
• A list of deprecated options for DPC++ and lists of deprecated and removed options for C++
• General rules for compiler options and the conventions we use when referring to options
• Details about what appears in the compiler option descriptions
• A description of each compiler option. The descriptions appear under the option's functional category. Within each category, the options are listed in alphabetical order.

Clang compiler options are supported for this compiler. We do not document these options, but you can check dpcpp -help and icx/icpx -help on the command line to see if a particular option is supported. For more information about Clang options, see the Clang documentation. The Clang website is https://clang.llvm.org/.

NOTE
On Windows, two compilers are available: dpcpp and dpcpp-cl.

If you want to use Linux-style option syntax, where options start with -, you should continue to use the dpcpp compiler.

If you want to use Microsoft Visual C++ (MSVC)-compatible option syntax, where options start with /,, you should use the dpcpp-cl compiler.

NOTE macOS* is not supported for the icx/icpx, dpcpp, or dpcpp-cl compilers. For macOS or Xcode* support, visit the icc compiler: Intel® C++ Compiler 19.1 Developer Guide and Reference.

Alphabetical List of Compiler Options

The following table lists all the current compiler options in alphabetical order.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>align</td>
<td>Determines whether variables and arrays are naturally aligned. This content is specific to C++; it does not apply to DPC++ .</td>
</tr>
<tr>
<td>ansi</td>
<td>Enables language compatibility with the gcc option ansi.</td>
</tr>
<tr>
<td>arch</td>
<td>Tells the compiler which features it may target, including which instruction sets it may generate.</td>
</tr>
<tr>
<td>B</td>
<td>Specifies a directory that can be used to find include files, libraries, and executables.</td>
</tr>
<tr>
<td>Bdynamic</td>
<td>Enables dynamic linking of libraries at run time. This content is specific to C++; it does not apply to DPC++ .</td>
</tr>
<tr>
<td>Bstatic</td>
<td>Enables static linking of a user’s library. This content is specific to C++; it does not apply to DPC++ .</td>
</tr>
<tr>
<td>Bsymbolic</td>
<td>Binds references to all global symbols in a program to the definitions within a user’s shared library. This content is specific to C++; it does not apply to DPC++ .</td>
</tr>
<tr>
<td>Bsymbolic-functions</td>
<td>Binds references to all global function symbols in a program to the definitions within a user’s shared library. This content is specific to C++; it does not apply to DPC++ .</td>
</tr>
<tr>
<td>C</td>
<td>Places comments in preprocessed source output.</td>
</tr>
<tr>
<td>c</td>
<td>Prevents linking.</td>
</tr>
<tr>
<td>D</td>
<td>Defines a macro name that can be associated with an optional value.</td>
</tr>
<tr>
<td>dD, QdD</td>
<td>Same as option -dM, but outputs #define directives in preprocessed source.</td>
</tr>
</tbody>
</table>
debug (Linux*)  Enables or disables generation of debugging information. This content is specific to C++; it does not apply to DPC++.

ddebug (Windows*)  Enables or disables generation of debugging information. This content is specific to C++; it does not apply to DPC++.

device-math-lib  Enables or disables certain device libraries. This is a deprecated option that may be removed in a future release. This content is specific to C++; it does not apply to DPC++.

dM, QdM  Tells the compiler to output macro definitions in effect after preprocessing.

ddryrun  Specifies that driver tool commands should be shown but not executed. This content is specific to C++; it does not apply to DPC++.

dumpmachine  Displays the target machine and operating system configuration.

dumpversion  Displays the version number of the compiler.

dynamic-linker  Specifies a dynamic linker other than the default. This content is specific to C++; it does not apply to DPC++.

E  Causes the preprocessor to send output to stdout.

EH  Specifies the model of exception handling to be performed.

EP  Causes the preprocessor to send output to stdout, omitting #line directives.

F (Windows*)  Specifies the stack reserve amount for the program. This content is specific to C++; it does not apply to DPC++.

Fa  Specifies that an assembly listing file should be generated.

FA  Specifies the contents of an assembly listing file.

fasm-blocks  Enables the use of blocks and entire functions of assembly code within a C or C++ file.

fast  Maximizes speed across the entire program. This content is specific to C++; it does not apply to DPC++.

fasyncunwind-tables  Determines whether unwind information is precise at an instruction boundary or at a call boundary.

fbuiltin, Oi  Enables or disables inline expansion of intrinsic functions.

FC  Displays the full path of source files passed to the compiler in diagnostics.

fcommon  Determines whether the compiler treats common symbols as global definitions.

FD  Generates file dependencies related to the Microsoft* C/C++ compiler.

Fd  Lets you specify a name for a program database (PDB) file created by the compiler.

Fe  Specifies the name for a built program or dynamic-link library.

fexceptions  Enables exception handling table generation.
### ffp-contract
Controls when the compiler is permitted to form fused floating-point operations, such as fused multiply-add (FMA). Fused operations are allowed to produce more precise results than performing the individual operations separately.

### ffreestanding, Qfreestanding
Ensures that compilation takes place in a freestanding environment.

### ffunction-sections
Places each function in its own COMDAT section.

### fgnu89-inline
Tells the compiler to use C89 semantics for inline functions when in C99 mode.

### fimf-absolute-error, Qimf-absolute-error
Defines the maximum allowable absolute error for math library function results. This content is specific to C++; it does not apply to DPC++.

### fimf-accuracy-bits, Qimf-accuracy-bits
Defines the relative error for math library function results, including division and square root. This content is specific to C++; it does not apply to DPC++.

### fimf-arch-consistency, Qimf-arch-consistency
Ensures that the math library functions produce consistent results across different microarchitectural implementations of the same architecture. This content is specific to C++; it does not apply to DPC++.

### fimf-domain-exclusion, Qimf-domain-exclusion
Indicates the input arguments domain on which math functions must provide correct results. This content is specific to C++; it does not apply to DPC++.

### fimf-force-dynamic-target, Qimf-force-dynamic-target
Instructs the compiler to use run-time dispatch in calls to math functions. This content is specific to C++; it does not apply to DPC++.

### fimf-max-error, Qimf-max-error
Defines the maximum allowable relative error for math library function results, including division and square root. This content is specific to C++; it does not apply to DPC++.

### fimf-precision, Qimf-precision
Lets you specify a level of accuracy (precision) that the compiler should use when determining which math library functions to use. This content is specific to C++; it does not apply to DPC++.

### fimf-use-svml, Qimf-use-svml
Instructs the compiler to use the Short Vector Math Library (SVML) rather than the Intel® Math Library (LIBM) to implement math library functions.

### finline
Tells the compiler to inline functions declared with __inline and perform C++ inlining.

### finline-functions
Enables function inlining for single file compilation.

### fintelfpga
Lets you perform ahead-of-time (AOT) compilation for the Field Programmable Gate Array (FPGA). This content is specific to DPC++.

### fiopenmp, Qreopenmp
Enables recognition of OpenMP* features, such as parallel, simd, and offloading directives. This is an alternate Linux* option for compiler option qopenmp.

### FI
Tells the preprocessor to include a specified file name as the header file.

### fixed
Causes the linker to create a program that can be loaded only at its preferred base address. This content is specific to C++; it does not apply to DPC++.

### fjump-tables
Determines whether jump tables are generated for switch statements.

### fkeep-static-consts, Qkeep-static-consts
Tells the compiler to preserve allocation of variables that are not referenced in the source.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fm</td>
<td>Tells the linker to generate a link map file. This is a deprecated option. There is no replacement option. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>fma, Qfma</td>
<td>Determines whether the compiler generates fused multiply-add (FMA) instructions if such instructions exist on the target processor.</td>
</tr>
<tr>
<td>fmath-errno</td>
<td>Tells the compiler that errno can be reliably tested after calls to standard math library functions.</td>
</tr>
<tr>
<td>fno-gnu-keywords</td>
<td>Tells the compiler to not recognize typeof as a keyword.</td>
</tr>
<tr>
<td>fno-operator-names</td>
<td>Disables support for the operator names specified in the standard.</td>
</tr>
<tr>
<td>fno-rtti</td>
<td>Disables support for run-time type information (RTTI).</td>
</tr>
<tr>
<td>fno-sycl-libspirv</td>
<td>Disables the check for libspirv (the SPIR-V* tools library). This content is specific to DPC++.</td>
</tr>
<tr>
<td>Fo</td>
<td>Specifies the name for an object file.</td>
</tr>
<tr>
<td>foffload-static-lib</td>
<td>Tells the compiler to link with a fat (multi-architecture) static library. This is a deprecated option that may be removed in a future release. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fomit-frame-pointer, Oy</td>
<td>Determines whether EBP is used as a general-purpose register in optimizations.</td>
</tr>
<tr>
<td>fopenmp</td>
<td>Option -fopenmp is a deprecated option that will be removed in a future release.</td>
</tr>
<tr>
<td>fopenmp-device-lib</td>
<td>Enables or disables certain device libraries for an OpenMP* target.</td>
</tr>
<tr>
<td>fopenmp-target-buffers, Qopenmp-target-buffers</td>
<td>Enables a way to overcome the problem where some OpenMP* offload SPIR-V* devices produce incorrect code when a target object is larger than 4GB.</td>
</tr>
<tr>
<td>fopenmp-targets, Qopenmp-targets</td>
<td>Enables offloading to a specified GPU target if OpenMP* features have been enabled.</td>
</tr>
<tr>
<td>foptimize-sibling-calls</td>
<td>Determines whether the compiler optimizes tail recursive calls.</td>
</tr>
<tr>
<td>Fp</td>
<td>Lets you specify an alternate path or file name for precompiled header files.</td>
</tr>
<tr>
<td>fpack-struct</td>
<td>Specifies that structure members should be packed together.</td>
</tr>
<tr>
<td>fpascal-strings</td>
<td>Tells the compiler to allow for Pascal-style string literals. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>fpermissive</td>
<td>Tells the compiler to allow for non-conformant code.</td>
</tr>
<tr>
<td>fpic</td>
<td>Determines whether the compiler generates position-independent code.</td>
</tr>
<tr>
<td>fpie</td>
<td>Tells the compiler to generate position-independent code. The generated code can only be linked into executables.</td>
</tr>
<tr>
<td>fp-model, fp</td>
<td>Controls the semantics of floating-point calculations.</td>
</tr>
<tr>
<td>fp-speculation, Qfp-speculation</td>
<td>Tells the compiler the mode in which to speculate on floating-point operations.</td>
</tr>
<tr>
<td>freg-struct-return</td>
<td>Tells the compiler to return struct and union values in registers when possible. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fshort-enums</td>
<td>Tells the compiler to allocate as many bytes as needed for enumerated types.</td>
</tr>
<tr>
<td>fstack-protector</td>
<td>Enables or disables stack overflow security checks for certain (or all) routines.</td>
</tr>
<tr>
<td>fstack-security-check</td>
<td>Determines whether the compiler generates code that detects some buffer overruns. This content is specific to C++; it does not apply to DPC++</td>
</tr>
<tr>
<td>fsycl</td>
<td>Enables a program to be compiled as a SYCL* program rather than as plain C++11 program.</td>
</tr>
<tr>
<td>fsycl-add-targets</td>
<td>Lets you add arbitrary device binary images to the fat SYCL* binary when linking. This is a deprecated option that may be removed in a future release. This content is specific to DPC++</td>
</tr>
<tr>
<td>fsycl-dead-args-optimization</td>
<td>Enables elimination of DPC++ dead kernel arguments. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-device-code-split</td>
<td>Specifies a SYCL* device code module assembly. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-device-lib</td>
<td>Enables or disables certain device libraries for a SYCL* target.</td>
</tr>
<tr>
<td>fsycl-device-only</td>
<td>Tells the compiler to generate a device-only binary. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-early-optimizations</td>
<td>Enables LLVM-related optimizations before SPIR-V* generation. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-enable-function-pointers</td>
<td>Enables function pointers and support for virtual functions for DPC++ kernels and device functions. This is an experimental feature. This content is specific to DPC++</td>
</tr>
<tr>
<td>fsycl-explicit-simd</td>
<td>Enables or disables the experimental &quot;Explicit SIMD&quot; SYCL* extension. This is a deprecated option that may be removed in a future release. This content is specific to DPC++</td>
</tr>
<tr>
<td>fsycl-help</td>
<td>Causes help information to be emitted from the device compiler backend. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-host-compiler</td>
<td>Tells the compiler to use the specified compiler for the host compilation of the overall offloading compilation that is performed. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-host-compiler-options</td>
<td>Passes options to the compiler specified by option fsycl-host-compiler. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-id-queries-fit-in-int</td>
<td>Tells the compiler to assume that SYCL ID queries fit within MAX_INT. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-link</td>
<td>Tells the compiler to perform a partial link of device binaries to be used with Field Programmable Gate Array (FPGA). This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-link-targets</td>
<td>Tells the compiler to link only device code. This is a deprecated option that may be removed in a future release. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-max-parallel-link-jobs</td>
<td>Tells the compiler that it can simultaneously spawn up to the specified number of processes to perform actions required to link DPC++ applications. This is an experimental feature. This content is specific to DPC++.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fsycl-targets</td>
<td>Tells the compiler to generate code for specified device targets. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-unnamed-lambda</td>
<td>Enables unnamed SYCL* lambda kernels. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsycl-use-bitcode</td>
<td>Tells the compiler to produce device code in LLVM IR bitcode format into fat objects. This content is specific to DPC++.</td>
</tr>
<tr>
<td>fsyntax-only</td>
<td>Tells the compiler to check only for correct syntax.</td>
</tr>
<tr>
<td>ftrapuv, Qtrapuv</td>
<td>Initializes stack local variables to an unusual value to aid error detection.</td>
</tr>
<tr>
<td>funsigned-char</td>
<td>Change default char type to unsigned.</td>
</tr>
<tr>
<td>fuse-ld</td>
<td>Tells the compiler to use a different linker instead of the default linker (ld).</td>
</tr>
<tr>
<td>fverbose-asm</td>
<td>Produces an assembly listing with compiler comments, including options and version information.</td>
</tr>
<tr>
<td>fvisibility</td>
<td>Specifies the default visibility for global symbols or the visibility for symbols in declarations, functions, or variables. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>fzero-initialized-in-bss, Qzero-initialized-in-bss</td>
<td>Determines whether the compiler places in the DATA section any variables explicitly initialized with zeros.</td>
</tr>
<tr>
<td>g</td>
<td>Tells the compiler to generate a level of debugging information in the object file.</td>
</tr>
<tr>
<td>GA</td>
<td>Enables faster access to certain thread-local storage (TLS) variables.</td>
</tr>
<tr>
<td>gcc-toolchain</td>
<td>Lets you specify the location of the base toolchain.</td>
</tr>
<tr>
<td>Gd</td>
<td>Makes __cdecl the default calling convention.</td>
</tr>
<tr>
<td>gdwarf</td>
<td>Lets you specify a DWARF Version format when generating debug information.</td>
</tr>
<tr>
<td>GF</td>
<td>Enables read-only string-pooling optimization.</td>
</tr>
<tr>
<td>Gm</td>
<td>Enables a minimal rebuild.</td>
</tr>
<tr>
<td>GR</td>
<td>Enables or disables C++ Run Time Type Information (RTTI).</td>
</tr>
<tr>
<td>Gr</td>
<td>Makes __fastcall the default calling convention. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>grecord-gcc-switches</td>
<td>Causes the command line options that were used to invoke the compiler to be appended to the DW_AT_producer attribute in DWARF debugging information.</td>
</tr>
<tr>
<td>GS</td>
<td>Determines whether the compiler generates code that detects some buffer overruns.</td>
</tr>
<tr>
<td>Gs</td>
<td>Lets you control the threshold at which the stack checking routine is called or not called.</td>
</tr>
<tr>
<td>gsplit-dwarf</td>
<td>Creates a separate object file containing DWARF debug information.</td>
</tr>
<tr>
<td>guard</td>
<td>Enables the control flow protection mechanism.</td>
</tr>
<tr>
<td>Gv</td>
<td>Tells the compiler to use the vector calling convention (__vectorcall) when passing vector type arguments.</td>
</tr>
</tbody>
</table>
Gy  Separates functions into COMDATs for the linker. This is a deprecated option. There is no replacement option. This content is specific to C++; it does not apply to DPC++.

GZ  Initializes all local variables. This is a deprecated option. The replacement option is /RTC1. This content is specific to C++; it does not apply to DPC++.

Gz  Makes __stdcall the default calling convention. This content is specific to C++; it does not apply to DPC++.

H, QH  Tells the compiler to display the include file order and continue compilation.

help  Displays a list of supported compiler options in alphabetical order.

I  Specifies an additional directory to search for include files.

I-  Splits the include path.

idirafter  Adds a directory to the second include file search path.

imacros  Allows a header to be specified that is included in front of the other headers in the translation unit.

intel-freestanding  Lets you compile in the absence of a gcc environment. This content is specific to C++; it does not apply to DPC++.

intel-freestanding-target-os  Lets you specify the target operating system for compilation. This content is specific to C++; it does not apply to DPC++.

ipo, Qipo  Enables interprocedural optimization between files.

ipp-link, Qipp-link  Controls whether the compiler links to static or dynamic threaded Intel® Integrated Performance Primitives (Intel® IPP) run-time libraries.

iprefix  Lets you indicate the prefix for referencing directories that contain header files.

iquote  Adds a directory to the front of the include file search path for files included with quotes but not brackets.

isystem  Specifies a directory to add to the start of the system include path.

iwithprefix  Appends a directory to the prefix passed in by -iprefix and puts it on the include search path at the end of the include directories.

iwithprefixbefore  Similar to -iwithprefix except the include directory is placed in the same place as -I command-line include directories.

J  Sets the default character type to unsigned.

Kc++, TP  Tells the compiler to process all source or unrecognized file types as C++ source files. This is a deprecated option. The replacement option for Kc++ is -x c++; the replacement option for /TP is /Tp<file>. This content is specific to C++; it does not apply to DPC++.

L  Tells the linker to search for a specified library when linking.

LD  Specifies that a program should be linked as a dynamic-link (DLL) library.

link  Passes user-specified options directly to the linker at compile time.
Tells the compiler which features it may target, including which instruction set architecture (ISA) it may generate.

M, QM
Tells the compiler to generate makefile dependency lines for each source file.

m32, m64, Qm32, Qm64
Tells the compiler to generate code for a specific architecture.

m80387
Specifies whether the compiler can use x87 instructions.

malign-double
Determines whether double, long double, and long long types are naturally aligned. This option is equivalent to specifying option align. This content is specific to C++; it does not apply to DPC++.

march
Tells the compiler to generate code for processors that support certain features.

masm
Tells the compiler to generate the assembler output file using a selected dialect.

mbranches-within-32B-boundaries, Qbranches-within-32B-boundaries
Tells the compiler to align branches and fused branches on 32-byte boundaries for better performance. This content is specific to C++; it does not apply to DPC++.

mcmodel
Tells the compiler to use a specific memory model to generate code and store data.

mconditional-branch, Qconditional-branch
Lets you identify and fix code that may be vulnerable to speculative execution side-channel attacks, which can leak your secure data as a result of bad speculation of a conditional branch direction. This content is specific to C++; it does not apply to DPC++.

MD
Tells the linker to search for unresolved references in a multithreaded, dynamic-link run-time library.

MD, QMD
Preprocess and compile, generating output file containing dependency information ending with extension .d.

MF, QMF
Tells the compiler to generate makefile dependency information in a file.

MG, QMG
Tells the compiler to generate makefile dependency lines for each source file.

mintrinsic-promote, Qintrinsic-promote
Enables functions containing calls to intrinsics that require a specific CPU feature to have their target architecture automatically promoted to allow the required feature.

MM, QMM
Tells the compiler to generate makefile dependency lines for each source file.

MMD, QMMD
Tells the compiler to generate an output file containing dependency information.

moment-leaf-frame-pointer
Determines whether the frame pointer is omitted or kept in leaf functions.

MP
Tells the compiler to add a phony target for each dependency.

MQ
Changes the default target rule for dependency generation.

mregparm
Lets you control the number registers used to pass integer arguments. This content is specific to C++; it does not apply to DPC++.

MT
Tells the linker to search for unresolved references in a multithreaded, static run-time library. This content is specific to C++; it does not apply to DPC++.
MT, QMT  Changes the default target rule for dependency generation.

mtune, tune  Performs optimizations for specific processors but does not cause extended instruction sets to be used (unlike -march).

multibyte-chars, Qmultibyte-chars  Determines whether multi-byte characters are supported. This content is specific to C++; it does not apply to DPC++.

multiple-processes  Creates multiple processes that can be used to compile large numbers of source files at the same time. This content is specific to C++; it does not apply to DPC++.

nodefaultlibs  Prevents the compiler from using standard libraries when linking.

no-intel-lib, Qno-intel-lib  Disables linking to specified Intel® libraries, or to all Intel® libraries.

no-libgcc  Prevents the linking of certain gcc-specific libraries. This content is specific to C++; it does not apply to DPC++.

nolib-inline  Disables inline expansion of standard library or intrinsic functions. This content is specific to C++; it does not apply to DPC++.

nolibsycl  Disables linking of the SYCL* runtime library. This content is specific to DPC++.

nologo  Tells the compiler to not display compiler version information.

nostartfiles  Prevents the compiler from using standard startup files when linking.

nostdinc++  Do not search for header files in the standard directories for C++, but search the other standard directories.

nostdlib  Prevents the compiler from using standard libraries and startup files when linking.

O  Specifies the code optimization for applications.

o  Specifies the name for an output file.

Od  Disables all optimizations.

Ofast  Sets certain aggressive options to improve the speed of your application.

Os  Enables optimizations that do not increase code size; it produces smaller code size than O2.

Ot  Enables all speed optimizations.

Ox  Enables maximum optimizations.

P  Tells the compiler to stop the compilation process and write the results to a file.

pdbfile  Lets you specify the name for a program database (PDB) file created by the linker. This content is specific to C++; it does not apply to DPC++.

pie  Determines whether the compiler generates position-independent code that will be linked into an executable.

pragma-optimization-level  Specifies which interpretation of the optimization_level pragma should be used if no prefix is specified. This content is specific to C++; it does not apply to DPC++.

print-multi-lib  Prints information about where system libraries should be found.

pthread  Tells the compiler to use pthreads library for multithreading support.
qactypes, Qactypes
Tells the compiler to include the Algorithmic C (AC) data type folder for header searches and link to the AC data types libraries for Field Programmable Gate Array (FPGA) and CPU compilations.

Qcxx-features
Enables standard C++ features without disabling Microsoft* features.

daal, Qdaal
Tells the compiler to link to certain libraries in the Intel® oneAPI Data Analytics Library (oneDAL).

Qinstall
Specifies the root directory where the compiler installation was performed. This content is specific to C++; it does not apply to DPC++.

qipp, Qipp
Tells the compiler to link to the some or all of the Intel® Integrated Performance Primitives (Intel® IPP) libraries.

Qlocation
Specifies the directory for supporting tools. This content is specific to C++; it does not apply to DPC++.

Qlong-double
Changes the default size of the long double data type.

qmkl, Qmkl
Tells the compiler to link to certain libraries in the Intel® oneAPI Math Kernel Library (oneMKL). On Windows systems, you must specify this option at compile time.

qopenmp, Qopenmp
Enables recognition of OpenMP* features and tells the parallelizer to generate multi-threaded code based on OpenMP* directives.

qopenmp-lib, Qopenmp-lib
Lets you specify an OpenMP* run-time library to use for linking.

qopenmp-link
Controls whether the compiler links to static or dynamic OpenMP* run-time libraries.

qopenmp-simd, Qopenmp-simd
Enables or disables OpenMP* SIMD compilation.

qopenmp-stubs, Qopenmp-stubs
Enables compilation of OpenMP* programs in sequential mode.

qopenmp-threadprivate, Qopenmp-threadprivate
Lets you specify an OpenMP* threadprivate implementation. This content is specific to C++; it does not apply to DPC++.

qopt-assume-no-loop-carried-dep, Qopt-assume-no-loop-carried-dep
Lets you set a level of performance tuning for loops. This content is specific to C++; it does not apply to DPC++.

qopt-for-throughput, Qopt-for-throughput
Determines how the compiler optimizes for throughput depending on whether the program is to run in single-job or multi-job mode.

Qoption
Passes options to a specified tool. This content is specific to C++; it does not apply to DPC++.

qopt-multiple-gather-scatter-by-shuffles, Qopt-multiple-gather-scatter-by-shuffles
Enables or disables the optimization for multiple adjacent gather/scatter type vector memory references. This content is specific to C++; it does not apply to DPC++.

qopt-report, Qopt-report
Enables the generation of a YAML file that includes optimization transformation information.

Qpatchable-addresses
Tells the compiler to generate code such that references to statically assigned addresses can be patched.

Qsafeseh
Registers exception handlers for safe exception handling. This content is specific to C++; it does not apply to DPC++.
qtbb, Qtbb  Tells the compiler to link to the Intel® oneAPI Threading Building Blocks (oneTBB) libraries.

regcall, Qregcall  Tells the compiler that the __regcall calling convention should be used for functions that do not directly specify a calling convention.

reuse-exe  Tells the compiler to speed up Field Programmable Gate Array (FPGA) target compile time by reusing a previously compiled FPGA hardware image. This option is useful only when compiling for hardware. This content is specific to DPC++.

RTC  Enables checking for certain run-time conditions.

S  Causes the compiler to compile to an assembly file only and not link.

save-temps, Qsave-temps  Tells the compiler to save intermediate files created during compilation.

shared  Tells the compiler to produce a dynamic shared object instead of an executable.

shared-intel  Causes Intel-provided libraries to be linked in dynamically. This content is specific to C++; it does not apply to DPC++.

shared-libgcc  Links the GNU libgcc library dynamically.

showIncludes  Tells the compiler to display a list of the include files.

sox, Qsox  Tells the compiler to save the compilation options in the executable file.

static  Prevents linking with shared libraries.

static-intel  Causes Intel-provided libraries to be linked in statically. This content is specific to C++; it does not apply to DPC++.

static-libgcc  Links the GNU libgcc library statically.

static-libstdc++  Links the GNU libstdc++ library statically.

std, Qstd  Tells the compiler to conform to a specific language standard.

strict-ansi  Tells the compiler to implement strict ANSI conformance dialect. This content is specific to C++; it does not apply to DPC++.

sysroot  Specifies the root directory where headers and libraries are located.

T  Tells the linker to read link commands from a file.

Tc  Tells the compiler to process a file as a C source file.

TC  Tells the compiler to process all source or unrecognized file types as C source files.

Tp  Tells the compiler to process a file as a C++ source file.

U  Undefines any definition currently in effect for the specified macro.

u (Linux*)  Tells the compiler the specified symbol is undefined.

u (Windows*)  Disables all predefined macros and assertions. This content is specific to C++; it does not apply to DPC++.

undef  Disables all predefined macros.

unroll, Qunroll  Tells the compiler the maximum number of times to unroll loops.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use-intel-optimized-headers, Quse-intel-optimized-headers</td>
<td>Determines whether the performance headers directory is added to the include path search list. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>use-msasm</td>
<td>Enables the use of blocks and entire functions of assembly code within a C or C++ file.</td>
</tr>
<tr>
<td>v</td>
<td>Specifies that driver tool commands should be displayed and executed.</td>
</tr>
<tr>
<td>vd</td>
<td>Enables or suppresses hidden vtordisp members in C++ objects.</td>
</tr>
<tr>
<td>vec, Qvec</td>
<td>Enables or disables vectorization. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>vec-threshold, Qvec-threshold</td>
<td>Sets a threshold for the vectorization of loops. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>version</td>
<td>Tells the compiler to display GCC-style version information.</td>
</tr>
<tr>
<td>vmg</td>
<td>Selects the general representation that the compiler uses for pointers to members.</td>
</tr>
<tr>
<td>vmv</td>
<td>Enables pointers to members of any inheritance type.</td>
</tr>
<tr>
<td>w</td>
<td>Disables all warning messages.</td>
</tr>
<tr>
<td>w, W</td>
<td>Specifies the level of diagnostic messages to be generated by the compiler.</td>
</tr>
<tr>
<td>Wa</td>
<td>Passes options to the assembler for processing.</td>
</tr>
<tr>
<td>Wabi</td>
<td>Determines whether a warning is issued if generated code is not C++ ABI compliant.</td>
</tr>
<tr>
<td>Wall</td>
<td>Enables warning and error diagnostics.</td>
</tr>
<tr>
<td>watch</td>
<td>Tells the compiler to display certain information to the console output window. This content is specific to C++; it does not apply to DPC++.</td>
</tr>
<tr>
<td>Wcomment</td>
<td>Determines whether a warning is issued when /* appears in the middle of a /* */ comment.</td>
</tr>
<tr>
<td>Wdeprecated</td>
<td>Determines whether warnings are issued for deprecated C++ headers.</td>
</tr>
<tr>
<td>Weffc+++, Qeffc++</td>
<td>Enables warnings based on certain C++ programming guidelines.</td>
</tr>
<tr>
<td>Werror, WX</td>
<td>Changes all warnings to errors.</td>
</tr>
<tr>
<td>Werror-all</td>
<td>Causes all warnings and currently enabled remarks to be reported as errors.</td>
</tr>
<tr>
<td>Wextra-tokens</td>
<td>Determines whether warnings are issued about extra tokens at the end of preprocessor directives.</td>
</tr>
<tr>
<td>Wformat</td>
<td>Determines whether argument checking is enabled for calls to printf, scanf, and so forth.</td>
</tr>
<tr>
<td>Wformat-security</td>
<td>Determines whether the compiler issues a warning when the use of format functions may cause security problems.</td>
</tr>
<tr>
<td>WI</td>
<td>Passes options to the linker for processing.</td>
</tr>
<tr>
<td>Wmain</td>
<td>Determines whether a warning is issued if the return type of main is not expected.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Wmissing-declarations</td>
<td>Determines whether warnings are issued for global functions and variables without prior declaration.</td>
</tr>
<tr>
<td>Wmissing-prototypes</td>
<td>Determines whether warnings are issued for missing prototypes.</td>
</tr>
<tr>
<td>Wno-sycl-strict</td>
<td>Disables warnings that enforce strict SYCL* language compatibility.</td>
</tr>
<tr>
<td>Wp</td>
<td>Passes options to the preprocessor.</td>
</tr>
<tr>
<td>Wpointer-arith</td>
<td>Determines whether warnings are issued for questionable pointer arithmetic.</td>
</tr>
<tr>
<td>Wreorder</td>
<td>Tells the compiler to issue a warning when the order of member initializers does not match the order in which they must be executed.</td>
</tr>
<tr>
<td>Wreturn-type</td>
<td>Determines whether warnings are issued when a function is declared without a return type, when the definition of a function returning void contains a return statement with an expression, or when the closing brace of a function returning non-void is reached.</td>
</tr>
<tr>
<td>Wshadow</td>
<td>Determines whether a warning is issued when a variable declaration hides a previous declaration.</td>
</tr>
<tr>
<td>Wsign-compare</td>
<td>Determines whether warnings are issued when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.</td>
</tr>
<tr>
<td>Wstrict-aliasing</td>
<td>Determines whether warnings are issued for code that might violate the optimizer's strict aliasing rules.</td>
</tr>
<tr>
<td>Wstrict-prototypes</td>
<td>Determines whether warnings are issued for functions declared or defined without specified argument types.</td>
</tr>
<tr>
<td>Wtrigraphs</td>
<td>Determines whether warnings are issued if any trigraphs are encountered that might change the meaning of the program.</td>
</tr>
<tr>
<td>Wuninitialized</td>
<td>Determines whether a warning is issued if a variable is used before being initialized.</td>
</tr>
<tr>
<td>Wunknown-pragmas</td>
<td>Determines whether a warning is issued if an unknown #pragma directive is used.</td>
</tr>
<tr>
<td>Wunused-function</td>
<td>Determines whether a warning is issued if a declared function is not used.</td>
</tr>
<tr>
<td>Wunused-variable</td>
<td>Determines whether a warning is issued if a local or non-constant static variable is unused after being declared.</td>
</tr>
<tr>
<td>Wwrite-strings</td>
<td>Issues a diagnostic message if const char * is converted to (non-const) char *.</td>
</tr>
<tr>
<td>X</td>
<td>Removes standard directories from the include file search path.</td>
</tr>
<tr>
<td>x (type option)</td>
<td>All source files found subsequent to -x type will be recognized as a particular type.</td>
</tr>
<tr>
<td>x, Qx</td>
<td>Tells the compiler which processor features it may target, including which instruction sets and optimizations it may generate.</td>
</tr>
<tr>
<td>xHost, QxHost</td>
<td>Tells the compiler to generate instructions for the highest instruction set available on the compilation host processor.</td>
</tr>
<tr>
<td>Xlinker</td>
<td>Passes a linker option directly to the linker.</td>
</tr>
</tbody>
</table>
Xopenmp-target Enables options to be passed to the specified tool in the device compilation tool chain for the target. This compiler option supports OpenMP* offloading.

Xs Passes options to the backend tool. This content is specific to DPC++.

Xsycl-target Enables options to be passed to the specified tool in the device compilation tool chain for the target. This compiler option supports SYCL* offloading. This content is specific to DPC++.

Y- Tells the compiler to ignore all other precompiled header files.

Yc Tells the compiler to create a precompiled header file.

Yu Tells the compiler to use a precompiled header file.

Zc Lets you specify ANSI C standard conformance for certain language features.

Zg Tells the compiler to generate function prototypes. This is a deprecated option. There is no replacement option. This content is specific to C++; it does not apply to DPC++.

Zi, Z7, ZI Tells the compiler to generate full debugging information in either an object (.obj) file or a project database (PDB) file.

Zl Causes library names to be omitted from the object file.

Zp Specifies alignment for structures on byte boundaries.

Zs Tells the compiler to check only for correct syntax.

**Deprecated and Removed Compiler Options**

This topic lists deprecated and removed compiler options and suggests replacement options, if any are available.

Deprecated and removed options for DPC++ and C++ are listed in separate tables. There are currently no removed options for DPC++.

For more information on compiler options, see the detailed descriptions of the individual option descriptions in this section.

**Deprecated Options for DPC++**

Occasionally, compiler options are marked as "deprecated." Deprecated options are still supported in the current release, but they may be unsupported in future releases.

The following table lists options that are currently deprecated.

Note that deprecated options are not limited to this list.

<table>
<thead>
<tr>
<th>Deprecated Linux* and Windows* Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>-foffload-static-lib</td>
<td>None</td>
</tr>
<tr>
<td>-fsycl-add-targets</td>
<td>None</td>
</tr>
<tr>
<td>-fsycl-explicit-simd</td>
<td>None</td>
</tr>
<tr>
<td>-fsycl-link-targets</td>
<td>None</td>
</tr>
</tbody>
</table>
Deprecated Options for C++

Occasionally, compiler options are marked as "deprecated." Deprecated options are still supported in the current release, but they may be unsupported in future releases.

The following two tables list options that are currently deprecated.

Note that deprecated options are not limited to these lists.

<table>
<thead>
<tr>
<th>Deprecated Linux* Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>-daal</td>
<td>-qdaal</td>
</tr>
<tr>
<td>-device-math-lib</td>
<td>None</td>
</tr>
<tr>
<td>-fopenmp</td>
<td>None</td>
</tr>
<tr>
<td>-ipp</td>
<td>-qipp</td>
</tr>
<tr>
<td>-Kc++</td>
<td>-x c++</td>
</tr>
<tr>
<td>-m32</td>
<td>None</td>
</tr>
<tr>
<td>-march=pentiumii</td>
<td>None</td>
</tr>
<tr>
<td>-march=pentiumiii</td>
<td>-march=pentium3</td>
</tr>
<tr>
<td>-mkl</td>
<td>-qmk1</td>
</tr>
<tr>
<td>-msse</td>
<td>Linux* only: None</td>
</tr>
<tr>
<td>-tbb</td>
<td>-qtbb</td>
</tr>
<tr>
<td>-xH</td>
<td>-xSSE4.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deprecated Windows* Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>/device-math-lib</td>
<td>None</td>
</tr>
<tr>
<td>/GX</td>
<td>/EHsc</td>
</tr>
<tr>
<td>/Gy</td>
<td>None</td>
</tr>
<tr>
<td>/GZ</td>
<td>/RTC1</td>
</tr>
<tr>
<td>/H</td>
<td>None</td>
</tr>
<tr>
<td>/Oy</td>
<td>None</td>
</tr>
<tr>
<td>/Qm32</td>
<td>None</td>
</tr>
<tr>
<td>/Qsfalign</td>
<td>None</td>
</tr>
<tr>
<td>/Quse-asm</td>
<td>None</td>
</tr>
<tr>
<td>/QxH</td>
<td>/QxsSSE4.2</td>
</tr>
<tr>
<td>/Ze</td>
<td>None</td>
</tr>
<tr>
<td>/Zg</td>
<td>None</td>
</tr>
</tbody>
</table>
### Removed Options

Some compiler options are no longer supported and have been removed. If you use one of these options, the compiler issues a warning, ignores the option, and then proceeds with compilation.

The following two tables list options that are no longer supported.

Note that removed options are not limited to these lists.

<table>
<thead>
<tr>
<th>Removed Linux* Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A-</td>
<td>-undef</td>
</tr>
<tr>
<td>-0f_check</td>
<td>None</td>
</tr>
<tr>
<td>-c99</td>
<td>-std=c99</td>
</tr>
<tr>
<td>-check-uninit</td>
<td>-check=uninit</td>
</tr>
<tr>
<td>-export</td>
<td>None</td>
</tr>
<tr>
<td>-export-dir</td>
<td>None</td>
</tr>
<tr>
<td>-F</td>
<td>-P</td>
</tr>
<tr>
<td>-falign-stack=mode</td>
<td>None</td>
</tr>
<tr>
<td>-fdiv_check</td>
<td>None</td>
</tr>
<tr>
<td>-fp</td>
<td>-fno-omit-frame-pointer</td>
</tr>
<tr>
<td>-fvisibility=internal</td>
<td>-fvisibility=hidden</td>
</tr>
<tr>
<td>-fwriteable-strings</td>
<td>None</td>
</tr>
<tr>
<td>-gcc-name and -gxx-name</td>
<td>No exact replacement; use -gcc-toolchain</td>
</tr>
<tr>
<td>-guide-profile</td>
<td>None</td>
</tr>
<tr>
<td>-i-dynamic</td>
<td>-shared-intel</td>
</tr>
<tr>
<td>-i-static</td>
<td>-static-intel</td>
</tr>
<tr>
<td>-inline-debug-info</td>
<td>-debug inline-debug-info</td>
</tr>
<tr>
<td>-ipo_obj (and -ipo_obj)</td>
<td>None</td>
</tr>
<tr>
<td>-ipp-link=static-thread</td>
<td>None</td>
</tr>
<tr>
<td>-Knopic, -KNOPIC</td>
<td>-fpic</td>
</tr>
<tr>
<td>-Kpic, -KPIC</td>
<td>-fpic</td>
</tr>
<tr>
<td>-mp</td>
<td>-fp-model</td>
</tr>
<tr>
<td>-no-alias-args</td>
<td>-fargument-noalias</td>
</tr>
<tr>
<td>-no-c99</td>
<td>-std=c89</td>
</tr>
<tr>
<td>-openmp</td>
<td>-qopenmp</td>
</tr>
<tr>
<td>-openmp-lib</td>
<td>-qopenmp-lib</td>
</tr>
<tr>
<td>Removed Linux* Options</td>
<td>Suggested Replacement</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>-openmp-lib legacy</td>
<td>None</td>
</tr>
<tr>
<td>-openmp-link and -qopenmp-link</td>
<td>None</td>
</tr>
<tr>
<td>-openmpP</td>
<td>-qopenmp</td>
</tr>
<tr>
<td>-openmp-profile</td>
<td>None</td>
</tr>
<tr>
<td>-openmp-report</td>
<td>-qopt-report-phase=openmp</td>
</tr>
<tr>
<td>-openmpS</td>
<td>-qopenmp-stubs</td>
</tr>
<tr>
<td>-openmp-stubs</td>
<td>-qopenmp-stubs</td>
</tr>
<tr>
<td>-openmp-task</td>
<td>-qopenmp-task</td>
</tr>
<tr>
<td>-opt-gather-scatter-unroll</td>
<td>None</td>
</tr>
<tr>
<td>-opt-report</td>
<td>-qopt-report</td>
</tr>
<tr>
<td>-opt-streaming-cache-evict</td>
<td>None</td>
</tr>
<tr>
<td>-prefetch</td>
<td>-qopt-prefetch</td>
</tr>
<tr>
<td>-print-sysroot</td>
<td>None</td>
</tr>
<tr>
<td>-prof-format-32</td>
<td>None</td>
</tr>
<tr>
<td>-prof-genx</td>
<td>-prof-gen=srcpos</td>
</tr>
<tr>
<td>-profile-functions</td>
<td>None</td>
</tr>
<tr>
<td>-profile-loops</td>
<td>None</td>
</tr>
<tr>
<td>-profile-loops-report</td>
<td>None</td>
</tr>
<tr>
<td>-qopenmp-report</td>
<td>-qopt-report-phase=openmp</td>
</tr>
<tr>
<td>-qopenmp-task</td>
<td>None</td>
</tr>
<tr>
<td>-gp</td>
<td>-p</td>
</tr>
<tr>
<td>-rct</td>
<td>None</td>
</tr>
<tr>
<td>-shared-libcxa</td>
<td>-shared-libgcc</td>
</tr>
<tr>
<td>-ssp</td>
<td>None</td>
</tr>
<tr>
<td>-static-libcxa</td>
<td>-static-libgcc</td>
</tr>
<tr>
<td>-std=c9x</td>
<td>-std=c99</td>
</tr>
<tr>
<td>-syntax</td>
<td>-fsyntax-only</td>
</tr>
<tr>
<td>-tcheck</td>
<td>None</td>
</tr>
<tr>
<td>-tpp1</td>
<td>None</td>
</tr>
<tr>
<td>-tpp2</td>
<td>None</td>
</tr>
<tr>
<td>Removed Linux* Options</td>
<td>Suggested Replacement</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>-tpp5</td>
<td>None</td>
</tr>
<tr>
<td>-tpp6</td>
<td>None</td>
</tr>
<tr>
<td>-tpp7</td>
<td>None</td>
</tr>
<tr>
<td>-tprofile</td>
<td>None</td>
</tr>
<tr>
<td>-Wpragma-once</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Removed Windows* Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>/debug:parallel</td>
<td>None</td>
</tr>
<tr>
<td>/G5</td>
<td>None</td>
</tr>
<tr>
<td>/G6 (or /GB)</td>
<td>None</td>
</tr>
<tr>
<td>/G7</td>
<td>None</td>
</tr>
<tr>
<td>/GF</td>
<td>/GF</td>
</tr>
<tr>
<td>/ML[d]</td>
<td>Upgrade to /MT[d]</td>
</tr>
<tr>
<td>/Og</td>
<td>/O1, /O2, or /O3</td>
</tr>
<tr>
<td>/Op</td>
<td>/fp:precise</td>
</tr>
<tr>
<td>/QA-</td>
<td>/u</td>
</tr>
<tr>
<td>/Qc99</td>
<td>/Qstd=c99</td>
</tr>
<tr>
<td>/Qguide-profile</td>
<td>None</td>
</tr>
<tr>
<td>/Qgpu-arch:ivybridge</td>
<td>None</td>
</tr>
<tr>
<td>/QIfdiv</td>
<td>None</td>
</tr>
<tr>
<td>/Qinline-debug-info</td>
<td>/debug:inline-debug-info</td>
</tr>
<tr>
<td>/Qipo-obj (and /Qipo_obj)</td>
<td>None</td>
</tr>
<tr>
<td>/Qipp-link:static-thread</td>
<td>None</td>
</tr>
<tr>
<td>/Qmspp</td>
<td>None</td>
</tr>
<tr>
<td>/Qopenmp-lib:legacy</td>
<td>None</td>
</tr>
<tr>
<td>/Qopenmp-link</td>
<td>None</td>
</tr>
<tr>
<td>/Qopenmp-profile</td>
<td>None</td>
</tr>
<tr>
<td>/Qopenmp-report</td>
<td>/Qopt-report-phase:openmp</td>
</tr>
<tr>
<td>/Qopenmp-task</td>
<td>None</td>
</tr>
</tbody>
</table>
### Ways to Display Certain Option Information

This section describes how you can use a certain compiler option to get general information about compiler options.

#### Displaying General Option Information From the Command Line

To display a list of all available compiler options, specify option `help` on the command line.

To display functional groupings of compiler options, specify a functional category for option `help`. For example, to display a list of options that affect diagnostic messages, enter one of the following commands:

- `help diagnostics`  // Linux systems
- `/help diagnostics`  // Windows systems

For details on other categories you can specify, see `help`.

#### Compiler Option Details

This section contains the full details about compiler options, including descriptions of each compiler option.
In this section compiler options are listed within their categories. To see an alphabetical list of compiler options, see Alphabetical List of Compiler Options.

**General Rules for Compiler Options**

This section describes general rules for compiler options and it contains information about how we refer to compiler option names in descriptions.

**General Rules for Compiler Options**

You cannot combine options with a single dash (Linux*) or slash (Windows*). For example:

- On Linux* systems: This form is incorrect: `-Ec`; this form is correct: `--Ec`
- On Windows* systems: This form is incorrect: `/Ec`; this form is correct: `/E /c`

All compiler options are case sensitive. Some options have different meanings depending on their case; for example, option "c" prevents linking, but option "C" places comments in preprocessed source output.

Options specified on the command line apply to all files named on the command line.

Options can take arguments in the form of file names, strings, letters, or numbers. If a string includes spaces, the string must be enclosed in quotation marks. For example:

- On Linux* systems, `-unroll [=n]` or `-U name (string)`
- On Windows* systems, `/Famyfile.s (file name)` or `/V "version 5.0" (string)`

Compiler options can appear in any order.

On Windows* systems, all compiler options must precede `/link` options, if any, on the command line.

Unless you specify certain options, the command line will both compile and link the files you specify.

You can abbreviate some option names, entering as many characters as are needed to uniquely identify the option.

Certain options accept one or more keyword arguments following the option name. For example, the `x` option accepts several keywords.

To specify multiple keywords, you typically specify the option multiple times.

Compiler options remain in effect for the whole compilation unless overridden by a compiler `#pragma`.

To disable an option, specify the negative form of the option.

On Windows* systems, you can also disable one or more optimization options by specifying option `/Od` last on the command line.

**NOTE**

On Windows* systems, the `/Od` option is part of a mutually-exclusive group of options that includes `/Od`, `/O1`, `/O2`, `/O3`, and `/Ox`. The last of any of these options specified on the command line will override the previous options from this group.

If there are enabling and disabling versions of an option on the command line, the last one on the command line takes precedence.

**How We Refer to Compiler Option Names in Descriptions**

The following conventions are used as shortcuts when referencing compiler option names in descriptions:

- Many options have names that are the same on Linux* and Windows*. However, the Windows form starts with an initial `/` and the Linux form starts with an initial `-`. Within text, such option names are shown without the initial character; for example, `check`.
Many options have names that are the same on Linux* and Windows*, except that the Windows form starts with an initial Q. Within text, such option names are shown as [Q]option-name.

For example, if you see a reference to [Q]ipo, the Linux* form of the option is -ipo and the Windows form of the option is /Qipo.

This content is specific to C++; it does not apply to DPC++.

Several compiler options have similar names except that the Linux* forms start with an initial q and the Windows form starts with an initial Q. Within text, such option names are shown as [q or Q]option-name.

For example, if you see a reference to [q or Q]opt-report, the Linux* form of the option is -qopt-report and the Windows form of the option is /Qopt-report.

Compiler option names that are more dissimilar are shown in full.

**What Appears in the Compiler Option Descriptions**

This section contains details about what appears in the option descriptions.

Following sections include individual descriptions of all the current compiler options. The option descriptions are arranged by functional category. Within each category, the option names are listed in alphabetical order.

Each option description contains the following information:

- The primary name for the option and a short description of the option.
- **Architecture Restrictions**
  - This section only appears if there is a known architecture restriction for the option.
  - Restrictions can appear for any of the following architectures:
    - IA-32 architecture (C++ only)
    - Intel® 64 architecture
  - Certain operating systems are not available on all the above architectures. For the latest information, check your Release Notes.
- **Syntax**
  - This section shows the syntax on Linux* systems and the syntax on Windows* systems. If the option is not valid on a particular operating system, it will specify "None".
- **Arguments**
  - This section shows any arguments (parameters) that are related to the option. If the option has no arguments, it will specify "None".
- **Default**
  - This section shows the default setting for the option.
- **Description**
  - This section shows the full description of the option. It may also include further information on any applicable arguments.
- **IDE Equivalent**
  - This section shows information related to the Intel® Integrated Development Environment (Intel® IDE) Property Pages on Linux* and Windows* systems. It shows on which Property Page the option appears, and under what category it's listed. The Windows* IDE is Microsoft* Visual Studio* .NET. If the option has no IDE equivalent, it will specify "None".
- **Alternate Options (C++ only)**
  - This section lists any options that are synonyms for the described option. If there are no alternate option names, it will show "None".

Some alternate option names are deprecated and may be removed in future releases.

Many options have an older spelling where underscores ("_") instead of hyphens ("-") connect the main option names. The older spelling is a valid alternate option name.
Some option descriptions may also have the following:

- **Example (or Examples)**
  This section shows one or more examples that demonstrate the option.
- **See Also**
  This section shows where you can get further information on the option or it shows related options.

**Optimization Options**
This section contains descriptions for compiler options that pertain to optimization.

**fast**
Maximizes speed across the entire program. This content is specific to C++; it does not apply to DPC++.

**Syntax**

- **Linux OS:**
  - `fast`
  ```bash
  Linux OS:
  -fast
  ```

- **Windows OS:**
  - `/fast`

**Arguments**
None

**Default**
OFF The optimizations that maximize speed are not enabled.

**Description**
This option maximizes speed across the entire program.
It sets the following options:
- On Windows* systems: `/O3, /Qipo, /Qprec-div-, /fp:fast=2`
- On Linux* systems: `-ipo, -O3, -static, -fp-model fast=2`

For example:
- On Linux* systems, if you specify option `-fast -xSSE3, option -xSSE3 takes effect.`
- On Windows* systems, if you specify option `/fast /QxSSE3, option/QxSSE3 takes effect.`

**NOTE**
Option `fast` sets some aggressive optimizations that may not be appropriate for all applications. The resulting executable may not run on processor types different from the one on which you compile. You should make sure that you understand the individual optimization options that are enabled by option `fast`.

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).
IDE Equivalent
None
Alternate Options
None
See Also
fp-model, fp compiler option

fbuiltin, Oi
Enables or disables inline expansion of intrinsic functions.

Syntax
Linux OS:
-fbuiltin[-name]
-fno-builtin[-name]

Windows OS:
/Oi[-]
/Qno-builtin-name

Arguments
name Is a list of one or more intrinsic functions. If there is more than one intrinsic function, they must be separated by commas.

Default
ON Inline expansion of intrinsic functions is enabled.

Description
This option enables or disables inline expansion of one or more intrinsic functions.
If -fno-builtin-name or /Qno-builtin-name is specified, inline expansion is disabled for the named functions. If name is not specified, -fno-builtin or /Oi- disables inline expansion for all intrinsic functions.
For a list of built-in functions affected by -fbuiltin, search for "built-in functions" in the appropriate gcc* documentation.
For a list of built-in functions affected by /Oi, search for "/Oi" in the appropriate Microsoft* Visual C/C++* documentation.

IDE Equivalent
Windows
Visual Studio: Optimization > Enable Intrinsic Functions (/Oi)
**Linux**

Eclipse: None

**Alternate Options**

None

**ffunction-sections**

*Places each function in its own COMDAT section.*

**Syntax**

**Linux OS:**

-ffunction-sections

**Windows OS:**

None

**Arguments**

None

**Default**

OFF

**Description**

Places each function in its own COMDAT section.

**IDE Equivalent**

None

**Alternate Options**

-fdata-sections

**foptimize-sibling-calls**

*Determines whether the compiler optimizes tail recursive calls.*

**Syntax**

**Linux OS:**

-foptimize-sibling-calls

-fno-optimize-sibling-calls

**Windows OS:**

None

**Arguments**

None

**Default**

-foptimize-sibling-calls

The compiler optimizes tail recursive calls.
Description
This option determines whether the compiler optimizes tail recursive calls. It enables conversion of tail recursion into loops.

If you do not want to optimize tail recursive calls, specify `-fno-optimize-sibling-calls`.

Tail recursion is a special form of recursion that doesn't use stack space. In tail recursion, a recursive call is converted to a GOTO statement that returns to the beginning of the function. In this case, the return value of the recursive call is only used to be returned. It is not used in another expression. The recursive function is converted into a loop, which prevents modification of the stack space used.

IDE Equivalent
None

Alternate Options
None

GF
*Enables read-only string-pooling optimization.*

Syntax
Linux OS:
None

Windows OS:
/GF

Arguments
None

Default
OFF Read/write string-pooling optimization is enabled.

Description
This option enables read only string-pooling optimization.

IDE Equivalent
Windows
Visual Studio: **Code Generation > Enable String Pooling**

Linux
Eclipse: None

Alternate Options
None

nolib-inline
*Disables inline expansion of standard library or intrinsic functions. This content is specific to C++; it does not apply to DPC++.*
Syntax

Linux OS:
-nolib-inline

Windows OS:
None

Arguments
None

Default
OFF

The compiler inlines many standard library and intrinsic functions.

Description
This option disables inline expansion of standard library or intrinsic functions. It prevents the unexpected results that can arise from inline expansion of these functions.

IDE Equivalent

Windows
Visual Studio: None

Linux
Eclipse: Optimization > Disable Intrinsic Inline Expansion

Alternate Options
None

O
Specifies the code optimization for applications.

Syntax

Linux OS:
-O [n]

Windows OS:
/O [n]

Arguments
n
Is the optimization level. Possible values are 1, 2, or 3. On Linux* systems, you can also specify 0.

Default
O2

Optimizes for code speed. This default may change depending on which other compiler options are specified. For details, see below.

Description
This option specifies the code optimization for applications.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Linux*)</td>
<td>This is the same as specifying 02. Disables all optimizations. This option may set other options. This is determined by the compiler, depending on which operating system and architecture you are using. The options that are set may change from release to release.</td>
</tr>
</tbody>
</table>
| 00 (Linux) | Enables optimizations for speed and disables some optimizations that increase code size and affect speed. To limit code size, this option:  
- Enables global optimization; this includes data-flow analysis, code motion, strength reduction and test replacement, split-lifetime analysis, and instruction scheduling.  
- Disables inlining of some intrinsics. This option may set other options. This is determined by the compiler, depending on which operating system and architecture you are using. The options that are set may change from release to release. The 01 option may improve performance for applications with very large code size, many branches, and execution time not dominated by code within loops. |
| 01 | Enables optimizations for speed. This is the generally recommended optimization level. Vectorization is enabled at 02 and higher levels. This content is specific to C++; it does not apply to DPC++. On systems using IA-32 architecture: Some basic loop optimizations such as Distribution, Predicate Opt, Interchange, multi-versioning, and scalar replacements are performed. This option also enables:  
- Inlining of intrinsics  
- Intra-file interprocedural optimization, which includes:  
  - inlining  
  - constant propagation  
  - forward substitution  
  - routine attribute propagation  
  - variable address-taken analysis  
  - dead static function elimination  
  - removal of unreferenced variables  
- The following capabilities for performance gain:  
  - constant propagation  
  - copy propagation  
  - dead-code elimination  
  - global register allocation  
  - global instruction scheduling and control speculation  
  - loop unrolling  
  - optimized code selection  
  - partial redundancy elimination  
  - strength reduction/induction variable simplification |
| 02 | Enables optimizations for speed. This is the generally recommended optimization level. Vectorization is enabled at 02 and higher levels. This content is specific to C++; it does not apply to DPC++. On systems using IA-32 architecture: Some basic loop optimizations such as Distribution, Predicate Opt, Interchange, multi-versioning, and scalar replacements are performed. This option also enables:  
- Inlining of intrinsics  
- Intra-file interprocedural optimization, which includes:  
  - inlining  
  - constant propagation  
  - forward substitution  
  - routine attribute propagation  
  - variable address-taken analysis  
  - dead static function elimination  
  - removal of unreferenced variables  
- The following capabilities for performance gain:  
  - constant propagation  
  - copy propagation  
  - dead-code elimination  
  - global register allocation  
  - global instruction scheduling and control speculation  
  - loop unrolling  
  - optimized code selection  
  - partial redundancy elimination  
  - strength reduction/induction variable simplification |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• variable renaming</td>
<td></td>
</tr>
<tr>
<td>• exception handling optimizations</td>
<td></td>
</tr>
<tr>
<td>• tail recursions</td>
<td></td>
</tr>
<tr>
<td>• peephole optimizations</td>
<td></td>
</tr>
<tr>
<td>• structure assignment lowering and optimizations</td>
<td></td>
</tr>
<tr>
<td>• dead store elimination</td>
<td></td>
</tr>
</tbody>
</table>

This option may set other options, especially options that optimize for code speed. This is determined by the compiler, depending on which operating system and architecture you are using. The options that are set may change from release to release.

This content is specific to C++; it does not apply to DPC++.

On Linux systems, the `-debug inline-debug-info` option will be enabled by default if you compile with optimizations (option `-O2` or higher) and debugging is enabled (option `-g`).

Many routines in the shared libraries are more highly optimized for Intel® microprocessors than for non-Intel microprocessors.

```
O3
```

Performs `O2` optimizations and enables more aggressive loop transformations such as Fusion, Block-Unroll-and-Jam, and collapsing IF statements.

This option may set other options. This is determined by the compiler, depending on which operating system and architecture you are using. The options that are set may change from release to release.

The `O3` optimizations may not cause higher performance unless loop and memory access transformations take place. The optimizations may slow down code in some cases compared to `O2` optimizations.

The `O3` option is recommended for applications that have loops that heavily use floating-point calculations and process large data sets.

Many routines in the shared libraries are more highly optimized for Intel® microprocessors than for non-Intel microprocessors.

The last `O` option specified on the command line takes precedence over any others.

**IDE Equivalent**

**Windows**

Visual Studio: **Optimization > Optimization**

**Linux**

Eclipse: **General > Optimization Level**

**Alternate Options**

```
O0
```

Linux: None

```
Od
```

Windows: `/Od`

**See Also**

```
Od  compiler option
```
Od
Disables all optimizations.

Syntax
Linux OS:
None
Windows OS:
/Od

Arguments
None

Default
OFF

The compiler performs default optimizations.

Description
This option disables all optimizations. It can be used for selective optimizations, such as a combination of /Od and /Ob1 (disables all optimizations, but enables inlining).

This content is specific to C++; it does not apply to DPC++. On IA-32 architecture, this option sets the /Oy- option.

IDE Equivalent

Visual Studio
Visual Studio: Optimization > Optimization

Eclipse
Eclipse: None

Alternate Options
Linux: -00
Windows: None

See Also
o compiler option (see O0)

Ofast
Sets certain aggressive options to improve the speed of your application.

Syntax
Linux OS:
-Ofast

Windows OS:
None
Arguments
None

Default
OFF  The aggressive optimizations that improve speed are not enabled.

Description
This option improves the speed of your application.
On Linux* systems, this option is provided for compatibility with gcc.

IDE Equivalent
None

Alternate Options
None

See Also
0  compiler option
fast  compiler option
fp-model, fp  compiler option

Os
Enables optimizations that do not increase code size; it produces smaller code size than O2.

Syntax
Linux OS:
-Os

Windows OS:
/Os

Arguments
None

Default
OFF  Optimizations are made for code speed. However, if O1 is specified, Os is the default.

Description
This option enables optimizations that do not increase code size; it produces smaller code size than O2. It disables some optimizations that increase code size for a small speed benefit.
This option tells the compiler to favor transformations that reduce code size over transformations that produce maximum performance.

IDE Equivalent
Visual Studio
Visual Studio: **Optimization > Favor Size or Speed**
Eclipse

Eclipse: None

Alternate Options

None

See Also

- compiler option
- Ot compiler option

Ot

*Enables all speed optimizations.*

Syntax

Linux OS:

None

Windows OS:

/Out

Arguments

None

Default

/Out  Optimizations are made for code speed.

If Oo is specified, all optimizations are disabled. If O1 is specified, Os is the default.

Description

This option enables all speed optimizations.

IDE Equivalent

Windows

Visual Studio: **Optimization > Favor Size or Speed**

Linux

Eclipse: None

Alternate Options

None

See Also

- compiler option
- Os compiler option

Ox

*Enables maximum optimizations.*
Syntax

Linux OS: None

Windows OS: /Ox

Arguments
None

Default OFF The compiler does not enable optimizations.

Description
The compiler enables maximum optimizations by combining the following options:

• /Oi
• /Ot
• C++ only: /Oy

IDE Equivalent

Windows
Visual Studio: Optimization > Optimization

Linux
Eclipse: None

Alternate Options
None

Code Generation Options
This section contains descriptions for compiler options that pertain to code generation.

arch
*Tells the compiler which features it may target, including which instruction sets it may generate.*

Syntax

Linux OS: None

Windows OS: /arch:code

Arguments

*code* Indicates to the compiler a feature set that it may target, including which instruction sets it may generate. Many of the following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (SSSE). Possible values are:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALDERLAKE</td>
<td>May generate instructions for processors that support the specified Intel® processor or microarchitecture code name.</td>
</tr>
<tr>
<td>AMBERLAKE</td>
<td>Keyword ICELAKE is deprecated and may be removed in a future release.</td>
</tr>
<tr>
<td>BROADWELL</td>
<td></td>
</tr>
<tr>
<td>CANNONLAKE</td>
<td></td>
</tr>
<tr>
<td>CASCADELAKE</td>
<td></td>
</tr>
<tr>
<td>COFFEELAKE</td>
<td></td>
</tr>
<tr>
<td>COOPERLAKE</td>
<td></td>
</tr>
<tr>
<td>GOLDMONT</td>
<td></td>
</tr>
<tr>
<td>GOLDMONT-PLUS</td>
<td></td>
</tr>
<tr>
<td>HASWELL</td>
<td></td>
</tr>
<tr>
<td>ICELAKE-CLIENT</td>
<td></td>
</tr>
<tr>
<td>ICELAKE-SERVER</td>
<td></td>
</tr>
<tr>
<td>IVYBRIDGE</td>
<td></td>
</tr>
<tr>
<td>KABYLAKE</td>
<td></td>
</tr>
<tr>
<td>KNL</td>
<td></td>
</tr>
<tr>
<td>KNM</td>
<td></td>
</tr>
<tr>
<td>ROCKETLAKE</td>
<td></td>
</tr>
<tr>
<td>SANDYBRIDGE</td>
<td></td>
</tr>
<tr>
<td>SAPPHIREAPIDS</td>
<td></td>
</tr>
<tr>
<td>SILVERMONT</td>
<td></td>
</tr>
<tr>
<td>SKYLAKE</td>
<td></td>
</tr>
<tr>
<td>SKYLAKE-AVX512</td>
<td></td>
</tr>
<tr>
<td>TIGERLAKE</td>
<td></td>
</tr>
<tr>
<td>TREMONT</td>
<td></td>
</tr>
<tr>
<td>WHISKEYLAKE</td>
<td></td>
</tr>
<tr>
<td>CORE-AVX2</td>
<td>May generate Intel® Advanced Vector Extensions 2 (Intel® AVX2), Intel® AVX, SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.</td>
</tr>
<tr>
<td>CORE-AVX-I</td>
<td>May generate Float-16 conversion instructions and the RDRND instruction, Intel® Advanced Vector Extensions (Intel® AVX), Intel® SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.</td>
</tr>
<tr>
<td>AVX</td>
<td>May generate Intel® Advanced Vector Extensions (Intel® AVX), Intel® SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.</td>
</tr>
<tr>
<td>SSE4.2</td>
<td>May generate Intel® SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.</td>
</tr>
<tr>
<td>SSE4.1</td>
<td>May generate Intel® SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.</td>
</tr>
<tr>
<td>SSSE3</td>
<td>May generate SSSE3 instructions and Intel® SSE3, SSE2, and SSE instructions.</td>
</tr>
<tr>
<td>SSE3</td>
<td>May generate Intel® SSE3, SSE2, and SSE instructions.</td>
</tr>
<tr>
<td>SSE2</td>
<td>May generate Intel® SSE2 and SSE instructions.</td>
</tr>
<tr>
<td>SSE</td>
<td>This option has been deprecated; it is now the same as specifying IA32.</td>
</tr>
</tbody>
</table>
Generates x86/x87 generic code that is compatible with IA-32 architecture. Disables any default extended instruction settings, and any previously set extended instruction settings. It also disables all feature-specific optimizations and instructions.

This value is only available on IA-32 architecture.

**Default**

**SSE2**

The compiler may generate Intel® SSE2 and SSE instructions.

**Description**

This option tells the compiler which features it may target, including which instruction sets it may generate. Code generated with these options should execute on any compatible, non-Intel processor with support for the corresponding instruction set.

Options `/arch` and `/Qx` are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning.

If you specify both the `/Qax` and `/arch` options, the compiler will not generate Intel-specific instructions.

**IDE Equivalent**

**Visual Studio**

Visual Studio: **Code Generation > Enable Enhanced Instruction Set**

**Eclipse**

Eclipse: None

**Xcode**

Xcode: None

**Alternate Options**

None

**See Also**

`x`, `Qx` compiler option

`arch` compiler option

`march` compiler option

`m` compiler option

`m32`, `m64` compiler option

**EH**

*Specifies the model of exception handling to be performed.*

**Syntax**

**Linux OS:**

None
Windows OS:

/EHtype
/EHtype-

Arguments

type Specifies the exception handling model. Possible values are:

  a Specifies the asynchronous C++ exception handling model.

  s Specifies the synchronous C++ exception handling model.

  c Tells the compiler to assume that extern "C" functions do not throw exceptions.

  r Tells the compiler to always generate runtime termination checks for all noexcept functions. IT forces runtime termination checks in all functions that have a noexcept attribute.

If you specify c, you must also specify a or s.

Default

OFF Some exception handling is performed by default.

Description

This option specifies the model of exception handling to be performed.

If you specify the negative form of the option, it disables the exception handling performed by type or the last type if there are two. For example, if you specify /EHsc-, it is interpreted as /EHs.

For more details about option /EH, see the Microsoft documentation.

IDE Equivalent

Windows

Visual Studio: Code Generation > Enable C++ Exceptions

Linux

Eclipse: None

Alternate Options

/EHsc Linux: None

Windows: /GX

See Also

Qsafeeh compiler option
fasynchronous-unwind-tables

Determines whether unwind information is precise at an instruction boundary or at a call boundary.

Syntax

Linux OS:
-fasynchronous-unwind-tables
-fno-asynchronous-unwind-tables

Windows OS:
None

Arguments

None

Default

Intel® 64 architecture:
-fasynchronous-unwind-tables

C++: IA-32 architecture (Linux* only):
-fno-asynchronous-unwind-tables

Description

This option determines whether unwind information is precise at an instruction boundary or at a call boundary. The compiler generates an unwind table in DWARF2 or DWARF3 format, depending on which format is supported on your system.

If -fno-asynchronous-unwind-tables is specified, the unwind table is precise at call boundaries only. In this case, the compiler will avoid creating unwind tables for routines such as the following:

- A C++ routine that does not declare objects with destructors and does not contain calls to routines that might throw an exception.
- A C/C++ or Fortran routine compiled without -fexceptions.
- A C/C++ or Fortran routine compiled with -fexceptions that does not contain calls to routines that might throw an exception.

IDE Equivalent

None

Alternate Options

None

See Also

fexceptions compiler option

fexceptions

Enables exception handling table generation.
Syntax

Linux OS:
- `fexceptions`
- `fno-exceptions`

Windows OS:
None

Arguments
None

Default

```
-fexceptions
Exception handling table generation is enabled. Default for C++.

-fno-exceptions
Exception handling table generation is disabled. Default for C.
```

Description
This option enables exception handling table generation. The `-fno-exceptions` option disables exception handling table generation, resulting in smaller code. When this option is used, any use of exception handling constructs (such as try blocks and throw statements) will produce an error. Exception specifications are parsed but ignored. It also undefines the preprocessor symbol `__EXCEPTIONS`.

IDE Equivalent
None

Alternate Options
None

`fomit-frame-pointer, Oy`
Determine whether EBP is used as a general-purpose register in optimizations.

Architecture Restrictions
Option `/Oy[-]` is only available on IA-32 architecture. IA-32 architecture is only supported for C++.

Syntax

Linux OS:
- `fomit-frame-pointer`
- `fno-omit-frame-pointer`

Windows OS:
`/Oy` (C++ only)
`/Oy-` (C++ only)

Arguments
None
Default

-fomit-frame-pointer  EBP is used as a general-purpose register in optimizations. However, on Linux* systems, the default is -fno-omit-frame-pointer if option -O0 or -g is specified.

C++: or /Oy

C++: On Windows* systems, the default is /Oy- if option /Od is specified.

Description

These options determine whether EBP is used as a general-purpose register in optimizations. Option -fomit-frame-pointer and option /Oy allows this use. Option -fno-omit-frame-pointer and option /Oy- disallows it.

Some debuggers expect EBP to be used as a stack frame pointer, and cannot produce a stack backtrace unless this is so. The -fno-omit-frame-pointer and the /Oy- option directs the compiler to generate code that maintains and uses EBP as a stack frame pointer for all functions so that a debugger can still produce a stack backtrace without doing the following:

- For -fno-omit-frame-pointer: turning off optimizations with -O0
- This content is specific to C++; it does not apply to DPC++.

For /Oy-: turning off /O1, /O2, or /O3 optimizations

The -fno-omit-frame-pointer option is set when you specify option -O0 or the -g option. The -fomit-frame-pointer option is set when you specify option -O1, -O2, or -O3.

This content is specific to C++; it does not apply to DPC++.

The /Oy option is set when you specify the /O1, /O2, or /O3 option. Option /Oy- is set when you specify the /Od option.

Using the -fno-omit-frame-pointer or /Oy- option reduces the number of available general-purpose registers by 1, and can result in slightly less efficient code.

NOTE

For Linux* systems:

There is currently an issue with GCC 3.2 exception handling. Therefore, the compiler ignores this option when GCC 3.2 is installed for C++ and exception handling is turned on (the default).

IDE Equivalent

Windows

Visual Studio: Optimization > Omit Frame Pointers

Linux

Eclipse: Optimization > Provide Frame Pointer

Alternate Options

Linux: -fp (this is a deprecated option)

Windows: None

See Also

momit-leaf-frame-pointer  compiler option
Gd
Makes __cdecl the default calling convention.

Architecture Restrictions
Not available on IA-32 architecture. IA-32 architecture is only supported for C++.

Syntax
Linux OS:
None

Windows OS:
/Gd

Arguments
None

Default
ON
The default calling convention is __cdecl.

Description
This option makes __cdecl the default calling convention.

IDE Equivalent
Windows
Visual Studio: Advanced > Calling Convention

Linux
Eclipse: None

Alternate Options
None

See Also
C C++ Calling Conventions

Gr
Makes __fastcall the default calling convention. This content is specific to C++; it does not apply to DPC++.

Architecture Restrictions
Only available on IA-32 architecture

Syntax
Linux OS:
None
**Windows OS:**

/Gr

**Arguments**

None

**Default**

OFF  The default calling convention is __cdecl

**Description**

This option makes __fastcall the default calling convention.

**IDE Equivalent**

**Windows**

Visual Studio: Advanced > Calling Convention

**Linux**

Eclipse: None

**Alternate Options**

None

**See Also**

C C++ Calling Conventions

**GR**

Enables or disables C++ Run Time Type Information (RTTI).

**Syntax**

**Linux OS:**

None

**Windows OS:**

/GR

/GR-

**Arguments**

None

**Default**

/GR  C++ Run Time Type Information (RTTI) is enabled.

**Description**

This option enables or disables C++ Run Time Type Information (RTTI).

To disable C++ Run Time Type Information (RTTI), specify option /GR-.
IDE Equivalent

Windows
Visual Studio: **Language > Enable Run-Time Type Information**

Linux
Eclipse: None

Alternate Options
None

**guard**
*Enables the control flow protection mechanism.*

Syntax

**Linux OS:**
None

**Windows OS:**

/\guard:keyword

Arguments

**keyword**
Specifies the control flow protection mechanism. Possible values are:

- \texttt{cf[-]}
  - Tells the compiler to analyze control flow of valid targets for indirect calls and to insert code to verify the targets at runtime.
  - To explicitly disable this option, specify /\texttt{guard:cf-}.

Default

**OFF**
The control flow protection mechanism is disabled.

Description

This option enables the control flow protection mechanism. It tells the compiler to analyze control flow of valid targets for indirect calls and inserts a call to a checking routine before each indirect call to verify the target of the given indirect call.

The /\texttt{guard:cf} option must be passed to both the compiler and linker.

Code compiled using /\texttt{guard:cf} can be linked to libraries and object files that are not compiled using the option.

This option has been added for Microsoft compatibility. It uses the Microsoft implementation.

IDE Equivalent

Windows
Visual Studio: **Code Generation > Control Flow Guard**

Linux
Eclipse: None
Alternate Options
None

**Gv**
*Tells the compiler to use the vector calling convention (__vectorcall) when passing vector type arguments.*

Syntax

**Linux OS:**
None

**Windows OS:**
/Gv

Arguments
None

Default
OFF  The default calling convention is __cdecl.

Description
This option tells the compiler to use the vector calling convention (__vectorcall) when passing vector type arguments.
It causes each function in the module to compile as __vectorcall unless the function is declared with a conflicting attribute, or the name of the function is main.
This option has been added for Microsoft compatibility.
For more details about the __vectorcall calling convention, see the Microsoft documentation.

IDE Equivalent

**Windows**
Visual Studio: *Advanced > Calling Convention*

**Linux**
Eclipse: None

Alternate Options
None

See Also
C C++ Calling Conventions

**Gz**
*Makes __stdcall the default calling convention. This content is specific to C++; it does not apply to DPC++.*

Architecture Restrictions
Only available on IA-32 architecture
Syntax
Linux OS:
None
Windows OS:
/Gz

Arguments
None

Default
OFF

The default calling convention is __cdecl.

Description
This option makes __stdcall the default calling convention.

IDE Equivalent
Windows
Visual Studio: Advanced > Calling Convention

Linux
Eclipse: None

Alternate Options
None

See Also
C C++ Calling Conventions

m
Tells the compiler which features it may target, including which instruction set architecture (ISA) it may generate.

Syntax
Linux OS:
-m code

Windows OS:
None

Arguments
code

Indicates to the compiler a feature set that it may target, including which instruction sets it may generate. Many of the following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (SSSE). Possible values are:
avx
May generate Intel® Advanced Vector Extensions (Intel® AVX), SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.

sse4.2
May generate Intel® SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.

sse4.1
May generate Intel® SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.

ssse3
May generate SSSE3 instructions and Intel® SSE3, SSE2, and SSE instructions.

sse3
May generate Intel® SSE3, SSE2, and SSE instructions.

sse2
May generate Intel® SSE2 and SSE instructions.

sse
This setting has been deprecated; it is the same as specifying ia32.

ia32
Generates x86/x87 generic code that is compatible with IA-32 architecture. Disables any default extended instruction settings, and any previously set extended instruction settings. It also disables all feature-specific optimizations and instructions.

This value is only available on Linux* systems using IA-32 architecture.

This compiler option also supports many of the –m option settings available with gcc. For more information on gcc settings for –m, see the gcc documentation.

Default
-msse2
For more information on the default values, see Arguments above.

Description
This option tells the compiler which features it may target, including which instruction sets it may generate.
Code generated with these options should execute on any compatible, non-Intel processor with support for the corresponding instruction set.
For compatibility with gcc, the compiler allows the following options but they have no effect. You will get a warning error, but the instructions associated with the name will not be generated. You should use the suggested replacement options.

<table>
<thead>
<tr>
<th>gcc Compatibility Option</th>
<th>Suggested Replacement Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mfma</td>
<td>-march=core-avx2</td>
</tr>
<tr>
<td>-mbmi,-mavx2,-mlzcnt</td>
<td>-march=core-avx2</td>
</tr>
<tr>
<td>-mmovbe</td>
<td>-march=atom -minstruction=movbe</td>
</tr>
<tr>
<td>-mcrc32,-maes,-mpclmul,-mpopcnt</td>
<td>-march=corei7</td>
</tr>
<tr>
<td>-mvzeroupper</td>
<td>-march=corei7-avx</td>
</tr>
<tr>
<td>-mfsngsbase,-mrdrnd,-mf16c</td>
<td>-march=core-avx-i</td>
</tr>
</tbody>
</table>
IDE Equivalent
None

Alternate Options
None

See Also
x, Qx  compiler option
arch  compiler option
march  compiler option
m32, m64  compiler option

m32, m64, Qm32, Qm64
Tells the compiler to generate code for a specific architecture.

Syntax
Linux OS:
- m32  (C++ only)
- m64

Windows OS:
/Qm32  (C++ only)
/Qm64  (C++ only)

Windows OS:
None  (DPC++ only)

Arguments
None

Default
OFF  The compiler's behavior depends on the host system.

Description
These options tell the compiler to generate code for a specific architecture.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++: -m32</td>
<td>Tells the compiler to generate code for IA-32 architecture.</td>
</tr>
<tr>
<td>-m64</td>
<td>Tells the compiler to generate code for Intel® 64 architecture.</td>
</tr>
<tr>
<td>C++: /Qm32</td>
<td></td>
</tr>
</tbody>
</table>

On Linux* systems, these options are provided for compatibility with gcc.

IDE Equivalent
None
Alternate Options
None

**m80387**
Specifies whether the compiler can use x87 instructions.

**Syntax**

**Linux OS:**
- `-m80387`
- `-mno-80387`

**Windows OS:**
None

**Arguments**
None

**Default**

- `-m80387`
  The compiler may use x87 instructions.

**Description**
This option specifies whether the compiler can use x87 instructions.
If you specify option `-mno-80387`, it prevents the compiler from using x87 instructions. If the compiler is forced to generate x87 instructions, it issues an error message.

**IDE Equivalent**
None

Alternate Options
- `-m[no-]x87`

**march**
Tells the compiler to generate code for processors that support certain features.

**Syntax**

**Linux OS:**
- `-march=processor`

**Windows OS:**
None

**Arguments**

- `processor` 
  Tells the compiler the code it can generate. Possible values are:
alderlake
amberlake
broadwell
cannonlake
cascadelake
coffeelake
cooperlake
goldmont
goldmont-plus
haswell
icelake-client (or icelake)
icelake-server
ivybridge
kabylake
knl
knm
rocketlake
sandybridge
sapphirerapids
silvermont
skylake
skylake-avx512
tigerlake
tremont
whiskeylake
core-avx2

May generate instructions for processors that support the specified Intel® processor or microarchitecture code name.

Keywords knl and silvermont are only available on Linux® systems.

Keyword icelake is deprecated and may be removed in a future release.

corei7-avx

Generates code for processors that support Intel® Advanced Vector Extensions 2 (Intel® AVX2), Intel® AVX, SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.

corei7

Generates code for processors that support Intel® SSE4 Efficient Accelerated String and Text Processing instructions. May also generate code for Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSE3, SSE2, SSE, and SSSE3 instructions.

atom

Generates code for processors that support MOVBE instructions. May also generate code for SSSE3 instructions and Intel® SSE3, SSE2, and SSE instructions.
Generates code for the Intel® Core™ 2 processor family.

Generates code for Intel® Pentium® 4 processors with MMX technology.

Generates code for Intel® Pentium® processors. Value `pentium3` is only available on Linux* systems.

**Default**

If no architecture option is specified, value `pentium4` is used by the compiler to generate code.

**Description**

This option tells the compiler to generate code for processors that support certain features.

Options `-x` and `-march` are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning.

For compatibility, a number of historical processor values are also supported, but the generated code will not differ from the default.

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

**IDE Equivalent**

None

**Alternate Options**

- `-march=pentium3`
  - Linux: `-xSSE`
  - Windows: None

- `-march=pentium4`
  - Linux: `-xSSE2`
  - Windows: None

- `-march=pentium-m`
  - Windows: None

- `-march=core2`
  - Linux: `-xSSSE3`
  - Windows: None

**See Also**

- `x`, `Qx` compiler option
- `arch` compiler option
- `m` compiler option

**masm**

*Tells the compiler to generate the assembler output file using a selected dialect.*
Syntax

Linux OS:
-masm= dialect

Windows OS:
None

Arguments

dialect

Is the dialect to use for the assembler output file. Possible values are:

att
Tells the compiler to generate the assembler output file using AT&T* syntax.

intel
Tells the compiler to generate the assembler output file using Intel syntax.

Default

-masm=att
The compiler generates the assembler output file using AT&T* syntax.

Description

This option tells the compiler to generate the assembler output file using a selected dialect.

IDE Equivalent

None

Alternate Options

None

mbranches-within-32B-boundaries, Qbranches-within-32B-boundaries

Tells the compiler to align branches and fused branches on 32-byte boundaries for better performance. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
-mbranches-within-32B-boundaries
-mno-branches-within-32B-boundaries

Windows OS:
/Qbranches-within-32B-boundaries
/Qbranches-within-32B-boundaries-

Arguments

None
Default

-mno-branches-within-32B-boundaries or /Qbranches-within-32B-boundaries-

Branches and fused branches are not aligned on 32-byte boundaries.

Description

This option tells the compiler to align branches and fused branches on 32-byte boundaries for better performance.

NOTE

When you use this option, it may affect binary utilities usage experience, such as debugability.

IDE Equivalent

None

Alternate Options

None

mconditional-branch, Qconditional-branch

Lets you identify and fix code that may be vulnerable to speculative execution side-channel attacks, which can leak your secure data as a result of bad speculation of a conditional branch direction. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:

-mconditional-branch=keyword

Windows OS:

/Qconditional-branch:keyword

Arguments

keyword    Tells the compiler the action to take. Possible values are:

keep          Tells the compiler to not attempt any vulnerable code detection or fixing. This is equivalent to not specifying the -mconditional-branch option.

pattern-report  Tells the compiler to perform a search of vulnerable code patterns in the compilation and report all occurrences to stderr.

pattern-fix   Tells the compiler to perform a search of vulnerable code patterns in the compilation and generate code to ensure that the identified data accesses are not executed speculatively. It will also report any fixed patterns to stderr.
This setting does not guarantee total mitigation, it only fixes cases where all components of the vulnerability can be seen or determined by the compiler. The pattern detection will be more complete if advanced optimization options are specified or are in effect, such as option `O3` and option `-ipo` (or `/Qipo`).

**all-fix**

Tells the compiler to fix all of the vulnerable code so that it is either not executed speculatively, or there is no observable side-channel created from their speculative execution. Since it is a complete mitigation against Spectre variant 1 attacks, this setting will have the most run-time performance cost.

In contrast to the `pattern-fix` setting, the compiler will not attempt to identify the exact conditional branches that may have led to the mis-speculated execution.

**all-fix-lfence**

This is the same as specifying setting `all-fix`.

**all-fix-cmov**

Tells the compiler to treat any path where speculative execution of a memory load creates vulnerability (if mispredicted). The compiler automatically adds mitigation code along any vulnerable paths found, but it uses a different method then the one used for `all-fix` (or `all-fix-lfence`).

This method uses CMOVcc instruction execution, which constrains speculative execution. Thus, it is used for keeping track of the predicate value, which is updated on each conditional branch.

To prevent Spectre v.1 attack, each memory load that is potentially vulnerable is bitwise ORed with the predicate to mask out the loaded value if the code is on a mispredicted path.

This is analogous to the Clang compiler's option to do Speculative Load Hardening.

This setting is only supported on Intel® 64 architecture-based systems.

**Default**

```
-mconditional-branch=keep
and /Qconditional-branch:keep
```

The compiler does not attempt any vulnerable code detection or fixing.

**Description**

This option lets you identify code that may be vulnerable to speculative execution side-channel attacks, which can leak your secure data as a result of bad speculation of a conditional branch direction. Depending on the setting you choose, vulnerabilities may be detected and code may be generated to attempt to mitigate the security risk.

**IDE Equivalent**

**Visual Studio**

Visual Studio: *Code Generation [Intel C++] > Spectre Variant 1 Mitigation*
**Eclipse**

Eclipse: None

**Alternate Options**

None

**mintrinsic-promote, Qintrinsic-promote**

Enables functions containing calls to intrinsics that require a specific CPU feature to have their target architecture automatically promoted to allow the required feature.

**Syntax**

Linux OS:

- `-mintrinsic-promote`

Windows OS:

`/Qintrinsic-promote`

**Arguments**

None

**Default**

OFF

If this option is not specified and you call an intrinsic that requires a CPU feature not provided by the specified (or default) target processor, an error will be reported.

**Description**

This option enables functions containing calls to intrinsics that require a specific CPU feature to have their target architecture automatically promoted to allow the required feature.

All code within the function will be compiled with that target architecture, and the resulting code for such functions will not execute correctly on processors that do not support the required feature.

You are responsible for guarding the execution path at run time so that such functions are not dynamically reachable when the program is run on processors that do not support the required feature.

**NOTE**

We recommend that you use `__attribute__((target(<required target>)))` to mark functions that are intended to be executed on specific target architectures instead of using this option. This attribute will provide significantly better compile time error checking.

**IDE Equivalent**

None

**Alternate Options**

None

**momit-leaf-frame-pointer**

Determines whether the frame pointer is omitted or kept in leaf functions.
Syntax

Linux OS:
- `momit-leaf-frame-pointer`
- `mno-omit-leaf-frame-pointer`

Windows OS:
None

Arguments
None

Default
Varies

If you specify option `-fomit-frame-pointer` (or it is set by default), the default is `-momit-leaf-frame-pointer`. If you specify option `-fno-omit-frame-pointer`, the default is `-mno-omit-leaf-frame-pointer`.

Description

This option determines whether the frame pointer is omitted or kept in leaf functions. It is related to option `-f[no-]omit-frame-pointer` and the setting for that option has an effect on this option.

Consider the following option combinations:

<table>
<thead>
<tr>
<th>Option Combination</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-fomit-frame-pointer -momit-leaf-frame-pointer</code> or <code>-fomit-frame-pointer -mno-omit-leaf-frame-pointer</code></td>
<td>Both combinations are the same as specifying <code>-fomit-frame-pointer</code>. Frame pointers are omitted for all routines.</td>
</tr>
<tr>
<td><code>-fno-omit-frame-pointer -momit-leaf-frame-pointer</code></td>
<td>In this case, the frame pointer is omitted for leaf routines, but other routines will keep the frame pointer. This is the intended effect of option <code>-momit-leaf-frame-pointer</code>.</td>
</tr>
<tr>
<td><code>-fno-omit-frame-pointer -mno-omit-leaf-frame-pointer</code></td>
<td>In this case, <code>-mno-omit-leaf-frame-pointer</code> is ignored since <code>-fno-omit-frame-pointer</code> retains frame pointers in all routines. This combination is the same as specifying <code>-fno-omit-frame-pointer</code>.</td>
</tr>
</tbody>
</table>

This option is provided for compatibility with gcc.

IDE Equivalent

Windows
Visual Studio: None
Linux
Eclipse: Optimization > Omit frame pointer for leaf routines

Alternate Options
None

See Also
fomit-frame-pointer, Oy compiler option

mregparm
Lets you control the number registers used to pass integer arguments. This content is specific to C++; it does not apply to DPC++.

Architecture Restrictions
Only available on IA-32 architecture

Syntax
Linux OS:
-mregparm=n

Windows OS:
None

Arguments

\( n \)

Specifies the number of registers to use when passing integer arguments. You can specify at most 3 registers. If you specify a nonzero value for \( n \), you must build all modules, including startup modules, and all libraries, including system libraries, with the same value.

Default
OFF
The compiler does not use registers to pass arguments.

Description
Control the number registers used to pass integer arguments. This option is provided for compatibility with gcc.

IDE Equivalent
None

Alternate Options
None

mtune, tune
Performs optimizations for specific processors but does not cause extended instruction sets to be used (unlike -march).
Syntax

Linux OS:
- `mtune=processor`

Windows OS:
/`tune:processor`

Arguments

`processor`

Is the processor for which the compiler should perform optimizations. Possible values are:

- `generic`
  Optimizes code for the compiler's default behavior.

- `alderlake`
  Optimizes code for processors that support the specified Intel® processor or microarchitecture code name.

- `amberlake`

- `broadwell`

- `cannonlake`

- `cascadelake`

- `coffeelake`

- `cooperlake`

- `goldmont`

- `goldmont-plus`

- `haswell`

- `icelake-client` (or `icelake`)
  Keywords `knl` and `silvermont` are only available on Windows* and Linux* systems.
  Keyword `icelake` is deprecated and may be removed in a future release.

- `icelake-server`

- `ivybridge`

- `kabylake`

- `knl`

- `knm`

- `rocketlake`

- `sandybridge`

- `sapphirerapids`

- `silvermont`

- `skylake`

- `skylake-avx512`

- `tigerlake`

- `tremont`

- `whiskeylake`

- `core-avx2`

  Optimizes code for processors that support Intel® Advanced Vector Extensions 2 (Intel® AVX2), Intel® AVX, SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.

- `core-avx-i`

  Optimizes code for processors that support Float-16 conversion instructions and the RDRND instruction, Intel® Advanced Vector Extensions (Intel® AVX), Intel® SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.
<table>
<thead>
<tr>
<th>Optimize for</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>corei7-avx</td>
<td>Optimizes code for processors that support Intel® Advanced Vector Extensions (Intel® AVX), Intel® SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions.</td>
</tr>
<tr>
<td>corei7</td>
<td>Optimizes code for processors that support Intel® SSE4 Efficient Accelerated String and Text Processing instructions. May also generate code for Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSE3, SSE2, SSE, and SSSE3 instructions.</td>
</tr>
<tr>
<td>atom</td>
<td>Optimizes code for processors that support MOVBE instructions. May also generate code for SSSE3 instructions and Intel® SSE3, SSE2, and SSE instructions.</td>
</tr>
<tr>
<td>core2</td>
<td>Optimizes for the Intel® Core™ processor family, including support for MMX™, Intel® SSE, SSE2, SSE3 and SSSE3 instruction sets.</td>
</tr>
<tr>
<td>pentium-mmx</td>
<td>Optimizes for Intel® Pentium® with MMX technology.</td>
</tr>
<tr>
<td>pentiumpro</td>
<td>Optimizes for Intel® Pentium® Pro, Intel Pentium II, and Intel Pentium III processors.</td>
</tr>
<tr>
<td>pentium4m</td>
<td>Optimizes for Intel® Pentium® 4 processors with MMX technology.</td>
</tr>
<tr>
<td>pentium-m</td>
<td>Optimizes code for Intel® Pentium® processors. Value pentium3 is only available on Linux® systems.</td>
</tr>
<tr>
<td>pentium4</td>
<td></td>
</tr>
<tr>
<td>pentium3</td>
<td></td>
</tr>
<tr>
<td>pentium</td>
<td></td>
</tr>
</tbody>
</table>

**Default**

- generic: Code is generated for the compiler's default behavior.

**Description**

This option performs optimizations for specific processors but does not cause extended instruction sets to be used (unlike -march).

The resulting executable is backwards compatible and generated code is optimized for specific processors. For example, code generated with -mtune=core2 or /tune:core2 will run correctly on 4th Generation Intel® Core™ processors, but it might not run as fast as if it had been generated using -mtune=haswell or /tune:haswell. Code generated with -mtune=haswell (/tune:haswell) or -mtune=core-avx2 (/tune:core-avx2) will also run correctly on Intel® Core™ processors, but it might not run as fast as if it had been generated using -mtune=core2 or /tune:core2. This is in contrast to code generated with -march=core-avx2, which will not run correctly on older processors such as Intel® Core™ 2 processors.

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Notice revision #20201201
IDE Equivalent

Windows
Visual Studio: Code Generation [Intel C++] > Intel Processor Microarchitecture-Specific Optimization

Linux
Eclipse: Code Generation > Intel Processor Microarchitecture-Specific Optimization

OS X
Xcode: Code Generation > Intel Processor Microarchitecture-Specific Optimization

Alternate Options

-mtune
Linux: -mcpu (this is a deprecated option)
Windows: None

See Also
march compiler option

Qcxx-features
Enables standard C++ features without disabling Microsoft* features.

Syntax
Linux OS:
None

Windows OS:
/Qcxx-features

Arguments
None

Default
OFF The compiler enables standard C++ features.

Description
This option enables standard C++ features without disabling Microsoft* features within the bounds of what is provided in the Microsoft headers and libraries.
This option has the same effect as specifying /EHsc /GR.

IDE Equivalent
None

Alternate Options
None
**Qpatchable-addresses**

*Tells the compiler to generate code such that references to statically assigned addresses can be patched.*

**Architecture Restrictions**

Only available on Intel® 64 architecture

**Syntax**

**Linux OS:**

None

**Windows OS:**

/Qpatchable-addresses

**Arguments**

None

**Default**

OFF  The compiler does not generate patchable addresses.

**Description**

This option tells the compiler to generate code such that references to statically assigned addresses can be patched with arbitrary 64-bit addresses.

Normally, the Windows® system compiler that runs on Intel® 64 architecture uses 32-bit relative addressing to reference statically allocated code and data. That assumes the code or data is within 2GB of the access point, an assumption that is enforced by the Windows object format.

However, in some patching systems, it is useful to have the ability to replace a global address with some other arbitrary 64-bit address, one that might not be within 2GB of the access point.

This option causes the compiler to avoid 32-bit relative addressing in favor of 64-bit direct addressing so that the addresses can be patched in place without additional code modifications. This option causes code size to increase, and since 32-bit relative addressing is usually more efficient than 64-bit direct addressing, you may see a performance impact.

**IDE Equivalent**

None

**Alternate Options**

None

**Qsafeseh**

*Registers exception handlers for safe exception handling. This content is specific to C++; it does not apply to DPC++.*

**Architecture Restrictions**

Only available on IA-32 architecture
Syntax

Linux OS:
None

Windows OS:
/Qsafeseh[-]

Arguments
None

Default
ON    Exception handlers are enabled for safe exception handling.

Description
Registers exception handlers for safe exception handling. It also marks objects as "compatible with the Registered Exception Handling feature" whether they contain handlers or not. This is important because the Windows linker will only generate the "special registered EH table" if ALL objects that it is building into an image are marked as compatible. If any objects are not marked as compatible, the EH table is not generated.

Digital signatures certify security and are required for components that are shipped with Windows, such as device drivers.

IDE Equivalent
None

Alternate Options
None

See Also
/EH compiler option

/regcall, Qregcall
Tells the compiler that the __regcall calling convention should be used for functions that do not directly specify a calling convention.

Syntax

Linux OS:
-regcall

Windows OS:
/Qregcall

Arguments
None

Default
OFF    The __regcall calling convention will only be used if a function explicitly specifies it.
Description
This option tells the compiler that the __regcall calling convention should be used for functions that do not directly specify a calling convention. This calling convention ensures that as many values as possible are passed or returned in registers.

It ensures that __regcall is the default calling convention for functions in the compilation, unless another calling convention is specified in a declaration.

This calling convention is ignored if it is specified for a function with variable arguments.

Note that all __regcall functions must have prototypes.

IDE Equivalent
None

Alternate Options
None

See Also
C/C++ Calling Conventions

x, Qx
_Tells the compiler which processor features it may target, including which instruction sets and optimizations it may generate._

Syntax

**Linux OS:**
-xcode

**Windows OS:**
/Qxcode

Arguments
code
_Specifies a feature set that the compiler can target, including which instruction sets and optimizations it may generate. Many of the following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (Intel® SSSE). Possible values are:_

- **ALDERLAKE**
- **AMBERLAKE**
- **BROADWELL**
- **CANNONLAKE**
- **CASCADELAKE**
- **COFFEELAKE**
- **COOPERLAKE**
- **GOLDMONT**
- **GOLDMONT-PLUS**
- **HASWELL**
- **ICELAKE-CLIENT (or ICELAKE)**
- **ICELAKE-SERVER**
- **IVYBRIDGE**

- May generate instructions for processors that support the specified Intel® processor or microarchitecture code name. Optimizes for the specified Intel® processor or microarchitecture code name.
- Keywords **KNL** and **SILVERMONT** are only available on Windows* and Linux* systems.
- Keyword **ICELAKE** is deprecated and may be removed in a future release.
**KABYLKE**

**KNL**

**KNM**

**ROCKETLAKE**

**SANDYBRIDGE**

**SAPPHIRERAPIDS**

**SILVERMONT**

**SKYLAKE**

**SKYLAKE-AVX512**

**TIGERLAKE**

**TREMONT**

**WHISKEYLAKE**

**COMMON-AVX512**

May generate Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Foundation instructions, Intel® AVX-512 Conflict Detection Instructions (CDI), as well as the instructions enabled with CORE-AVX2. Optimizes for Intel® processors that support Intel® AVX-512 instructions.

**CORE-AVX512**

May generate Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Foundation instructions, Intel® AVX-512 Conflict Detection Instructions (CDI), Intel® AVX-512 Doubleword and Quadword Instructions (DQI), Intel® AVX-512 Byte and Word Instructions (BWI) and Intel® AVX-512 Vector Length Extensions, as well as the instructions enabled with CORE-AVX2. Optimizes for Intel® processors that support Intel® AVX-512 instructions.

**CORE-AVX2**

May generate Intel® Advanced Vector Extensions 2 (Intel® AVX2), Intel® AVX, SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions for Intel® processors. Optimizes for Intel® processors that support Intel® AVX2 instructions.

**CORE-AVX-I**

May generate Float-16 conversion instructions and the RDRND instruction, Intel® Advanced Vector Extensions (Intel® AVX), Intel® SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions for Intel® processors. Optimizes for Intel® processors that support Float-16 conversion instructions and the RDRND instruction.

**AVX**

May generate Intel® Advanced Vector Extensions (Intel® AVX), Intel® SSE4.2, SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions for Intel® processors. Optimizes for Intel® processors that support Intel® AVX instructions.

**SSE4.2**

May generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions, Intel® SSE4 Vectorizing Compiler and Media Accelerator, and Intel® SSE3, SSE2, SSE, and SSSE3 instructions for Intel® processors. Optimizes for Intel® processors that support Intel® SSE4.2 instructions.
SSE4.1
May generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel® processors. May generate Intel® SSE4.1, SSE3, SSE2, SSE, and SSSE3 instructions for Intel processors that support Intel® SSE4.1 instructions.

ATOM_SSE4.2
May generate MOVBE instructions for Intel® processors, depending on the setting of option -minstruction (Linux*) or /Qinstruction (Windows*). May also generate Intel® SSE4.2, SSE3, SSE2, and SSE instructions for Intel processors. Optimizes for Intel Atom® processors that support Intel® SSE4.2 and MOVBE instructions.
This keyword is only available on Windows* and Linux* systems.

ATOM_SSSE3
May generate MOVBE instructions for Intel® processors, depending on the setting of option -minstruction (Linux*) or /Qinstruction (Windows*). May also generate SSSE3, Intel® SSE3, SSE2, and SSE instructions for Intel processors. Optimizes for Intel Atom® processors that support Intel® SSE3 and MOVBE instructions.
This keyword is only available on Windows* and Linux* systems.

SSSE3
May generate SSSE3 and Intel® SSE3, SSE2, and SSE instructions for Intel® processors. Optimizes for Intel processors that support SSSE3 instructions.

SSE3
May generate Intel® SSE3, SSE2, and SSE instructions for Intel® processors. Optimizes for Intel processors that support Intel® SSE3 instructions.

SSE2
May generate Intel® SSE2 and SSE instructions for Intel® processors. Optimizes for Intel processors that support Intel® SSE2 instructions.

Default
Windows* systems: None
Linux* systems: None
On Windows systems, if neither /Qx nor /arch is specified, the default is /arch:SSE2.
On Linux systems, if neither -x nor -m is specified, the default is -msse2.

Description
This option tells the compiler which processor features it may target, including which instruction sets and optimizations it may generate. It also enables optimizations in addition to Intel feature-specific optimizations. The specialized code generated by this option may only run on a subset of Intel® processors.
The resulting executables created from these option code values can only be run on Intel® processors that support the indicated instruction set.
The binaries produced by these code values will run on Intel® processors that support the specified features.
Do not use code values to create binaries that will execute on a processor that is not compatible with the targeted processor. The resulting program may fail with an illegal instruction exception or display other unexpected behavior.

Compiling the function main() with any of the code values produces binaries that display a fatal run-time error if they are executed on unsupported processors, including all non-Intel processors.

Compiler options m and arch produce binaries that should run on processors not made by Intel that implement the same capabilities as the corresponding Intel® processors.

The -x and /Qx options enable additional optimizations not enabled with options -m or /arch.

On Windows systems, options /Qx and /arch are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning. Similarly, on Linux systems, options -x and -m are mutually exclusive. If both are specified, the compiler uses the last one specified and generates a warning.

**NOTE**

All settings except SSE2 do a CPU check. However, if you specify option -O0 (Linux*) or option /Od (Windows*), no CPU check is performed.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

**IDE Equivalent**

**Visual Studio**

Visual Studio: **Code Generation > Intel Processor-Specific Optimization**

**Eclipse**

Eclipse: **Code Generation > Intel Processor-Specific Optimization**

**Xcode**

Xcode: **Code Generation > Intel Processor-Specific Optimization**

**Alternate Options**

None

**See Also**

arch  compiler option
march  compiler option
m  compiler option

**xHost, QxHost**

_Tells the compiler to generate instructions for the highest instruction set available on the compilation host processor._
Syntax

Linux OS:
- \texttt{xHost}

Windows OS:
/\texttt{QxHost}

Arguments
None

Default

Windows* systems: None
Linux* systems: None
macOS* systems: -xSSSE3

On Windows systems, if neither /Qx nor /arch is specified, the default is /arch:SSE2.

On Linux systems, if neither -x nor -m is specified, the default is -msse2.

Description

This option tells the compiler to generate instructions for the highest instruction set available on the compilation host processor.

The instructions generated by this compiler option differ depending on the compilation host processor.

The following table describes the effects of specifying the \texttt{[Q]xHost} option and it tells whether the resulting executable will run on processors different from the host processor.

Descriptions in the table refer to Intel® Advanced Vector Extensions 2 (Intel® AVX2), Intel® Advanced Vector Extensions (Intel® AVX), Intel® Streaming SIMD Extensions (Intel® SSE), and Supplemental Streaming SIMD Extensions (SSSE).

<table>
<thead>
<tr>
<th>Instruction Set of Host Processor</th>
<th>Effects When the -xHost or /QxHost Compiler Option is Specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® AVX2</td>
<td>When compiling on Intel® processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option \texttt{[Q]xCORE-AVX2}. The generated executable will not run on non-Intel processors and it will not run on Intel® processors that do not support Intel® AVX2 instructions.</td>
</tr>
<tr>
<td></td>
<td>When compiling on non-Intel processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option -march=core-avx2 (Linux*) or /arch:CORE-AVX2 (Windows*). The generated executable will run on Intel® processors and non-Intel processors that support at least Intel® AVX2 instructions.. You may see a run-time error if the run-time processor does not support Intel® AVX2 instructions.</td>
</tr>
<tr>
<td>Intel® AVX</td>
<td>When compiling on Intel® processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option \texttt{[Q]xAVX}. The generated executable will not run on non-Intel processors and it will not run on Intel® processors that do not support Intel® AVX instructions.</td>
</tr>
<tr>
<td></td>
<td>When compiling on non-Intel processors:</td>
</tr>
<tr>
<td>Instruction Set of Host Processor</td>
<td>Effects When the -xHost or /QxHost Compiler Option is Specified</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Intel® SSE4.2</strong></td>
<td>Corresponds to option -mavx (Linux) or /arch:AVX (Windows). The generated executable will run on Intel® processors and non-Intel processors that support at least Intel® AVX instructions. You may see a run-time error if the run-time processor does not support Intel® AVX instructions.</td>
</tr>
<tr>
<td></td>
<td>When compiling on Intel® processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option [Q]xSSE4.2. The generated executable will not run on non-Intel processors and it will not run on Intel® processors that do not support Intel® SSE4.2 instructions.</td>
</tr>
<tr>
<td></td>
<td>When compiling on non-Intel processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option -msse4.2 (Linux) or /arch:SSE4.2 (Windows). The generated executable will run on Intel® processors and non-Intel processors that support at least Intel® SSE4.2 instructions. You may see a run-time error if the run-time processor does not support Intel® SSE4.2 instructions.</td>
</tr>
<tr>
<td><strong>Intel® SSE4.1</strong></td>
<td>When compiling on Intel® processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option [Q]xSSE4.1. The generated executable will not run on non-Intel processors and it will not run on Intel® processors that do not support Intel® SSE4.1 instructions.</td>
</tr>
<tr>
<td></td>
<td>When compiling on non-Intel processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option -msse4.1 (Linux) or /arch:SSE4.1 (Windows). The generated executable will run on Intel® processors and non-Intel processors that support at least Intel® SSE4.1 instructions. You may see a run-time error if the run-time processor does not support Intel® SSE4.1 instructions.</td>
</tr>
<tr>
<td><strong>SSSE3</strong></td>
<td>When compiling on Intel® processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option [Q]xSSSE3. The generated executable will not run on non-Intel processors and it will not run on Intel® processors that do not support SSSE3 instructions.</td>
</tr>
<tr>
<td></td>
<td>When compiling on non-Intel processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option -mssse3 (Linux) or /arch:SSSE3 (Windows). The generated executable will run on Intel® processors and non-Intel processors that support at least SSSE3 instructions. You may see a run-time error if the run-time processor does not support SSSE3 instructions.</td>
</tr>
<tr>
<td><strong>Intel® SSE3</strong></td>
<td>When compiling on Intel® processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option [Q]xSSE3. The generated executable will not run on non-Intel processors and it will not run on Intel® processors that do not support Intel® SSE3 instructions.</td>
</tr>
<tr>
<td></td>
<td>When compiling on non-Intel processors:</td>
</tr>
<tr>
<td></td>
<td>Corresponds to option -msse3 (Linux) or /arch:SSE3 (Windows). The generated executable will run on Intel® processors and non-Intel processors that support at least Intel® SSE3 instructions. You may see a run-time error if the run-time processor does not support Intel® SSE3 instructions.</td>
</tr>
</tbody>
</table>
Instruction Set of Host Processor | Effects When the -xHost or /QxHost Compiler Option is Specified
---|---
Intel® SSE2 | When compiling on Intel® processors or non-Intel processors:
Corresponds to option -msse2 (Linux and macOS*) or /arch:SSE2 (Windows). The generated executable will run on Intel® processors and non-Intel processors that support at least Intel® SSE2 instructions. You may see a run-time error if the run-time processor does not support Intel® SSE2 instructions.

For more information on other settings for option [Q]x, see that option description.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

**IDE Equivalent**

**Visual Studio**
Visual Studio: **Code Generation > Intel Processor-Specific Optimization**

**Eclipse**
Eclipse: **Code Generation > Intel Processor-Specific Optimization**

**Alternate Options**
None

**See Also**
-x, Qx compiler option
m compiler option
arch compiler option

**Interprocedural Optimization (IPO) Options**
This section contains descriptions for compiler options that pertain to interprocedural optimization.

**ipo, Qipo**
*Enables interprocedural optimization between files.*

**Syntax**

**Linux OS:**
- -ipo[n]
- -no-ipo

**Windows OS:**
/ipo[n]
/Qipo
Arguments

\( n \)

Is an optional integer that specifies the number of object files the compiler should create. The integer must be greater than or equal to 0.

Default

-no-ipo or /Qipo-

Multifile interprocedural optimization is not enabled.

Description

This option enables interprocedural optimization between files. This is also called multifile interprocedural optimization (multifile IPO) or Whole Program Optimization (WPO).

When you specify this option, the compiler performs inline function expansion for calls to functions defined in separate files.

You cannot specify the names for the files that are created.

If \( n \) is 0, the compiler decides whether to create one or more object files based on an estimate of the size of the application. It generates one object file for small applications, and two or more object files for large applications.

If \( n \) is greater than 0, the compiler generates \( n \) object files, unless \( n \) exceeds the number of source files (\( m \)), in which case the compiler generates only \( m \) object files.

If you do not specify \( n \), the default is 0.

NOTE

When you specify option [Q]ipo with option [q or Q]opt-report, IPO information is generated in the compiler optimization report at link time. After linking, you will see a report named ipo_out.optrpt in the folder where you linked all of the object files.

IDE Equivalent

Windows

Visual Studio: Optimization > Interprocedural Optimization

Linux

Eclipse: Optimization > Enable Whole Program Optimization

Alternate Options

None

Advanced Optimization Options

This section contains descriptions for compiler options that pertain to advanced optimization.

ffreestanding, Qfreestanding

Ensures that compilation takes place in a freestanding environment.
Syntax

Linux OS:
-ffreestanding

Windows OS:
/Qfreestanding (C++ only)

Arguments

None

Default

OFF Standard libraries are used during compilation.

Description

This option ensures that compilation takes place in a freestanding environment. The compiler assumes that the standard library may not exist and program startup may not necessarily be at main. This environment meets the definition of a freestanding environment as described in the C and C++ standard. An example of an application requiring such an environment is an OS kernel.

NOTE

When you specify this option, the compiler will not assume the presence of compiler-specific libraries. It will only generate calls that appear in the source code.

IDE Equivalent

None

Alternate Options

None

fjump-tables

Determines whether jump tables are generated for switch statements.

Syntax

Linux OS:
-fjump-tables
-fno-jump-tables

Windows OS:

None

Arguments

None

Default

-fjump-tables The compiler may use jump tables for switch statements.
**Description**

This option determines whether jump tables are generated for switch statements.

Option `-fno-jump-tables` prevents the compiler from generating jump tables for switch statements. This action is performed unconditionally and independent of any generated code performance consideration.

Option `-fno-jump-tables` also prevents the compiler from creating switch statements internally as a result of optimizations.

Use `-fno-jump-tables` with `-fpic` when compiling objects that will be loaded in a way where the jump table relocation cannot be resolved.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

`fpic` compiler option

**ipp-link, Qipp-link**

Controls whether the compiler links to static or dynamic threaded Intel® Integrated Performance Primitives (Intel® IPP) run-time libraries.

**Syntax**

**Linux OS:**

`-ipp-link[=lib]`

**Windows OS:**

`/Qipp-link[:lib]`

**Arguments**

`lib` Specifies the Intel® IPP library to use. Possible values are:

- `static` Tells the compiler to link to the set of static run-time libraries.
- `dynamic` Tells the compiler to link to the set of dynamic threaded run-time libraries.

**Default**

`dynamic` The compiler links to the Intel® IPP set of dynamic run-time libraries. However, if Linux® option `-static` is specified, the compiler links to the set of static run-time libraries.

**Description**

This option controls whether the compiler links to static or dynamic threaded Intel® Integrated Performance Primitives (Intel® IPP) run-time libraries.

To use this option, you must also specify the `[Q]ipp` option.
IDE Equivalent

None

Alternate Options

None

See Also
ipp, Qipp compiler option

qactypes, Qactypes

Tells the compiler to include the Algorithmic C (AC) data type folder for header searches and link to the AC data types libraries for Field Programmable Gate Array (FPGA) and CPU compilations.

Syntax

Linux OS:
-qactypes

Windows OS:
/Qactypes

Arguments

None

Default

OFF  The compiler does not search the Algorithmic C (AC) data type folders for headers and doesn’t link to AC data type libraries for FPGA and CPU compilations. As a result, AC data types cannot be used in the source program.

Description

This option tells the compiler to include the Algorithmic C (AC) data type folder when searching for headers, and to link to the AC data types libraries for Field Programmable Gate Array (FPGA) and CPU compilations. AC data types provide support for arbitrary precision integers, fixed precision integers and arbitrary precision floating-point data types. They are built on top of the _ExtInt extended-integer type class.

When you specify option [q or Q]actypes, dynamic linking is the default. You cannot link to the AC data type libraries statically.
NOTE
On Windows* systems, this option adds directives to the compiled code, which the linker then reads without further input from the driver. You do not need to specify a separate link command.

On Linux* systems, the driver must add the library names explicitly to the link command. You must use option `-qactypes` to perform the link to pull in the dependent libraries.

IDE Equivalent
None

Alternate Options
None

`qdaal`, `Qdaal`
*Tells the compiler to link to certain libraries in the Intel® oneAPI Data Analytics Library (oneDAL).*

Syntax

**Linux OS:**

-qdaal[=lib]

**Windows OS:**

/Qdaal[:lib]

Arguments

- `lib`
  Indicates which oneDAL library files should be linked. Possible values are:

  - `parallel`
    Tells the compiler to link using the threaded oneDAL libraries. This is the default if the option is specified with no `lib`.

  - `sequential`
    Tells the compiler to link using the non-threaded oneDAL libraries.

Default

OFF  The compiler does not link to the oneDAL.

Description

This option tells the compiler to link to certain libraries in the Intel® oneAPI Data Analytics Library (oneDAL). On Linux* systems, the associated oneDAL headers are included when you specify this option.
NOTE
On Windows* systems, this option adds directives to the compiled code, which the linker then reads without further input from the driver. You do not need to specify a separate link command.

On Linux* systems, the driver must add the library names explicitly to the link command. You must use option `-qdaal` to perform the link to pull in the dependent libraries.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

IDE Equivalent

Visual Studio
Visual Studio: None

Eclipse
Eclipse: **Performance Library Build Components -> Use Intel® oneAPI Data Analytics Library**

Alternate Options
Linux: `-daal` (this is a deprecated option)

See Also
Using Intel® Performance Libraries

qipp, Qipp
*Tells the compiler to link to the some or all of the Intel® Integrated Performance Primitives (Intel® IPP) libraries. *

Syntax

Linux OS:
`-qipp[=lib]`

Windows OS:
`/Qipp[:lib]`

Arguments

`lib`
Indicates the Intel® IPP libraries that the compiler should link to. Possible values are:

- **common**
  *Tells the compiler to link using the main libraries set. This is the default if the option is specified with no `lib`.*

- **crypto**
  *Tells the compiler to link using the Intel® IPP Cryptography libraries.*
**nonpic (Linux* only)**
Tells the compiler to link using the version of the libraries that do not have position-independent code.

**nonpic_crypto (Linux only)**
Tells the compiler to link using the Intel® IPP Cryptography libraries. It uses the version of the libraries that do not have position-independent code.

**Default**
OFF
The compiler does not link to the Intel® IPP libraries.

**Description**
The option tells the compiler to link to the some or all of the Intel® Integrated Performance Primitives (Intel® IPP) libraries and include the Intel® IPP headers.

The **Qipp-link** option controls whether the compiler links to static, dynamic threaded, or static threaded Intel® IPP run-time libraries.

**NOTE**
On Windows* systems, this option adds directives to the compiled code, which the linker then reads without further input from the driver. You do not need to specify a separate link command.

On Linux* systems, the driver must add the library names explicitly to the link command. You must use option **qipp** to perform the link to pull in the dependent libraries.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDE Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Studio</td>
</tr>
</tbody>
</table>

**Eclipse**

Eclipse: **Performance Library Build Components > Use Intel(R) Integrated Performance Primitives Libraries**

<table>
<thead>
<tr>
<th>Alternate Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux: <code>-qipp</code> (this is a deprecated option)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipp-link, Qipp-link compiler option</td>
</tr>
</tbody>
</table>
**qmkl, Qmkl**
*Tells the compiler to link to certain libraries in the Intel® oneAPI Math Kernel Library (oneMKL). On Windows systems, you must specify this option at compile time.*

### Syntax

**Linux OS:**

- `qmkl[=lib]`

**Windows OS:**

- `/Qmkl[:lib]`

### Arguments

*lib*  
Indicates which oneMKL library files should be linked. Possible values are:

- `parallel`  
  Tells the compiler to link using the threaded libraries in oneMKL. This is the default if the option is specified with no *lib*.

- `sequential`  
  Tells the compiler to link using the sequential libraries in oneMKL.

- `cluster`  
  Tells the compiler to link using the cluster-specific libraries and the sequential libraries in oneMKL.

### Default

OFF  
The compiler does not link to the oneMKL library.

### Description

This option tells the compiler to link to certain libraries in the Intel® oneAPI Math Kernel Library (oneMKL).

On Linux® systems, dynamic linking is the default when you specify `-qmkl`.

On C++ systems, to link with oneMKL statically, you must specify:

- `-qmkl -static-intel`

On Windows® systems, static linking is the default when you specify `/Qmkl`. To link with oneMKL dynamically, you must specify:

- `/Qmkl /MD`

For more information about using oneMKL libraries, see the article in Intel® Developer Zone titled: *Intel® oneAPI Math Kernel Library Link Line Advisor*, which is located in https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl/link-line-advisor.html.
NOTE
On Windows* systems, this option adds directives to the compiled code, which the linker then reads without further input from the driver. You do not need to specify a separate link command.

On Linux* systems, the driver must add the library names explicitly to the link command. You must use option -qmkl to perform the link to pull in the dependent libraries.

NOTE
If you specify option [q or Q]mkl, or -qmkl=parallel or /Qmkl:parallel, and you also specify option [Q]tbb, the compiler links to the standard threaded version of oneMKL.

However, if you specify [q or Q]mkl, or -qmkl=parallel or /Qmkl:parallel, and you also specify option [Q]tbb and option [q or Q]openmp, the compiler links to the OpenMP* threaded version of oneMKL.

IDE Equivalent

Visual Studio
Visual Studio: None

Eclipse
Eclipse: Performance Library Build Components > Use Intel® oneAPI Math Kernel Library

Alternate Options
None

See Also
static-intel compiler option
MD compiler option
qopt-assume-no-loop-carried-dep, Qopt-assume-no-loop-carried-dep
Lets you set a level of performance tuning for loops. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
-qopt-assume-no-loop-carried-dep [=n]

Windows OS:
/Qopt-assume-no-loop-carried-dep [=n]

Arguments

n
Is the action for loop-carried dependencies. Possible values are:

0
The compiler does not assume there are no loop carried dependencies. This is the default if this option is not specified.
Tells the compiler to assume there are no loop-carried dependencies for innermost loops. This is the default if the option is used but \( n \) is not specified.

Tells the compiler to assume there are no loop-carried dependencies for all loop levels.

**Default**

[q opt-assume-no-loop-carried-dep=0]

The compiler does not assume there are no loop carried dependencies.

**Description**

This option lets you set a level of performance tuning for loops.

It is useful for C/C++ applications and benchmarks where pointers and arguments could be aliased. This is because when you specify level 1 or level 2, more loops will be vectorized or benefit from loop transformations.

This option is applied to all loops in the file. It does not apply to code outside loops.

**IDE Equivalent**

None

**Alternate Options**

None

**Examples**

The following loop will not be vectorized because of data dependency. Specifying [q opt-assume-no-loop-carried-dep=1] tells the compiler to assume no data dependence will occur in this loop and it allows this loop to be vectorized:

```c
void sub (float *A, float *B, int* M ) {
    for (int i =0; i< 10000 ; i++) {
        A[i]  += B[M[i]] + 1;
    }
}
```

In the following example, this matrix multiply kernel will not be optimized because of dependency in all loop nests. Specifying [q opt-assume-no-loop-carried-dep=2] will result in loop transformations such as blocking, unroll and jam, and vectorization:

```c
void matmul(double *a, double *b, double *c) {
    int i, j, k;
    int n = 1024;
    for (i = 0; i < 1024; i++) {
        for (j = 0; j < 1024; j++) {
            for (k = 0; k < 1024; k++) {
                c[i * n + j] += a[i * n + k] * b[k * n + j];
            }
        }
    }
}
```
**qopt-for-throughput, Qopt-for-throughput**  
*Determines how the compiler optimizes for throughput depending on whether the program is to run in single-job or multi-job mode.*

**Syntax**

**Linux OS:**  
-qopt-for-throughput[=value]

**Windows OS:**  
/Qopt-for-throughput[:value]

**Arguments**

*value*  
Is one of the values "multi-job" or "single-job". If no value is specified, the default is "multi-job".

**Default**  
OFF  
If this option is not specified, the compiler will not optimize for throughput performance.

**Description**

This option determines whether throughput performance optimization occurs for a program that is run as a single job or one that is run in a multiple job environment.

The memory optimizations for a single job versus multiple jobs can be tuned in different ways by the compiler. For example, the cost model for loop tiling and prefetching are different for a single job versus multiple jobs. When a single job is running, more memory is available and the tunings will be different.

**NOTE**  
When offloading is enabled, this option only applies to host compilation.

**IDE Equivalent**

None

**Alternate Options**

None

**qopt-multiple-gather-scatter-by-shuffles, Qopt-multiple-gather-scatter-by-shuffles**  
*Enables or disables the optimization for multiple adjacent gather/scatter type vector memory references. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**  
-qopt-multiple-gather-scatter-by-shuffles  
-qno-opt-multiple-gather-scatter-by-shuffles
Windows OS:
/Qopt-multiple-gather-scatter-by-shuffles
/Qopt-multiple-gather-scatter-by-shuffles-

Arguments
None

Default
varies

When this option is not specified, the compiler uses default heuristics for optimization.

Description
This option controls the optimization for multiple adjacent gather/scatter type vector memory references. This optimization hint is useful for performance tuning. It tries to generate more optimal software sequences using shuffles.

If you specify this option, the compiler will apply the optimization heuristics. If you specify
-qno-opt-multiple-gather-scatter-by-shuffles
or /Qopt-multiple-gather-scatter-by-shuffles-, the compiler will not apply the optimization.

NOTE
Optimization is affected by optimization compiler options, such as [Q]x, -march (Linux*), and /arch (Windows*).

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.
Notice revision #20201201

IDE Equivalent
None

Alternate Options
None

See Also
march
qtbb, Qtbb

Tells the compiler to link to the Intel® oneAPI Threading Building Blocks (oneTBB) libraries.

Syntax
Linux OS:
-qtbb

Windows OS:
/Qtbb
Arguments
None

Default
OFF The compiler does not link to the oneTBB libraries.

Description
This option tells the compiler to link to the Intel® oneAPI Threading Building Blocks (oneTBB) libraries and include the oneTBB headers.

NOTE
On Windows* systems, this option adds directives to the compiled code, which the linker then reads without further input from the driver. You do not need to specify a separate link command.

On Linux* systems, the driver must add the library names explicitly to the link command. You must use option -qtbb to perform the link to pull in the dependent libraries.

IDE Equivalent
Visual Studio
Visual Studio: None

Eclipse
Eclipse: Performance Library Build Components > Use Intel® oneAPI Threading Building Blocks

Alternate Options
Linux: -tbb (this is a deprecated option)

unroll, Qunroll
Tells the compiler the maximum number of times to unroll loops.

Syntax
Linux OS:
-unroll[=n]

Windows OS:
/Qunroll[:n] (C++ only)

Arguments

n Is the maximum number of times a loop can be unrolled. To disable loop enrolling, specify 0.

Default
-unroll The compiler uses default heuristics when unrolling loops.
or /Qunroll (C++ only)
Description
This option tells the compiler the maximum number of times to unroll loops.
If you do not specify \( n \), the optimizer determines how many times loops can be unrolled.

IDE Equivalent
Windows
Visual Studio: Optimization > Loop Unrolling

Linux
Eclipse: Optimization > Loop Unroll Count

Alternate Options
Linux: -funroll-loops
Windows: None

use-intel-optimized-headers, Quse-intel-optimized-headers
Determines whether the performance headers directory is added to the include path search list. This content is specific to C++; it does not apply to DPC++.

Syntax
Linux OS:
-use-intel-optimized-headers

Windows OS:
/Quse-intel-optimized-headers

Arguments
None

Default
-no-use-intel-optimized-headers or /Quse-intel-optimized-headers-
The performance headers directory is not added to the include path search list.

Description
This option determines whether the performance headers directory is added to the include path search list.
The performance headers directory is added if you specify [Q]use-intel-optimized-headers. Appropriate libraries are also linked in, as needed, for proper functionality.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>
IDE Equivalent

Windows
Visual Studio: **Optimization > Use Intel Optimized Headers**

Linux
Eclipse: **Preprocessor > Use Intel Optimized Headers**

Alternate Options
None

See Also
Using Intel's valarray Implementation

**vec, Qvec**

*Enables or disables vectorization. This content is specific to C++; it does not apply to DPC++.*

Syntax

**Linux OS:**
- `–vec`
- `–no-vec`

**Windows OS:**
- `/Qvec`
- `/Qvec-`

Arguments
None

Default

- `–vec`
  
  Vectorization is enabled if option `O2` or higher is in effect.

  or `/Qvec`

Description

This option enables or disables vectorization.

To disable vectorization, specify `–no-vec (Linux*)` or `/Qvec- (Windows*)`.

**NOTE**

Using this option enables vectorization at default optimization levels for both Intel® microprocessors and non-Intel microprocessors. Vectorization may call library routines that can result in additional performance gain on Intel microprocessors than on non-Intel microprocessors.

IDE Equivalent
None
Alternate Options

None

**vec-threshold, Qvec-threshold**

Sets a threshold for the vectorization of loops. This content is specific to C++; it does not apply to DPC++.

**Syntax**

**Linux OS:**

-vec-threshold\[n\]

**Windows OS:**

/Qvec-threshold[[:]]n

**Arguments**

\(n\)  
Is an integer whose value is the threshold for the vectorization of loops. Possible values are 0 through 100.

If \(n\) is 0, loops get vectorized always, regardless of computation work volume.

If \(n\) is 100, loops get vectorized when performance gains are predicted based on the compiler analysis data. Loops get vectorized only if profitable vector-level parallel execution is almost certain.

The intermediate 1 to 99 values represent the percentage probability for profitable speed-up. For example, \(n=50\) directs the compiler to vectorize only if there is a 50% probability of the code speeding up if executed in vector form.

**Default**

-vec-threshold100  
or /Qvec-threshold100  
Loops get vectorized only if profitable vector-level parallel execution is almost certain. This is also the default if you do not specify \(n\).

**Description**

This option sets a threshold for the vectorization of loops based on the probability of profitable execution of the vectorized loop in parallel.

This option is useful for loops whose computation work volume cannot be determined at compile-time. The threshold is usually relevant when the loop trip count is unknown at compile-time.

The compiler applies a heuristic that tries to balance the overhead of creating multiple threads versus the amount of work available to be shared amongst the threads.

**IDE Equivalent**

**Windows**

Visual Studio: **Optimization > Threshold For Vectorization**

**Linux**

Eclipse: **Optimization > Enable Maximum Vector-level Parallelism**
OS X

Xcode: **Optimization > Enable Maximum Vector-level Parallelism**

**Alternate Options**

None

**Optimization Report Options**

This section contains descriptions for compiler options that pertain to optimization reports.

**qopt-report, Qopt-report**

*Enables the generation of a YAML file that includes optimization transformation information.*

**Syntax**

**Linux OS:**

- `qopt-report [=arg]`

**Windows OS:**

/`Qopt-report [=arg]`

**Arguments**

`arg`

Determines the level of detail in the report. Possible values are:

- **0**
  
  Disables generation of an optimization report. This is the default when the option is not specified.

- **1 or min**
  
  Tells the compiler to create a report with minimum details.

- **2 or med**
  
  Tells the compiler to create a report with medium details. This is the default if you do not specify `arg`.

- **3 or max**
  
  Tells the compiler to create a report with maximum details.

Levels 1, 2, and 3 (min, med, and max) include all the information of the previous level, as well as potentially some additional information.

**Default**

OFF

No optimization report is generated.

**Description**

This option enables the generation of a YAML file that includes optimization transformation information. The YAML-formatted file provides the optimization information for the source file being compiled. For example:

```shell
cx -fiopenmp -qopt-report foo.c
```

This command will generate a file called `foo.opt.yaml` containing the optimization report messages.

Use `opt-viewer.py` (from llvm/tools/opt-viewer) to createhtml files from the YAML file. For example:

```shell
opt-viewer.py foo.opt.yaml
```
You can use any web-browser to open the html file to see the opt-report messages displayed inline with the original. For example:

Firefox html/foo.c.html source code

For SYCL compilations, you can also use this option to detail the variables passed to the OpenCL kernel in the optimization report. For example:

icpx -fsycl -qopt-report foo.cpp          ! command for the C++ compiler
dpcpp -qopt-report foo.cpp               ! command for the DPC++ compiler

The above command will generate a YAML-formatted optimization report that contains optimization remarks for the SYCL pass. These remarks will list the OpenCL kernel arguments generated by the compiler for the user-defined SYCL kernels in foo.cpp. The remarks will also provide additional information like name, type, and size for the OpenCL kernel arguments.

You can then use opt-viewer.py to create html files from the YAML file, and use any web-browser to open the html file to see the opt-report remarks

Note that the YAML file is used to drive the community llvm-opt-report tool.

**IDE Equivalent**
None

**Alternate Options**
None

**Offload Compilation Options, OpenMP* Options, and Parallel Processing Options**

This section contains descriptions for compiler options that pertain to offload compilation, OpenMP*, or parallel processing.

**Device Offload Compilation Considerations**

Data Parallel C++ (DPC++) compilation performs a compile that generates both host and target binaries for a single source file. The DPC++ compilation flow generates file dependencies from the device compilation to the host compilation. These dependent files are considered to be integration files that are included in the host side compilation. A file, called an integration footer, is added to the end of the original source file before being compiled. To accomplish this process, a new temporary source file is generated and is considered the host source file for the compilation. The file is a new source dependency and could break your build environments that track the generated files during a compilation. These build environments need to be configured in the DPC++ space for the additional intermediate file to be part of the compilation flow. The location of the additional file is generated in the common temporary file location, specified by the `TMP` then `TEMP` environment variables.

**device-math-lib**

*Enables or disables certain device libraries. This is a deprecated option that may be removed in a future release. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**

- `--device-math-lib=library`
- `--no-device-math-lib=library`
Windows OS:
/device-math-lib:library
/no-device-math-lib:library

Arguments

library Possible values are:
fp32 Links the fp32 device math library.
fp64 Links the fp64 device math library.

To link more than one library, include a comma between the library names.
For example, if you want to link both the fp32 and fp64 device libraries, specify: fp32, fp64

Default
fp32, fp64 Both the fp32 and fp64 device libraries are linked.

Description
This option enables or disables certain device libraries.

IDE Equivalent
None

Alternate Options
None

See Also
fopenmp-device-lib compiler option
fintelfpga
Lets you perform ahead-of-time (AOT) compilation for the Field Programmable Gate Array (FPGA). This content is specific to DPC++.

Syntax

Linux OS:
-fintelfpga

Windows OS:
-fintelfpga

Arguments
None

Default
OFF The ahead-of-time (AOT) compilation is not performed.
Description
This option lets you perform ahead-of-time (AOT) compilation for the FPGA.
It is functionally equivalent to specifying the following, which is compiled with dependency and debug information enabled:

```
-fsycl-targets=spir64-unknown-unknown
```

IDE Equivalent

Visual Studio
Visual Studio: **DPC++ > General > Enable FPGA workflows**

Eclipse
Eclipse: **Intel(R) oneAPI DPC++ Compiler > General > Enable FPGA workflows**

Alternate Options
None

See Also
- **fsycl-targets** compiler option
- **fsycl-link** compiler option
- **Xs** compiler option

**fiopenmp, Qiopenmp**

*Enables recognition of OpenMP* features, such as parallel, simd, and offloading directives. This is an alternate Linux* option for compiler option **qopenmp**.

Syntax

**Linux OS:**

```
-fiopenmp
```

**Windows OS:**

```
/Qiopenmp
```

Arguments
None

Default
OFF If this option is not specified, OpenMP* features are not transformed in LLVM*.

Description
This option enables recognition of OpenMP* features, such as parallel, simd, and offloading directives. This is an alternate Linux* option for compiler option **qopenmp**.

The **-fiopenmp** and **/Qiopenmp** options enable Intel's implementation of OpenMP* in the compiler back end. The compiler front end produces an intermediate representation that preserves the parallelism exposed by OpenMP* directives. The back end uses the exposed parallelism to do more advanced optimizations, such as SIMD vectorization.
NOTE
To enable offloading to a specified GPU target, you must also specify option `fopenmp-targets` (Linux*) or `/Qopenmp-targets` (Windows).

NOTE
Option `-fopenmp` is not the same as option `-fiopenmp`. Option `-fopenmp` will not do offloading.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

**IDE Equivalent**

**Windows**

Visual Studio: **DPC++ > Language > OpenMP Support**

**C/C++ > Language [Intel C++] > OpenMP Support**

**Intel(R) oneAPI DPC++ Compiler > Language > OpenMP Support**

**Intel C++ Compiler > Language > OpenMP Support**

**Linux**

Eclipse: **Intel(R) oneAPI DPC++ Compiler > Language > OpenMP Support**

**Intel C++ Compiler > Language > OpenMP Support**

**Alternate Options**

**Linux**: `-qopenmp`

**Windows**: `/Qopenmp`

**See Also**

`qopenmp`, `Qopenmp` compiler option

`fopenmp-targets`, `Qopenmp-targets` compiler option

**fno-sycl-libspirv**

*Disables the check for libspirv (the SPIR-V* tools library). This content is specific to DPC++.*

**Syntax**

**Linux OS:**

`-fno-sycl-libspirv`

**Windows OS:**

`-fno-sycl-libspirv`

**Arguments**

None
Default

OFF The check for libspirv is enabled.

Description
This option disables the check for libspirv (the SPIR-V* tools library).

IDE Equivalent
None

Alternate Options
None

foffload-static-lib
Tells the compiler to link with a fat (multi-architecture) static library. This is a deprecated option that may be removed in a future release. This content is specific to DPC++.

Syntax
Linux OS:
-foffload-static-lib=file

Windows OS:
-foffload-static-lib=file

Arguments

file Is the name of the fat static library to use. It can include the path where the library is located.

Default
OFF No linking occurs to a fat static library.

Description
This option tells the compiler to link with a fat (multi-architecture) static library.
The filename specified is treated as a "fat" static library of device code - an archive of fat objects. When linking, the compiler will extract the device code from the objects contained in the library and link it with other device objects coming from the individual fat objects passed on the command line.

NOTE
If you try to pass libraries by using compiler option l, there can be dynamic libraries and partial linking with dynamic libraries, which may lead to a crash.

IDE Equivalent
None
Alternate Options
None

**fopenmp**
Option `-fopenmp` is a deprecated option that will be removed in a future release.

**Syntax**

**Linux OS:**
- `-fopenmp`

**macOS:**
None

**Windows OS:**
None

**Arguments**
None

**Default**
OFF
No OpenMP* multi-threaded code is generated by the compiler.

**Description**
Enables recognition of OpenMP* features and tells the parallelizer to generate multi-threaded code based on OpenMP* directives.

This option is meant for advanced users who prefer to use OpenMP* as it is implemented by the LLVM community. You can get most of that functionality by using this option and option `-fopenmp-simd`.

Option `-fopenmp` is a deprecated option that will be removed in a future release. For most users, we recommend that you instead use option `qopenmp`, `Qopenmp`.

**NOTE**
Option `-fopenmp` is not the same as option `-fiopenmp`. If you want to get full advantage of SIMD vectorization or offloading, you must use option `-qopenmp`.

**IDE Equivalent**
None

Alternate Options
None

**fopenmp-device-lib**
Enables or disables certain device libraries for an OpenMP* target.
Syntax

Linux OS:
-fopenmp-device-lib=library[,library,...]
-fno-openmp-device-lib=library[,library,...]

Windows OS:
-fopenmp-device-lib=library[,library,...]
-fopenmp-device-lib=library[,library,...]

Arguments

library

Possible values are:

libm-fp32
- Enables linking to the fp32 device math library.

libm-fp64
- Enables linking to the fp64 device math library.

libc
- Enables linking to the C library.

all
- Enables linking to libraries libm-fp32, libm-fp-64, and libc.

To link more than one library, include a comma between the library names. For example, if you want to link both the libm-fp32 device library and the C library, specify: libm-fp32,libc.

Do not add spaces between library names.

Note that if you specify "all", it supersedes any additional value you may specify.

Default

OFF
- Disables linking to device libraries for this target.

Description

This option enables or disables certain device libraries for an OpenMP* target.

If you specify fno-openmp-device-lib=library, linking to the specified library is disabled for the OpenMP* target.

IDE Equivalent

Windows

Visual Studio: Linker > General > Enable linking of the device libraries for OpenMP offload
  Linker > General > Disable linking of the device libraries for OpenMP offload

Linux

Eclipse: Linker > Libraries > Enable linking of the device libraries for OpenMP offload
  Linker > Libraries > Disable linking of the device libraries for OpenMP offload
Alternate Options

None

**fopenmp-target-buffers, Qopenmp-target-buffers**

*Enables a way to overcome the problem where some OpenMP* offload SPIR-V* devices produce incorrect code when a target object is larger than 4GB.*

**Syntax**

**Linux OS:**

`-fopenmp-target-buffers=keyword`

**Windows OS:**

`/Qopenmp-target-buffers:keyword`

**Arguments**

<table>
<thead>
<tr>
<th><strong>keyword</strong></th>
<th><strong>Possible values are:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>default</strong></td>
<td>Tells the compiler to use default heuristics. This may produce incorrect code on some OpenMP* offload SPIR-V* devices when a target object is larger than 4GB.</td>
</tr>
<tr>
<td><strong>4GB</strong></td>
<td>Tells the compiler to generate code to prevent the issue described by default. OpenMP* offload programs that access target objects of size larger than 4GB in target code require this option. This setting applies to the following:</td>
</tr>
<tr>
<td></td>
<td>• Target objects declared in OpenMP* target regions or inside OpenMP* declare target functions</td>
</tr>
<tr>
<td></td>
<td>• Target objects that exist in the OpenMP* device data environment</td>
</tr>
<tr>
<td></td>
<td>• Objects that are mapped and/or allocated by means of OpenMP* APIs (such as omp_target_alloc)</td>
</tr>
</tbody>
</table>

**Default**

**default** If you do not specify this option, the compiler may produce incorrect code on some OpenMP* offload SPIR-V* devices when a target object is larger than 4GB.

**Description**

This option enables a way to overcome the problem where some OpenMP* offload SPIR-V* devices produce incorrect code when a target object is larger than 4GB (4294959104 bytes).

However, note that when `-fopenmp-target-buffers=4GB (or /Qopenmp-target-buffers:4GB)` is specified on Intel® GPUs, there may be a decrease in performance.

To use this option, you must also specify option `-fopenmp-targets (Linux*)` or `/Qopenmp-targets (Windows*)`.

**NOTE**

This option may have no effect for some OpenMP* offload SPIR-V* devices, and for OpenMP* offload targets different from SPIR*.
IDE Equivalent

Windows
Visual Studio: DPC++ > Language > Specify buffer size for OpenMP offload kernel access limitations (DPC++)

Windows
Visual Studio: C/C++ > Language [Intel C++] > Specify buffer size for OpenMP offload kernel access limitations (C++)

Linux
Eclipse: Intel(R) oneAPI DPC++ Compiler > Language > Specify buffer size for OpenMP offload kernel access limitations (DPC++)

Linux
Eclipse: Intel C++ Compiler > Language > Specify buffer size for OpenMP offload kernel access limitations (C++)

Alternate Options
None

See Also
fopenmp-targets, Qopenmp-targets compiler option

fopenmp-targets, Qopenmp-targets
Enables offloading to a specified GPU target if OpenMP* features have been enabled.

Syntax
Linux OS:
-fopenmp-targets=keyword

Windows OS:
/Qopenmp-targets:keyword

Arguments

keyword

The only supported value for this argument is spir64.

When you specify spir64, the compiler generates an x86 + SPIR64 (64-bit Standard Portable Intermediate Representation) fat binary for Intel® GPU devices.

Default
OFF

If this option is not specified, no x86 + SPIR64 fat binary is created.

Description
This option enables offloading to a specified GPU target if OpenMP* features have been enabled.
To use this option, you must enable recognition of OpenMP* features by specifying one of the following options:
- [q or Q]openmp
- -fiopenmp (Linux*) or /Qopenmp (Windows*)
- -fopenmp (deprecated; it is equivalent to -qopenmp on Linux*)

The following shows an example:

```
icx (or icpx) -fiopenmp -fopenmp-targets=spir64 matmul_offload.cpp -o matmul
```

When you specify -fopenmp-targets (Linux*) or /Qopenmp-targets (Windows*), C++ exception handling is disabled for target compilations.

For host compilations on Linux* systems, if you want to disable C++ exception handling, you must specify option -fno-exceptions.

**IDE Equivalent**

**Windows**

Visual Studio: **DPC++ > Language > Enable OpenMP Offloading**
**C/C++ > Language [Intel C++] > Enable OpenMP Offloading**
**Intel(R) oneAPI DPC++ Compiler > Language > Enable OpenMP Offloading**
**Intel C++ Compiler > Language > Enable OpenMP Offloading**

**Linux**

Eclipse: **Intel(R) oneAPI DPC++ Compiler > Language > Enable OpenMP Offloading**
**Intel C++ Compiler > Language > Enable OpenMP Offloading**

**Alternate Options**

None

**See Also**

fiopenmp, Qiopenmp  compiler option
qopenmp, Qopenmp  compiler option

**fsycl**

*Enables a program to be compiled as a SYCL* program rather than as plain C++11 program.*

**Syntax**

**Linux OS:**
- -fsycl

**Windows OS:**
- -fsycl

**Arguments**

None

**Default**

DPC++: A C++ program is compiled as a SYCL* program.
ON
**C++**: OFF  A C++ program is compiled as a C++11 program.

**Description**

This option enables a program to be compiled as a SYCL* program rather than as plain C++11 program.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

fsycl-targets  compiler option

**fsycl-add-targets**

*Lets you add arbitrary device binary images to the fat SYCL* binary when linking. This is a deprecated option that may be removed in a future release. This content is specific to DPC++.*

**Syntax**

**Linux OS:**

```
-fsycl-add-targets=T1:file1,...,Tn:filen
```

**Windows OS:**

```
-fsycl-add-targets=T1:file1,...,Tn:filen
```

**Arguments**

- **T**: Is a target triple for the device binary image.
- **file**: Is the location of the device binary image.

You can specify one or more pair of **T**:**file**.

**Default**

OFF  Arbitrary device images are not added to any fat SYCL* binary being linked.

**Description**

This option lets you add arbitrary device binary images to the fat SYCL* binary when linking.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

fsycl-link-targets  compiler option
**fsycl-dead-args-optimization**

*Enables elimination of DPC++ dead kernel arguments. This content is specific to DPC++.*

**Syntax**

**Linux OS:**
- `fsycl-dead-args-optimization`
- `fno-sycl-dead-args-optimization`

**Windows OS:**
- `fsycl-dead-args-optimization`
- `fno-sycl-dead-args-optimization`

**Arguments**

None

**Default**

OFF  
DPC++ dead kernel arguments are not eliminated. This default may change in the future.

**Description**

This option enables elimination of DPC++ dead kernel arguments. This optimization can improve performance.

If you specify `-fno-sycl-dead-args-optimization`, this optimization is disabled.

**IDE Equivalent**

None

**Alternate Options**

None

**fsycl-device-code-split**

*Specifies a SYCL* device code module assembly. *This content is specific to DPC++.*

**Syntax**

**Linux OS:**
- `fsycl-device-code-split[=value]`

**Windows OS:**
- `fsycl-device-code-split[=value]`

**Arguments**

`value`  
Can be only one of the following:
per_kernel

Creates a separate device code module for each SYCL* kernel. Each device code module will contain a kernel and all its dependencies, such as called functions and used variables.

per_source

Creates a separate device code module for each source (translation unit).

Each device code module will contain a collection of kernels grouped on per-source basis and all their dependencies, such as all used variables and called functions, including the SYCL_EXTERNAL macro-marked functions from other translation units.

off

Creates a single module for all kernels.

auto

The compiler will use a heuristic to select the best way of splitting device code. This is the same as specifying fsycl-device-code-split with no value.

Default

This is the default whether you do not specify the compiler option or you do specify the compiler option, but omit a value. The compiler will use a heuristic to select the best way of splitting device code.

Description

This option specifies a SYCL* device code module assembly.

IDE Equivalent

None

Alternate Options

None

fsycl-device-lib

Enables or disables certain device libraries for a SYCL* target.

Syntax

Linux OS:

- fsycl-device-lib=library[,library,...]
- fno-sycl-device-lib=library[,library,...]

Windows OS:

- fsycl-device-lib=library[,library,...]
- fsycl-device-lib=library[,library,...]
Arguments

library

Possible values are:

libm-fp32
Enables linking to the fp32 device math library.

libm-fp64
Enables linking to the fp64 device math library.

libc
Enables linking to the C library.

all
Enables linking to libraries libm-fp32, libm-fp-64, and libc.

To link more than one library, include a comma between the library names. For example, if you want to link both the libm-fp32 device library and the C library, specify: libm-fp32,libc.

Do not add spaces between library names.

Note that if you specify "all", it supersedes any additional value you may specify.

Default

OFF Disables linking to device libraries for this target.

Description

This option enables or disables certain device libraries for a SYCL* target.

If you specify fno-sycl-device-lib=library, linking to the specified library is disabled for the SYCL* target.

IDE Equivalent

Windows

Visual Studio: Linker > General > Enable linking of the device libraries
Linker > General > Disable linking of the device libraries

Linux

Eclipse: Linker > Libraries > Enable linking of the device libraries
Linker > Libraries > Disable linking of the device libraries

Alternate Options

None

fsycl-device-only
Tells the compiler to generate a device-only binary.
This content is specific to DPC++.

Syntax

Linux OS:

- fsycl-device-only
Windows OS:
- `fsycl-device-only`

**Arguments**
None

**Default**
OFF  
No device-only binary is generated.

**Description**
This option tells the compiler to generate a device-only binary.

**IDE Equivalent**
None

**Alternate Options**
None

**fsycl-early-optimizations**
*Enables LLVM-related optimizations before SPIR-V* 
geneneration. *This content is specific to DPC++.*

**Syntax**

Linux OS:
- `fsycl-early-optimizations`
- `fno-sycl-early-optimizations`

Windows OS:
- `fsycl-early-optimizations`
- `fno-sycl-early-optimizations`

**Arguments**
None

**Default**
ON  
LLVM-related optimizations are enabled before SPIR-V* generation.

**Description**
This option enables LLVM-related optimizations before SPIR-V* generation. These optimizations can improve performance.

If you specify `fno-sycl-early-optimizations`, these optimizations are disabled.

**IDE Equivalent**

Visual Studio: **DPC++ > Optimization > Enable/Disable DPC++ early optimization before generation of SPIR-V code**
Eclipse

Eclipse: Intel(R) oneAPI DPC++ Compiler > Optimization > Enable/Disable DPC++ early optimization before generation of SPIR-V code

Alternate Options

None

fsycl-enable-function-pointers

Enables function pointers and support for virtual functions for DPC++ kernels and device functions. This is an experimental feature. This content is specific to DPC++.

Syntax

Linux OS:
-fsycl-enable-function-pointers

Windows OS:
-fsycl-enable-function-pointers

Arguments

None

Default

OFF  Function pointers are not enabled and virtual functions for DPC++ kernels and device functions are not supported.

Description

This option enables function pointers and support for virtual functions for DPC++ kernels and device functions. This is an experimental feature.

This enhanced support is limited to CPU-device only and cannot currently be used for GPU devices.

IDE Equivalent

None

Alternate Options

None

fsycl-explicit-simd

Enables or disables the experimental "Explicit SIMD" SYCL* extension. This is a deprecated option that may be removed in a future release. This content is specific to DPC++.

Syntax

Linux OS:
-fsycl-explicit-simd
-fno-sycl-explicit-simd
**Windows OS:**

- `fsycl-explicit-simd`
- `fno-sycl-explicit-simd`

**Arguments**

None

**Default**

- `fno-sycl-explicit-simd` The explicit SIMD SYCL* extension is disabled.

**Description**

This option enables or disables the experimental "Explicit SIMD" SYCL* extension.

If you specify option `-fsycl-explicit-simd`, it enables the experimental "Explicit SIMD" SYCL* extension for lower-level Intel GPU programming. It allows you to write explicitly vectorized device code. Note that APIs for this feature may change in the future.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

Explicit SIMD SYCL* Extension

**fsycl-help**

_Causes help information to be emitted from the device compiler backend. This content is specific to DPC++._

**Syntax**

**Linux OS:**

- `fsycl-help[=arg]`

**Windows OS:**

- `fsycl-help[=arg]`

**Arguments**

`arg` Can be one of "x86_64", "fpga", "gen", or "all". Option `-fsycl-help=all` outputs help for "x86_64", "fpga", and "gen".

Specifying "all" is the same as specifying `fsycl-help` with no `arg`.

**Default**

OFF No help information is emitted from the device compiler backend.

**Description**

This option causes help information to be emitted from the device compiler backend.
IDE Equivalent
None

Alternate Options
None

**fsycl-host-compiler**
*Tells the compiler to use the specified compiler for the host compilation of the overall offloading compilation that is performed. This content is specific to DPC++.*

**Syntax**

**Linux OS:**

```
-fsycl-host-compiler=arg
```

**Windows OS:**

```
-fsycl-host-compiler=arg
```

**Arguments**

*arg*  
Is the compiler that will be the host for compilation.  
It can be the name of a compiler or the specific path to the compiler.

**Default**

OFF  
The host compilation will be performed by the DPC++ compiler.

**Description**

This option tells the compiler to use the specified compiler for the host compilation of the overall offloading compilation that is performed.  
To use this option, you must also specify option *fsycl*.

IDE Equivalent
None

Alternate Options
None

**Example**

Consider the following:

```
-fsycl-host-compiler=g++  // the compiler looks for g++ in the current path
-fsycl-host-compiler=/usr/bin/g++  // the compiler looks for g++ in the explicit path
```

**See Also**

*fsycl*  compiler option  
*fsycl-host-compiler-options*  compiler option
**fsycl-host-compiler-options**

*Passes options to the compiler specified by option* 
*fsycl-host-compiler.* *This content is specific to DPC++.*

**Syntax**

**Linux OS:**

- `-fsycl-host-compiler-options="opts"`

**Windows OS:**

- `-fsycl-host-compiler-options="opts"`

**Arguments**

*opts* 

Is a string of compatible compiler options to be passed. The string must appear within quotes.

If there is more than one compiler option, a space must appear between each option name.

**Default**

*OFF* 

No options are passed to the compiler specified by `-fsycl-host-compiler`.

**Description**

This option tells the compiler to pass options to the compiler specified by option *fsycl-host-compiler.* The options must be compatible with the compiler specified by *fsycl-host-compiler.*

---

**NOTE**

Specifying any kind of phase limiting options (such as `-C`, `-E`, or `-S`) may interfere with the expected output set during the host compilation. This can cause undefined behavior.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

*fsycl-host-compiler* compiler option

*fsycl-id-queries-fit-in-int*  

* Tells the compiler to assume that SYCL ID queries fit within MAX_INT.* This content is specific to DPC++.

**Syntax**

**Linux OS:**

- `-fsycl-id-queries-fit-in-int`

- `-fno-sycl-id-queries-fit-in-int`
Windows OS:
-\texttt{fsycl-id-queries-fit-in-int}
-\texttt{fno-sycl-id-queries-fit-in-int}

**Arguments**
None

**Default**
ON

The compiler assumes that SYCL ID queries fit within \texttt{MAX\_INT}.

**Description**
This option tells the compiler to assume that SYCL ID queries fit within \texttt{MAX\_INT}. It assumes that the following values fit within \texttt{MAX\_INT}:

- id class \texttt{get()} member function and \texttt{operator[]} member
- item class \texttt{get\_id()} member function and \texttt{operator[]} member
- nd\_item class \texttt{get\_global\_id()}/\texttt{get\_global\_linear\_id()} member functions

For more information about these values, see the Khronos* Group SYCL* 1.2.1 Specification.

If you need to use a larger number of work items, use the OFF setting for this option, which is \texttt{-fno-sycl-id-queries-fit-in-int}.

---

**Caution**
You should carefully evaluate whether you should use the OFF setting when you have a larger number of work items. Truncating to data type int is often incorrect in such circumstances. If the OFF setting is used when the values fit within \texttt{MAX\_INT}, it can lead to unexpected performance issues.

---

**IDE Equivalent**
None

**Alternate Options**
None

**fsycl-link**
*Tells the compiler to perform a partial link of device binaries to be used with Field Programmable Gate Array (FPGA). This content is specific to DPC++.*

**Syntax**

**Linux OS:**
-\texttt{fsycl-link[=value]}

**Windows OS:**
-\texttt{fsycl-link[=value]}

**Arguments**

\texttt{value} Can be one of the following:
Tells the compiler to generate an HTML report when the partial link is created. This capability lets you check the program if need be.

You can resume from this point and generate an FPGA image by specifying option `-fintelfpga` with the generated binary.

**image**

Tells the compiler to generate an FPGA bitstream. It will then be ready to be linked and used on an FPGA board.

*image* takes much longer to generate than does *early*.

**Default**

OFF  No partial link of device binaries is performed.

**Description**

This option tells the compiler to perform a partial link of device binaries to be used with FPGA. This partial link is then wrapped by the offload wrapper, allowing the device binaries to be linked by the host compiler or linker.

If you do not specify a value, the following occurs:

- When used with just `-fsycl (-fsycl -fsycl-link)`, the driver will generate a host linkable device object.
- When also used with `-fintelfpga (-fsycl -fintelfpga -fsycl-link)`, the behavior is the same as specifying `-fsycl-link=early`.

**IDE Equivalent**

**Visual Studio**

Visual Studio: **Linker > General > Generate partially linked device object to be used with the host link**

**Eclipse**

Eclipse: **Linker > General > Generate partially linked device object to be used with the host link**

**Alternate Options**

None

**See Also**

fintelfpga  compiler option

**fsycl-link-targets**

*Tells the compiler to link only device code. This is a deprecated option that may be removed in a future release. This content is specific to DPC++.*
Syntax

Linux OS:
-\texttt{fsycl-link-targets=T1,...,Tn}

Windows OS:
-\texttt{fsycl-link-targets=T1,...,Tn}

Arguments

$ T $ is a target triple for the device code. You can specify more than one $ T $.

Default

OFF No link is performed.

Description

This option tells the compiler to link only device code. It is used in a link step. It tells the compiler to link device code for the given target triples, and output multiple linked device code images. It does not produce fat binary.

To use this option, you must also specify option \texttt{fsycl}.

\textbf{NOTE}

You should be familiar with ahead-of-time (AOT) compilation when using this option.

IDE Equivalent

None

Alternate Options

None

Example

The following command-line sequence demonstrates a way to use this option:

\begin{verbatim}
dpcpp -fsycl-targets=spir64 -c a.cpp -o a.o
dpcpp -fsycl-targets=spir64 -c b.cpp -o b.o
dpcpp -fsycl-link-targets=spir64 a.o b.o -o linked.spv
aoc linked.spv -o linked.aocx
dpcpp -fsycl-add-targets=fpga:linked.aocx a.o b.o -o final.out -lOpenCL -lsycl
\end{verbatim}

See Also

\texttt{fsycl} compiler option
\texttt{fsycl-add-targets} compiler option
\texttt{fsycl-targets} compiler option

\texttt{fsycl-max-parallel-link-jobs}

\textit{Tells the compiler that it can simultaneously spawn up to the specified number of processes to perform actions required to link DPC++ applications. This is an experimental feature. This content is specific to DPC++}.
Syntax

Linux OS:
- `fsycl-max-parallel-link-jobs=n`

Windows OS:
- `fsycl-max-parallel-link-jobs=n`

Arguments

`n` Is the number of processes to spawn to.

Default

`-fsycl-max-parallel-link-jobs=1` One process is simultaneously spawned to perform actions necessary to link DPC++ applications.

Description

This option tells the compiler that it can simultaneously spawn up to the specified number of processes to perform actions required to link DPC++ applications. This is an experimental feature.

For this option to be effective, you should also specify option `-fsycl`.

**NOTE**

If you specify a large number of processes, it can cause performance issues and compilation crashes due to excessive RAM consumption.

**NOTE**

This option has no effect if compiler options such as `c` or `E` are specified.

IDE Equivalent

None

Alternate Options

None

Example

The following shows examples of using this option on Linux*:

```
dpcpp -fsycl-max-parallel-link-jobs=4 a.cpp b.cpp c.cpp d.cpp -o a.out
```

```
dpcpp -fsycl-max-parallel-link-jobs=8 a.o b.o c.o d.so e.a -o b.out
```

See Also

- `fsycl` compiler option
- `fsycl-targets` Tells the compiler to generate code for specified device targets. This content is specific to DPC++.
Syntax
Linux OS:
-fsycl-targets=T1,...,Tn
Windows OS:
-fsycl-targets=T1,...,Tn

Arguments
$T$ is a target triple device name. If you specify more than one $T$, they must be separated by commas. The following triplets are supported:

- **spir64**
  - Tells the compiler to use default heuristics for SPIR64-based devices. This is the default. You can also specify this value as spir64-unknown-unknown.

- **spir64_x86_64**
  - Tells the compiler to generate code for Intel® CPUs. You can also specify this value as spir64_x86_64-unknown-unknown.

- **x86_64**
  - Tells the compiler to generate code ahead of time for x86_64 CPUs; it provides better debuggability. This triplet can also be specified as x86_64-unknown-unknown.

- **spir64_fpga**
  - Tells the compiler to generate code for Intel® FPGA. You can also specify this value as spir64_fpga-unknown-unknown.

- **spir64_gen**
  - Tells the compiler to generate code for Intel® Processor Graphics. You can also specify this value as spir64_gen-unknown-unknown.

Default
spir64

Description
This option tells the compiler to generate code for specified device targets.

**NOTE**
The long syntax values that include `-sycldevice`, such as spir64-unknown-unknown-sycldevice, are still supported, but they are deprecated.

IDE Equivalent

**Visual Studio**
Visual Studio: **DPC++ > General > Specify SYCL offloading targets for AOT compilation**
Eclipse
Eclipse: Intel(R) oneAPI DPC++ Compiler > General > Specify SYCL offloading targets for AOT compilation

Alternate Options
None

fsycl-unnamed-lambda
Enables unnamed SYCL* lambda kernels. This content is specific to DPC++.

Syntax
Linux OS:
- fsycl-unnamed-lambda
- fno-sycl-unnamed-lambda

Windows OS:
- fsycl-unnamed-lambda
- fno-sycl-unnamed-lambda

Arguments
None

Default
ON Unnamed SYCL lambda kernels are enabled.

Description
This option enables unnamed SYCL kernels that are defined as lambdas.
If you specify -fno-sycl-unnamed-lambda, unnamed SYCL lambda kernels are disabled.

IDE Equivalent

Visual Studio
Visual Studio: DPC++ > General > Allow unnamed SYCL lambda kernels

Eclipse
Eclipse: Intel(R) oneAPI DPC++ Compiler > Language > Allow unnamed SYCL lambda kernels

Alternate Options
None

fsycl-use-bitcode
Tells the compiler to produce device code in LLVM IR bitcode format into fat objects. This content is specific to DPC++.
Syntax

Linux OS:
-\texttt{fsycl-use-bitcode}

Windows OS:
-\texttt{fsycl-use-bitcode}

Arguments
None

Default
ON

LLVM IR bitcode format is emitted.

Description
This option tells the compiler to produce device code in LLVM IR bitcode format into fat objects.

IDE Equivalent
None

Alternate Options
None

\texttt{nolibsycl}
\textit{Disables linking of the SYCL* runtime library. This content is specific to DPC++.

Syntax

Linux OS:
-\texttt{nolibsycl}

Windows OS:
-\texttt{nolibsycl}

Arguments
None

Default
OFF

The SYCL* runtime library is linked.

Description
This option disables linking of the SYCL* runtime library.

IDE Equivalent
None

Alternate Options
None
**qopenmp, Qopenmp**

*Enables recognition of OpenMP* features and tells the parallelizer to generate multi-threaded code based on OpenMP* directives.*

**Syntax**

**Linux OS:**
- `-qopenmp`
- `-qno-openmp`

**Windows OS:**
- `/Qopenmp`
- `/Qopenmp-`

**Arguments**

None

**Default**

- `-qno-openmp` or `/Qopenmp-`  
No OpenMP* multi-threaded code is generated by the compiler.

**Description**

This option enables recognition of OpenMP* features and tells the parallelizer to generate multi-threaded code based on OpenMP* directives. The code can be executed in parallel on both uniprocessor and multiprocessor systems.

This option works with any optimization level. Specifying no optimization (`-O0` on Linux* or `/Od` on Windows*) helps to debug OpenMP applications.

This option can also be specified as `-fopenmp` on Linux* systems.

---

**NOTE**

To enable offloading to a specified GPU target, you must also specify option `fopenmp-targets` (Linux*) or `/Qopenmp-targets` (Windows).

---

**NOTE**

Option `-fopenmp` is not the same as option `-qopenmp`. Option `-fopenmp` will *not* do offloading.

---

**NOTE**

Options that use OpenMP* API are available for both Intel® microprocessors and non-Intel microprocessors, but these options may perform additional optimizations on Intel® microprocessors than they perform on non-Intel microprocessors. The list of major, user-visible OpenMP constructs and features that may perform differently on Intel® microprocessors versus non-Intel microprocessors include: locks (internal and user visible), the SINGLE construct, barriers (explicit and implicit), parallel loop scheduling, reductions, memory allocation, thread affinity, and binding.
IDE Equivalent

Visual Studio
Visual Studio: **Language > OpenMP* Support**

Eclipse
Eclipse: **Language > Process OpenMP Directives**

Alternate Options

Linux: `-fiopenmp`

Windows: `/Qiopenmp`

See Also

`fopenmp-targets`, `Qopenmp-targets` compiler option
`fiopenmp`, `Qiopenmp` compiler option

`qopenmp-lib`, `Qopenmp-lib`

*Let you specify an OpenMP* run-time library to use for linking.*

Syntax

**Linux OS:**
- `-qopenmp-lib=type`

**Windows OS:**
- `/Qopenmp-lib:type`

Arguments

**type**

Specifies the type of library to use; it implies compatibility levels. Currently, the only possible value is:

`compat`

Tells the compiler to use the compatibility OpenMP* run-time library (libiomp). This setting provides compatibility with object files created using Microsoft* and GNU* compilers.

Default

- `-qopenmp-lib=compat`
- `or /Qopenmp-lib:compat`

The compiler uses the compatibility OpenMP* run-time library (libiomp).

Description

This option lets you specify an OpenMP* run-time library to use for linking.
The compatibility OpenMP run-time libraries are compatible with object files created using the Microsoft* OpenMP run-time library (vcomp) or the GNU OpenMP run-time library (libgomp).

To use the compatibility OpenMP run-time library, compile and link your application using the compat setting for option [q or Q]openmp-lib. To use this option, you must also specify one of the following compiler options:

- Linux* systems: -qopenmp or -qopenmp-stubs
- Windows* systems: /Qopenmp or /Qopenmp-stubs

On Windows* systems, the compatibility OpenMP* run-time library lets you combine OpenMP* object files compiled with the Microsoft* C/C++ compiler with OpenMP* object files compiled with the Intel® C, Intel® C++, or Intel® Fortran compilers. The linking phase results in a single, coherent copy of the run-time library.

On Linux* systems, the compatibility Intel OpenMP* run-time library lets you combine OpenMP* object files compiled with the GNU* gcc or gfortran compilers with similar OpenMP* object files compiled with the Intel® C, Intel® C++, or Intel® Fortran Compiler. The linking phase results in a single, coherent copy of the run-time library.

**NOTE** The compatibility OpenMP run-time library is not compatible with object files created using versions of the Intel compilers earlier than 10.0.

**NOTE** On Windows* systems, this option is processed by the compiler, which adds directives to the compiled object file that are processed by the linker. On Linux* systems, this option is processed by the icx/icpx command that initiates linking, adding library names explicitly to the link command.

**IDE Equivalent**
None

**Alternate Options**
None

**See Also**
qopenmp, Qopenmp  compiler option
qopenmp-stubs, Qopenmp-stubs  compiler option

**qopenmp-link**
*Controls whether the compiler links to static or dynamic OpenMP* run-time libraries.*

**Syntax**

**Linux OS:**
-qopenmp-link=library

**Windows OS:**
None

**Arguments**

*library*  Specifies the OpenMP library to use. Possible values are:
static

Tells the compiler to link to static OpenMP run-time libraries. Note that static OpenMP libraries are deprecated.

dynamic

Tells the compiler to link to dynamic OpenMP run-time libraries.

Default

-qopenmp-link=dynamic

The compiler links to dynamic OpenMP* run-time libraries. However, if Linux* option -static is specified, the compiler links to static OpenMP run-time libraries.

Description

This option controls whether the compiler links to static or dynamic OpenMP* run-time libraries.

To link to the static OpenMP run-time library (RTL) and create a purely static executable, you must specify -qopenmp-link=static. However, we strongly recommend you use the default setting, -qopenmp-link=dynamic.

NOTE

Compiler options -static-intel and -shared-intel (Linux*) have no effect on which OpenMP run-time library is linked.

NOTE

On Linux* systems, -qopenmp-link=dynamic cannot be used in conjunction with option -static. If you try to specify both options together, an error will be displayed.

NOTE

On Linux systems, the OpenMP runtime library depends on using libpthread and libc (libgcc when compiled with gcc). Libpthread and libc (libgcc) must both be static or both be dynamic. If both libpthread and libc (libgcc) are static, then the static version of the OpenMP runtime should be used. If both libpthread and libc (libgcc) are dynamic, then either the static or dynamic version of the OpenMP runtime may be used.

IDE Equivalent

None

Alternate Options

None

qopenmp-simd, Qopenmp-simd

Enables or disables OpenMP* SIMD compilation.

Syntax

Linux OS:

-qopenmp-simd
-qno-openmp-simd

**Windows OS:**
/Qopenmp-simd
/Qopenmp-simd-

**Arguments**
None

**Default**
-qno-openmp-simd or /Qopenmp-simd- OpenMP* SIMD compilation is disabled.

**Description**
This option enables or disables OpenMP* SIMD compilation. You can use this option if you want to enable or disable the SIMD support with no impact on other OpenMP features. In this case, no OpenMP runtime library is needed to link and the compiler does not need to generate OpenMP runtime initialization code. If you specify this option with the [q or Q]openmp option, it can impact other OpenMP features. Option -qopenmp-simd is equivalent to option -fopenmp-simd; option /Qopenmp-simd is equivalent to option /Qopenmp-simd.

---

**NOTE**
Advanced users who prefer to use OpenMP* as it is implemented by the LLVM community, can get most of that functionality by using options -fopenmp and -fopenmp-simd.

**IDE Equivalent**
None

**Alternate Options**
Linux: -fopenmp-simd
Windows /Qopenmp-simd

**Example**
Consider the following:

<table>
<thead>
<tr>
<th>Command</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>-qno-openmp -qopenmp-simd</td>
<td>Linux</td>
</tr>
<tr>
<td>/Qopenmp- /Qopenmp-simd</td>
<td>Windows</td>
</tr>
</tbody>
</table>

The above is equivalent to specifying only [q or Q]openmp-simd. In this case, only SIMD support is provided, the OpenMP* library is not linked, and only the !$OMP directives related to SIMD are processed.

Consider the following:

<table>
<thead>
<tr>
<th>Command</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>-qopenmp -qopenmp-simd</td>
<td>Linux</td>
</tr>
<tr>
<td>/Qopenmp /Qopenmp-simd</td>
<td>Windows</td>
</tr>
</tbody>
</table>

In this case, SIMD support is provided, the OpenMP library is linked, and OpenMP runtime initialization code is generated. Note that when you specify [q or Q]openmp, it implies [q or Q]openmp-simd.

**See Also**
qopenmp, Qopenmp compiler option
**qopenmp-stubs, Qopenmp-stubs**  
*Enables compilation of OpenMP* programs in *sequential mode.*

**Syntax**

**Linux OS:**  
-qopenmp-stubs

**Windows OS:**  
/Qopenmp-stubs

**Arguments**

None

**Default**

OFF  
The library of OpenMP* function stubs is not linked.

**Description**

This option enables compilation of OpenMP* programs in sequential mode. The OpenMP directives are ignored and a stub OpenMP library is linked.

**IDE Equivalent**

**Windows**

Visual Studio: **Language > OpenMP Support**

**Linux**

Eclipse: **Language > Process OpenMP Directives**

**Alternate Options**

None

**See Also**

qopenmp, Qopenmp  compiler option

**qopenmp-threadprivate, Qopenmp-threadprivate**  
* Lets you specify an OpenMP* threadprivate implementation. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**  
-qopenmp-threadprivate=type

**Windows OS:**  
/Qopenmp-threadprivate:type
### Arguments

**type**

Specifies the type of threadprivate implementation. Possible values are:

- **legacy**
  
  Tells the compiler to use the legacy OpenMP* threadprivate implementation used in the previous releases of the Intel® compiler. This setting does not provide compatibility with the implementation used by other compilers.

- **compat**
  
  Tells the compiler to use the compatibility OpenMP* threadprivate implementation based on applying the `__declspec(thread)` attribute to each threadprivate variable. The limitations of the attribute on a given platform also apply to the threadprivate implementation. This setting provides compatibility with the implementation provided by the Microsoft* and GNU* compilers.

### Default

- `-qopenmp-threadprivate=compat`
  
  The compiler uses the compatibility OpenMP* threadprivate implementation.

### Description

This option lets you specify an OpenMP* threadprivate implementation.

The threadprivate implementation of the legacy OpenMP run-time library may not be compatible with object files created using OpenMP run-time libraries supported in other compilers.

To use this option, you must also specify one of the following compiler options:

- **Linux* systems:** `-qopenmp` or `-qopenmp-stubs`
- **Windows* systems:** `/Qopenmp` or `/Qopenmp-stubs`

The value specified for this option is independent of the value used for the `[q or Q]openmp-lib` option.

### IDE Equivalent

None

### Alternate Options

None

### See Also

- qopenmp, Qopenmp  compiler option
- qopenmp-stubs, Qopenmp-stubs  compiler option

### reuse-exe

*Tells the compiler to speed up Field Programmable Gate Array (FPGA) target compile time by reusing a previously compiled FPGA hardware image. This option is useful only when compiling for hardware.*

*This content is specific to DPC++.*
Syntax
Linux OS:
-reuse-exe=exe-filename

Windows OS:
-reuse-exe=exe-filename

Arguments
exe-filename Is a previously compiled SYCL* binary.

Default
OFF There is no potential compile-time speed up by reusing the executable for the FPGA target.

Description
This option tells the compiler to speed up FPGA target compile time by reusing a previously compiled FPGA hardware image. This option is useful only when compiling for hardware. It lets you minimize or avoid long Intel® Quartus® Prime Software compile times for FPGA targets when the device code is unchanged.

IDE Equivalent
None

Alternate Options
None

Wno-sycl-strict
Disables warnings that enforce strict SYCL* language compatibility.

Syntax
Linux OS:
-Wno-sycl-strict

Windows OS:
-Wno-sycl-strict

Arguments
None

Default
OFF Warnings that enforce strict SYCL* language compatibility are enabled.

Description
This option disables warnings that enforce strict SYCL* language compatibility.

IDE Equivalent
None
Alternate Options
None

Xs
*Passes options to the backend tool. This content is specific to DPC++.*

Syntax

**Linux OS:**

- `Xs -option` or `-Xs option`

**Windows OS:**

- `Xs -option` or `-Xs option`

Arguments

`option`

Is the option that you want to pass to the backend tool in device compilation.

To see the values you can use for `option`, specify compiler option `-fsycl-help` to display the help information for the offline tools.

Default

**OFF**

No options are passed to the backend tool.

Description

This option passes options to the backend tool. It is a shortcut for option `Xsycl-target-backend`.

For example, the following option (using syntax form `-Xs option`):

- `Xs version`

and the following option (using syntax form `-Xs -option`):

- `Xs -version`

are both equivalent to specifying:

- `Xsycl-target-backend -version`

IDE Equivalent

**Visual Studio**

Visual Studio: **Linker > General > Enable FPGA hardware build**

**Eclipse**

Eclipse: **Linker > General > Enable FPGA hardware build**

Alternate Options

None

See Also

`Xsycl-target`

compiler option
**Xopenmp-target**

*Enables options to be passed to the specified tool in the device compilation tool chain for the target. This compiler option supports OpenMP* offloading.*

**Syntax**

**Linux OS:**

-Xopenmp-target-tool=T "options"

**Windows OS:**

-Xopenmp-target-tool=T "options"

**Arguments**

```
tool
    Can be one of the following:
    frontend
        Indicates the frontend + middle end of the Standard Portable Intermediate Representation (SPIR-V*)-based device compiler for target triple T.
        The middle end is the part of a SPIR-V*-based device compiler that generates SPIR-V*. This SPIR-V* is then passed by the dpcpp driver to the backend of target T.
    backend
        Indicates Ahead of Time (AOT) compilation for target triple T and Just in Time (JIT) compilation for target T at runtime.
    linker
        Indicates the device code linker for target triple T.
```

Some targets may have *frontend* and *backend* in one component; in that case, options are merged.

```
T
    Is the target triple device.

options
    Are the options you want to pass to *tool*.
```

**Default**

OFF

No options are passed to a tool.

**Description**

This option enables options to be passed to the specified tool in the device compilation tool chain for the target. It supports OpenMP* offloading.

**IDE Equivalent**

**Windows**

Visual Studio: **Linker > General > Pass <arg> to the backend of target device compiler specified by <triple> for OpenMP offload**

**DPC++ > Language > Pass <arg> to the frontend of target device compiler for OpenMP offload**

**C/C++ > Language [Intel C++] > Pass <arg> to the frontend of target device compiler for OpenMP offload**

**Linker > General > Pass <arg> to the device code linker for OpenMP offload**
Linux

Eclipse: **Linker (Or Intel C++ Linker) > General > Pass <arg> to the backend of target device compiler specified by <triple> for OpenMP offload**

Intel(R) oneAPI DPC++ Compiler > Language > Pass <arg> to the frontend of target device compiler for OpenMP offload

**Intel C++ Compiler > Language > Pass <arg> to the frontend of target device compiler for OpenMP offload**

**Linker (Or Intel C++ Linker) > General > Pass <arg> to the device code linker for OpenMP offload**

Alternate Options

None

**Xsycl-target**

*Enables options to be passed to the specified tool in the device compilation tool chain for the target. This compiler option supports SYCL* offloading. This content is specific to DPC++.*

Syntax

Linux OS:

```
-Xsycl-target=tool "options"
```

Windows OS:

```
-Xsycl-target=tool "options"
```

Arguments

### tool

Can be one of the following:

- **frontend**
  
  Indicates the frontend + middle end of the Standard Portable Intermediate Representation (SPIR-V*)-based device compiler for target triple \( T \).
  
  The middle end is the part of a SPIR-V*-based device compiler that generates SPIR-V*. This SPIR-V* is then passed by the dpcpp driver to the backend of target \( T \).

- **backend**
  
  Indicates Ahead of Time (AOT) compilation for target triple \( T \) and Just in Time (JIT) compilation for target \( T \) at runtime.

- **linker**
  
  Indicates the device code linker for target triple \( T \).

Some targets may have **frontend** and **backend** in one component; in that case, options are merged.

### \( T \)

Is the target triple device.

### options

Are the options you want to pass to **tool**.

Default

OFF No options are passed to a tool.
Description
This option enables options to be passed to the specified tool in the device compilation tool chain for the target. It supports SYCL* offloading.

IDE Equivalent

Visual Studio
Visual Studio: Linker > General > Pass <arg> to the backend of target device compiler specified by <triple> (target-backend)

DPC++ > General > Pass <arg> to the frontend of target device compiler (target-frontend)

Linker > General > Pass <arg> to the device code linker (target-linker)

Eclipse
Eclipse: Linker > General > Pass <arg> to the backend of target device compiler specified by <triple> (target-backend)

Intel(R) oneAPI DPC++ Compiler > General > Pass <arg> to the frontend of target device compiler (target-frontend)

Linker > General > Pass <arg> to the device code linker (target-linker)

Alternate Options
None.

See Also
Xs compiler option

Floating-Point Options
This section contains descriptions for compiler options that pertain to floating-point calculations.

ffp-contract
Controls when the compiler is permitted to form fused floating-point operations, such as fused multiply-add (FMA). Fused operations are allowed to produce more precise results than performing the individual operations separately.

Syntax

Linux OS:
-ffp-contract=keyword

Windows OS:
None

Arguments

<table>
<thead>
<tr>
<th>keyword</th>
<th>Possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast</td>
<td>Fuses floating-point operations across statements.</td>
</tr>
<tr>
<td>on</td>
<td>Fuses floating-point operations within the same statement.</td>
</tr>
</tbody>
</table>
Does not fuse floating-point operations.

Default

-ffp-contract=fast

Fuses floating-point operations across statements.

However, if option -fp-model=strict is specified, the default is -ffp-contract=off.

Description

This option controls when the compiler is permitted to form fused floating-point operations, such as fused multiply-add (FMA). Fused operations are allowed to produce more precise results than performing the individual operations separately.

IDE Equivalent

None

Alternate Options

None

See Also

fp-model, fp compiler option

fimf-absolute-error, Qimf-absolute-error

Defines the maximum allowable absolute error for math library function results. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:

-fimf-absolute-error=value[:funclist]

Windows OS:

/Qimf-absolute-error:value[:funclist]

Arguments

value

Is a positive, floating-point number. Errors in math library function results may exceed the maximum relative error (max-error) setting if the absolute-error is less than or equal to value.

The format for the number is [digits] [.digits] [ { e | E }[sign]digits]

funclist

Is an optional list of one or more math library functions to which the attribute should be applied. If you specify more than one function, they must be separated with commas.

Precision-specific variants like sin and sinf are considered different functions, so you would need to use -fimf-absolute-error=0.00001:sin,sinf (or /Qimf-absolute-error:0.00001:sin,sinf) to specify the maximum allowable absolute error for both the single-precision and double-precision sine functions.
You also can specify the symbol /f to denote single-precision divides, symbol / to
denote double-precision divides, symbol /l to denote extended-precision divides, and
symbol /q to denote quad-precision divides. For example you can specify
-fimf-absolute-error=0.00001:/ or /Qimf-absolute-error: 0.00001:/.

Default
Zero ("0")

An absolute-error setting of 0 means that the function is bound by the relative error
setting. This is the default behavior.

Description
This option defines the maximum allowable absolute error for math library function results.
This option can improve run-time performance, but it may decrease the accuracy of results.
This option only affects functions that have zero as a possible return value, such as log, sin, asin, etc.
The relative error requirements for a particular function are determined by options that set the maximum
relative error (max-error) and precision. The return value from a function must have a relative error less
than the max-error value, or an absolute error less than the absolute-error value.
If you need to define the accuracy for a math function of a certain precision, specify the function name of the
precision that you need. For example, if you want double precision, you can specify :sin; if you want single
precision, you can specify :sinf, as in -fimf-absolute-error=0.00001:sin
or /Qimf-absolute-error:0.00001:sin, or -fimf-absolute-error=0.00001:sqrtf
or /Qimf-absolute-error:0.00001:sqrtf.
If you do not specify any function names, then the setting applies to all functions (and to all precisions).
However, as soon as you specify an individual function name, the setting applies only to the function of
corresponding precision. So, for example, sinf applies only to the single-precision sine function, sin applies
only to the double-precision sine function, sinl applies only to the extended-precision sine function, etc.

NOTE
Many routines in libraries LIBM (Math Library) and SVML (Short Vector Math Library) are
more highly optimized for Intel® microprocessors than for non-Intel microprocessors.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

IDE Equivalent
None

Alternate Options
None

See Also
fimf-accuracy-bits, Qimf-accuracy-bits compiler option
fimf-arch-consistency, Qimf-arch-consistency compiler option
fimf-domain-exclusion, Qimf-domain-exclusion compiler option
fimf-max-error, Qimf-max-error compiler option
**fimf-precision, Qimf-precision**  compiler option

**fimf-use-svml, Qimf-use-svml**  compiler option

### fimf-accuracy-bits, Qimf-accuracy-bits

Defines the relative error for math library function results, including division and square root. This content is specific to C++; it does not apply to DPC++.

#### Syntax

**Linux OS:**

-fimf-accuracy-bits=bits[:funclist]

**Windows OS:**

/Qimf-accuracy-bits:bits[:funclist]

#### Arguments

**bits**

Is a positive, floating-point number indicating the number of correct bits the compiler should use.

The format for the number is [digits] [.digits] [ { e | E }[sign]digits].

**funclist**

Is an optional list of one or more math library functions to which the attribute should be applied. If you specify more than one function, they must be separated with commas.

Precision-specific variants like sin and sinc are considered different functions, so you would need to use -fimf-accuracy-bits=23:sin,sinc (or /Qimf-accuracy-bits:23:sin,sinc) to specify the relative error for both the single-precision and double-precision sine functions.

You also can specify the symbol /f to denote single-precision divides, symbol / to denote double-precision divides, symbol /l to denote extended-precision divides, and symbol /q to denote quad-precision divides. For example you can specify -fimf-accuracy-bits=10.0:/f or /Qimf-accuracy-bits:10.0:/f.

#### Default

-fimf-precision=medium or /Qimf-precision:medium

The compiler uses medium precision when calling math library functions. Note that other options can affect precision; see below for details.

#### Description

This option defines the relative error, measured by the number of correct bits, for math library function results.

The following formula is used to convert bits into ulps: ulps = 2^{p-1-bits}, where p is the number of the target format mantissa bits (24, 53, and 64 for single, double, and long double, respectively).

This option can affect run-time performance and the accuracy of results.

If you need to define the accuracy for a math function of a certain precision, specify the function name of the precision that you need. For example, if you want double precision, you can specify :sin; if you want single precision, you can specify :sinf, as in the following:

- -fimf-accuracy-bits=23:sinf,cosf,logf or /Qimf-accuracy-bits:23:sinf,cosf,logf
If you do not specify any function names, then the setting applies to all functions (and to all precisions). However, as soon as you specify an individual function name, the setting applies only to the function of corresponding precision. So, for example, sinf applies only to the single-precision sine function, sin applies only to the double-precision sine function, sinl applies only to the extended-precision sine function, etc.

There are three options you can use to express the maximum relative error. They are as follows:

- \texttt{-fimf-precision} (Linux*) or \texttt{/Qimf-precision} (Windows*)
- \texttt{-fimf-max-error} (Linux*) or \texttt{/Qimf-max-error} (Windows*)
- \texttt{-fimf-accuracy-bits} (Linux) or \texttt{/Qimf-accuracy-bits} (Windows)

If more than one of these options are specified, the default value for the maximum relative error is determined by the last one specified on the command line.

If none of the above options are specified, the default values for the maximum relative error are determined by the setting of the following options:

- \texttt{-fp-model} (Linux) or \texttt{/fp} (Windows)

\textbf{NOTE}

Many routines in libraries LIBM (Math Library) and SVML (Short Vector Math Library) are more highly optimized for Intel® microprocessors than for non-Intel microprocessors.
Syntax

Linux OS:
-fimf-arch-consistency=value[:funclist]

Windows OS:
/Qimf-arch-consistency:value[:funclist]

Arguments

value
Is one of the logical values "true" or "false".

funclist
Is an optional list of one or more math library functions to which the attribute should be applied. If you specify more than one function, they must be separated with commas.

Precision-specific variants like sin and sinf are considered different functions, so you would need to use
-fimf-arch-consistency=true: sin, sinf (or /Qimf-arch-consistency: true: sin, sinf) to specify consistent results for both the single-precision and double-precision sine functions.

You also can specify the symbol /f to denote single-precision divides, symbol / to denote double-precision divides, symbol /l to denote extended-precision divides, and symbol /q to denote quad-precision divides. For example you can specify
-fimf-arch-consistency=true:/ (or /Qimf-arch-consistency:true:/).

Default

false
Implementations of some math library functions may produce slightly different results on implementations of the same architecture.

Description

This option ensures that the math library functions produce consistent results across different microarchitectural implementations of the same architecture (for example, across different microarchitectural implementations of IA-32 architecture). Consistency is only guaranteed for a single binary. Consistency is not guaranteed across different architectures. For example, consistency is not guaranteed across IA-32 architecture and Intel® 64 architecture.

If you need to define the accuracy for a math function of a certain precision, specify the function name of the precision that you need. For example, if you want double precision, you can specify :sin; if you want single precision, you can specify :sinf, as in -fimf-arch-consistency=true:sin or /Qimf-arch-consistency: true:sin, or -fimf-arch-consistency=false:sqrtf or /Qimf-arch-consistency: false:sqrtf.

If you do not specify any function names, then the setting applies to all functions (and to all precisions). However, as soon as you specify an individual function name, the setting applies only to the function of corresponding precision. So, for example, sinf applies only to the single-precision sine function, sin applies only to the double-precision sine function, sinl applies only to the extended-precision sine function, etc.
The -fimf-arch-consistency (Linux*) and /Qimf-arch-consistency (Windows*) option may decrease run-time performance, but the option will provide bit-wise consistent results on all Intel® processors and compatible, non-Intel processors, regardless of micro-architecture. This option may not provide bit-wise consistent results between different architectures.

**NOTE**
Many routines in libraries LIBM (Math Library) and SVML (Short Vector Math Library) are more highly optimized for Intel® microprocessors than for non-Intel microprocessors.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

**IDE Equivalent**
None

**Alternate Options**
None

**See Also**
- fimf-absolute-error, Qimf-absolute-error compiler option
- fimf-accuracy-bits, Qimf-accuracy-bits compiler option
- fimf-domain-exclusion, Qimf-domain-exclusion compiler option
- fimf-max-error, Qimf-max-error compiler option
- fimf-precision, Qimf-precision compiler option
- fimf-use-svml, Qimf-use-svml compiler option

**fimf-domain-exclusion, Qimf-domain-exclusion**
Indicates the input arguments domain on which math functions must provide correct results. This content is specific to C++; it does not apply to DPC++.

**Syntax**

- **Linux OS:**
  
  `-fimf-domain-exclusion=classList[:funclist]`

- **Windows OS:**
  
  `/Qimf-domain-exclusion:classList[:funclist]`

**Arguments**

- **classList**

  Is one of the following:

  - One or more of the following floating-point value classes you can exclude from the function domain without affecting the correctness of your program. The supported class names are:
extremes  This class is for values which do not lie within the usual domain of arguments for a given function.
nans  This means "x=Nan".
infinities  This means "x=infinities".
denormals  This means "x=denormal".
zeros  This means "x=0".

Each classlist element corresponds to a power of two. The exclusion attribute is the logical or of the associated powers of two (that is, a bitmask).

The following shows the current mapping from classlist mnemonics to numerical values:

<table>
<thead>
<tr>
<th>classlist</th>
<th>numerical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>extremes</td>
<td>1</td>
</tr>
<tr>
<td>nans</td>
<td>2</td>
</tr>
<tr>
<td>infinities</td>
<td>4</td>
</tr>
<tr>
<td>denormals</td>
<td>8</td>
</tr>
<tr>
<td>zeros</td>
<td>16</td>
</tr>
<tr>
<td>none</td>
<td>0</td>
</tr>
<tr>
<td>all</td>
<td>31</td>
</tr>
<tr>
<td>common</td>
<td>15</td>
</tr>
<tr>
<td>other combinations</td>
<td>bitwise OR of the used values</td>
</tr>
</tbody>
</table>

You must specify the integer value that corresponds to the class that you want to exclude.

Note that on excluded values, unexpected results may occur.

- One of the following short-hand tokens:

  none  This means that none of the supported classes are excluded from the domain. To indicate this token, specify 0, as in `-fimf-domain-exclusion=0` (or `/Qimf-domain-exclusion:0`).

  all  This means that all of the supported classes are excluded from the domain. To indicate this token, specify 31, as in `-fimf-domain-exclusion=31` (or `/Qimf-domain-exclusion:31`).

  common  This is the same as specifying extremes,nans,infinities,denormals. To indicate this token, specify 15 (1 + 2 + 4 + 8), as in `-fimf-domain-exclusion=15` (or `/Qimf-domain-exclusion:15`).

funclist  Is an optional list of one or more math library functions to which the attribute should be applied. If you specify more than one function, they must be separated with commas.
Precision-specific variants like sin and sinf are considered different functions, so you would need to use `-fimf-domain-exclusion=4:sin,sinf` (or `/Qimf-domain-exclusion:4:sin,sinf`) to specify infinities for both the single-precision and double-precision sine functions.

You also can specify the symbol `/f` to denote single-precision divides, symbol `/` to denote double-precision divides, symbol `/l` to denote extended-precision divides, and symbol `/q` to denote quad-precision divides. For example, you can specify:

- `-fimf-domain-exclusion=4` or `/Qimf-domain-exclusion:4`
- `-fimf-domain-exclusion=5:/,powf` or `/Qimf-domain-exclusion:5:/,powf`
- `-fimf-domain-exclusion=23:log,logf,/,sin,cosf` or `/Qimf-domain-exclusion:23:log,logf,/,sin,cosf`

If you don’t specify argument `funclist`, the domain restrictions apply to all math library functions.

**Default**
Zero ("0")

The compiler uses default heuristics when calling math library functions.

**Description**
This option indicates the input arguments domain on which math functions must provide correct results. It specifies that your program will function correctly if the functions specified in `funclist` do not produce standard conforming results on the number classes.

This option can affect run-time performance and the accuracy of results. As more classes are excluded, faster code sequences can be used.

If you need to define the accuracy for a math function of a certain precision, specify the function name of the precision that you need. For example, if you want double precision, you can specify :sin; if you want single precision, you can specify :sinf, as in `-fimf-domain-exclusion=denormals:sin` or `/Qimf-domain-exclusion:denormals:sin`, or `-fimf-domain-exclusion=extremes:sqrtf` or `/Qimf-domain-exclusion:extremes:sqrtf`.

If you do not specify any function names, then the setting applies to all functions (and to all precisions). However, as soon as you specify an individual function name, the setting applies only to the function of corresponding precision. So, for example, sinf applies only to the single-precision sine function, sin applies only to the double-precision sine function, sinl applies only to the extended-precision sine function, etc.

**NOTE**
Many routines in libraries LIBM (Math Library) and SVML (Short Vector Math Library) are more highly optimized for Intel® microprocessors than for non-Intel microprocessors.

**Product and Performance Information**
Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

**IDE Equivalent**
None
Alternate Options
None

Example
Consider the following single-precision sequence for function exp2f:

<table>
<thead>
<tr>
<th>Operation:</th>
<th>y = exp2f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy:</td>
<td>1.014 ulp</td>
</tr>
<tr>
<td>Instructions:</td>
<td>4 (2 without fix-up)</td>
</tr>
</tbody>
</table>

The following shows the 2-instruction sequence without the fix-up:

```c
vcvtfxpntps2dq  zmm1 {k1}, zmm0, 0x50      // zmm1 <-- rndToInt(2^24 * x)
vexp223ps       zmm1 {k1}, zmm1            // zmm1 <-- exp2(x)
```

However, the above 2-instruction sequence will not correctly process NaNs. To process NaNs correctly, the following fix-up must be included following the above instruction sequence:

```c
vpxord          zmm2, zmm2, zmm2             // zmm2 <-- 0
vfixupnanps     zmm1 {k1}, zmm0, zmm2 {aaaa} // zmm1 <-- QNaN(x) if x is NaN <F>
```

If the `vfixupnanps` instruction is not included, the sequence correctly processes any arguments except NaN values. For example, the following options generate the 2-instruction sequence:

- `-fimf-domain-exclusion=2:exp2f` <- NaN's are excluded (2 corresponds to NaNs)
- `-fimf-domain-exclusion=6:exp2f` <- NaN's and infinities are excluded (4 corresponds to infinities; 2 + 4 = 6)
- `-fimf-domain-exclusion=7:exp2f` <- NaN's, infinities, and extremes are excluded (1 corresponds to extremes; 2 + 4 + 1 = 7)
- `-fimf-domain-exclusion=15:exp2f` <- NaN's, infinities, extremes, and denormals are excluded (8 corresponds to denormals; 2 + 4 + 1 + 8=15)

If the `vfixupnanps` instruction is included, the sequence correctly processes any arguments including NaN values. For example, the following options generate the 4-instruction sequence:

- `-fimf-domain-exclusion=1:exp2f` <- only extremes are excluded (1 corresponds to extremes)
- `-fimf-domain-exclusion=4:exp2f` <- only infinities are excluded (4 corresponds to infinities)
- `-fimf-domain-exclusion=8:exp2f` <- only denormals are excluded (8 corresponds to denormals)
- `-fimf-domain-exclusion=13:exp2f` <- only extremes, infinities and denormals are excluded (1 + 4 + 8 = 13)

See Also
- `fimf-absolute-error, Qimf-absolute-error` compiler option
- `fimf-accuracy-bits, Qimf-accuracy-bits` compiler option
- `fimf-arch-consistency, Qimf-arch-consistency` compiler option
- `fimf-max-error, Qimf-max-error` compiler option
- `fimf-precision, Qimf-precision` compiler option
- `fimf-use-svml, Qimf-use-svml` compiler option
- **fimf-force-dynamic-target, Qimf-force-dynamic-target**

*Instructs the compiler to use run-time dispatch in calls to math functions. This content is specific to C++; it does not apply to DPC++.*
Syntax

Linux OS:
-fimf-force-dynamic-target[=funclist]

Windows OS:
/Qimf-force-dynamic-target[:funclist]

Arguments

*funclist* is an optional list of one or more math library functions to which the attribute should be applied. If you specify more than one function, they must be separated with commas.

Precision-specific variants like `sin` and `sinf` are considered different functions, so you would need to use
-fimf-force-dynamic-target=sin,sinf
(or /Qimf-force-dynamic-target: sin, sinf) to specify run-time dispatch for both the single-precision and double-precision sine functions.

You also can specify the symbol `/f` to denote single-precision divides, symbol `/` to denote double-precision divides, symbol `/l` to denote extended-precision divides, and symbol `/q` to denote quad-precision divides. For example, you can specify -fimf-force-dynamic-target=/
or /Qimf-force-dynamic-target:/. 

Default

OFF Run-time dispatch is not forced in math libraries calls. The compiler can choose to call a CPU-specific version of a math function if one is available.

Description

This option instructs the compiler to use run-time dispatch in calls to math functions. When this option set to ON, it lets you force run-time dispatch in math libraries calls.

If you want to target multiple CPU families with a single application or you prefer to choose a target CPU at run time, you can force run-time dispatch in math libraries by using this option.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

IDE Equivalent

None

Alternate Options

None
**fimf-max-error, Qimf-max-error**

Defines the maximum allowable relative error for math library function results, including division and square root. This content is specific to C++; it does not apply to DPC++.

**Syntax**

**Linux OS:**

```bash
-fimf-max-error=ulps[:funclist]
```

**Windows OS:**

```bash
/Qimf-max-error:ulps[:funclist]
```

**Arguments**

- **ulps**
  
  Is a positive, floating-point number indicating the maximum allowable relative error the compiler should use. The format for the number is [digits] [.digits] [ ± e | E ][sign]digits.

- **funclist**
  
  Is an optional list of one or more math library functions to which the attribute should be applied. If you specify more than one function, they must be separated with commas.

  Precision-specific variants like sin and sinf are considered different functions, so you would need to use
  ```bash
  -fimf-max-error=4.0:sin,sinf
  (or /Qimf-max-error=4.0:sin,sinf)
  ```
  to specify the maximum allowable relative error for both the single-precision and double-precision sine functions.

  You also can specify the symbol /f to denote single-precision divides, symbol / to denote double-precision divides, symbol /l to denote extended-precision divides, and symbol /q to denote quad-precision divides. For example you can specify
  ```bash
  -fimf-max-error=4.0:/
  (or /Qimf-max-error=4.0: /).
  ```

**Default**

- **-fimf-precision=medium or /Qimf-precision:medium**

  The compiler uses medium precision when calling math library functions. Note that other options can affect precision; see below for details.

**Description**

This option defines the maximum allowable relative error, measured in ulps, for math library function results. This option can affect run-time performance and the accuracy of results.

If you need to define the accuracy for a math function of a certain precision, specify the function name of the precision that you need. For example, if you want double precision, you can specify :sin; if you want single precision, you can specify :sinf, as in `-fimf-max-error=4.0:sin` or `/Qimf-max-error:4.0:sin`, or `-fimf-max-error=4.0:sqrtf` or `/Qimf-max-error:4.0:sqrtf`. 
If you do not specify any function names, then the setting applies to all functions (and to all precisions). However, as soon as you specify an individual function name, the setting applies only to the function of corresponding precision. So, for example, `sinf` applies only to the single-precision sine function, `sin` applies only to the double-precision sine function, `sinl` applies only to the extended-precision sine function, etc.

There are three options you can use to express the maximum relative error. They are as follows:

- `-fimf-precision` (Linux*) or `/Qimf-precision` (Windows*)
- `-fimf-max-error` (Linux*) or `/Qimf-max-error` (Windows*)
- `-fimf-accuracy-bits` (Linux) or `/Qimf-accuracy-bits` (Windows)

If more than one of these options are specified, the default value for the maximum relative error is determined by the last one specified on the command line.

If none of the above options are specified, the default values for the maximum relative error are determined by the setting of the following options:

- `-fp-model` (Linux) or `/fp` (Windows)

**NOTE**
Many routines in libraries LIBM (Math Library) and SVML (Short Vector Math Library) are more highly optimized for Intel® microprocessors than for non-Intel microprocessors.

---

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

---

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

- `fimf-absolute-error, Qimf-absolute-error` compiler option
- `fimf-accuracy-bits, Qimf-accuracy-bits` compiler option
- `fimf-arch-consistency, Qimf-arch-consistency` compiler option
- `fimf-domain-exclusion, Qimf-domain-exclusion` compiler option
- `fimf-precision, Qimf-precision` compiler option
- `fimf-use-svml, Qimf-use-svml` compiler option

**fimf-precision, Qimf-precision**

Lets you specify a level of accuracy (precision) that the compiler should use when determining which math library functions to use. This content is specific to C++; it does not apply to DPC++.

**Syntax**

**Linux OS:**

- `-fimf-precision[=value[:funclist]]`
Windows OS:
/Qimf-precision[:value[:funclist]]

Arguments

value

Is one of the following values denoting the desired accuracy:

- **high**: This is equivalent to max-error = 1.0.
- **medium**: This is equivalent to max-error = 4; this is the default setting if the option is specified and value is omitted.
- **low**: This is equivalent to accuracy-bits = 11 for single-precision functions; accuracy-bits = 26 for double-precision functions.

In the above explanations, max-error means option -fimf-max-error (Linux*) or /Qimf-max-error (Windows*); accuracy-bits means option -fimf-accuracy-bits (Linux*) or /Qimf-accuracy-bits (Windows*).

funclist

Is an optional list of one or more math library functions to which the attribute should be applied. If you specify more than one function, they must be separated with commas.

Precision-specific variants like sin and sinf are considered different functions, so you would need to use

- **-fimf-precision=high:sin,sinf**
  (or **/Qimf-precision:high:sin,sinf**) to specify high precision for both the single-precision and double-precision sine functions.

You also can specify the symbol /f to denote single-precision divides, symbol / to denote double-precision divides, symbol /l to denote extended-precision divides, and symbol /q to denote quad-precision divides. For example you can specify **-fimf-precision=low:/** or **/Qimf-precision:low:/** and **-fimf-precision=low:/f** or **/Qimf-precision:low:/f**.

Default

**medium**

The compiler uses medium precision when calling math library functions. Note that other options can affect precision; see below for details.

Description

This option lets you specify a level of accuracy (precision) that the compiler should use when determining which math library functions to use.

This option can be used to improve run-time performance if reduced accuracy is sufficient for the application, or it can be used to increase the accuracy of math library functions selected by the compiler.

In general, using a lower precision can improve run-time performance and using a higher precision may reduce run-time performance.
If you need to define the accuracy for a math function of a certain precision, specify the function name of the precision that you need. For example, if you want double precision, you can specify :sin; if you want single precision, you can specify :sinf, as in -fimf-precision=low:sin or /Qimf-precision:low:sin, or -fimf-precision=high:sqrtf or /Qimf-precision:high:sqrtf.

If you do not specify any function names, then the setting applies to all functions (and to all precisions). However, as soon as you specify an individual function name, the setting applies only to the function of corresponding precision. So, for example, sinf applies only to the single-precision sine function, sin applies only to the double-precision sine function, sinl applies only to the extended-precision sine function, etc.

There are three options you can use to express the maximum relative error. They are as follows:

- -fimf-precision (Linux*) or /Qimf-precision (Windows*)
- -fimf-max-error (Linux*) or /Qimf-max-error (Windows*)
- -fimf-accuracy-bits (Linux) or /Qimf-accuracy-bits (Windows)

If more than one of these options are specified, the default value for the maximum relative error is determined by the last one specified on the command line.

If none of the above options are specified, the default values for the maximum relative error are determined by the setting of the following options:

- -fp-model (Linux) or /fp (Windows)

**NOTE**

Many routines in libraries LIBM (Math Library) and SVML (Short Vector Math Library) are more highly optimized for Intel® microprocessors than for non-Intel microprocessors.

---

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

- fimf-absolute-error, Qimf-absolute-error compiler option
- fimf-accuracy-bits, Qimf-accuracy-bits compiler option
- fimf-arch-consistency, Qimf-arch-consistency compiler option
- fimf-domain-exclusion, Qimf-domain-exclusion compiler option
- fimf-max-error, Qimf-max-error compiler option
- fp-model, fp compiler option
- fimf-use-svml, Qimf-use-svml compiler option

**fimf-use-svml, Qimf-use-svml**

Instructs the compiler to use the Short Vector Math Library (SVML) rather than the Intel® Math Library (LIBM) to implement math library functions.
**Syntax**

Linux OS:
- `fimf-use-svml=value[:funclist]`

Windows OS:
/`Qimf-use-svml:value[:funclist]`

**Arguments**

`funclist`  
Is an optional list of one or more math library functions to which the attribute should be applied. If you specify more than one function, they must be separated with commas.

Precision-specific variants like `sin` and `sinf` are considered different functions, so you would need to use
- `fimf-use-svmlt=true:sin,sinf`  
(or `/Qimf-use-svml:true:sin,sinf`) to specify that both the single-precision and double-precision sine functions should use SVML.

**Default**

false  
Math library functions are implemented using the Intel® Math Library, though other compiler options may give the compiler the flexibility to implement math library functions with either LIBM or SVML.

**Description**

This option instructs the compiler to implement math library functions using the Short Vector Math Library (SVML). When you specify `fimf-use-svml=true` or `/Qimf-use-svml:true`, the specific SVML variant chosen is influenced by other compiler options such as `-fimf-precision` (Linux*) or `/Qimf-precision` (Windows*) and `-fp-model` (Linux) or `/fp` (Windows). This option has no effect on math library functions that are implemented in LIBM but not in SVML.

In value-safe settings of option `-fp-model` (Linux) or option `/fp` (Windows) such as precise, this option causes a slight decrease in the accuracy of math library functions, because even the high accuracy SVML functions are slightly less accurate than the corresponding functions in LIBM. Additionally, the SVML functions might not accurately raise floating-point exceptions, do not maintain errno, and are designed to work correctly only in round-to-nearest-even rounding mode.

The benefit of using `-fimf-use-svml=true` or `/Qimf-use-svml:true` with value-safe settings of `-fp-model` (Linux) or `/fp` (Windows) is that it can significantly improve performance by enabling the compiler to efficiently vectorize loops containing calls to math library functions.

If you need to use SVML for a specific math function of a certain precision, specify the function name of the precision that you need. For example, if you want double precision, you can specify :`sin`; if you want single precision, you can specify :`sqrtf`, as in `-fimf-use-svml=true:sin` or `/Qimf-use-svml=true:sin`, or `-fimf-use-svml=false:sqrtf` or `/Qimf-use-svml=false:sqrtf`.

If you do not specify any function names, then the setting applies to all functions (and to all precisions). However, as soon as you specify an individual function name, the setting applies only to the function of corresponding precision. So, for example, `sinf` applies only to the single-precision sine function, `sin` applies only to the double-precision sine function, `sinl` applies only to the extended-precision sine function, etc.
NOTE
Since SVML functions may raise unexpected floating-point exceptions, be cautious about using features that enable trapping on floating-point exceptions.

Product and Performance Information

| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. |
| Notice revision #20201201 |

IDE Equivalent
None

Alternate Options
None

See Also
fp-model, fp compiler option
fma, Qfma

fma, Qfma
Determines whether the compiler generates fused multiply-add (FMA) instructions if such instructions exist on the target processor.

Syntax

Linux OS:
-fma
-no-fma

Windows OS:
/Qfma
/Qfma-

Arguments
None

Default
-fma
or /Qfma

If the instructions exist on the target processor, the compiler generates fused multiply-add (FMA) instructions.

However, if you specify -fp-model strict (Linux*) or /fp(strict (Windows*), but do not explicitly specify -fma or /Qfma, the default is -no-fma or /Qfma-.

Description

This option determines whether the compiler generates fused multiply-add (FMA) instructions if such instructions exist on the target processor. When the [Q]fma option is specified, the compiler may generate FMA instructions for combining multiply and add operations. When the negative form of the [Q]fma option is specified, the compiler must generate separate multiply and add instructions with intermediate rounding.
This option has no effect unless setting CORE-AVX2 or higher is specified for option [Q]x,-march (Linux), or /arch (Windows).

**IDE Equivalent**

None

**See Also**

fp-model, fp compiler option
x, Qx compiler option
march compiler option
arch compiler option

**fp-model, fp**

*Controls the semantics of floating-point calculations.*

**Syntax**

Linux OS:

```
-fp-model=keyword
```

Windows OS:

```
/fp:keyword
```

**Arguments**

<table>
<thead>
<tr>
<th>keyword</th>
<th>Specifies the semantics to be used. Possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>precise</td>
<td>Disables optimizations that are not value-safe on floating-point data.</td>
</tr>
<tr>
<td>fast[=1</td>
<td>2]</td>
</tr>
<tr>
<td></td>
<td>There is currently no difference between fast=1 and fast=2.</td>
</tr>
<tr>
<td>strict</td>
<td>Enables precise, disables contractions, and enables pragma stdc fenv_access.</td>
</tr>
</tbody>
</table>

**Default**

```
-fp-model=fast
```

The compiler uses more aggressive optimizations on floating-point calculations.

**Description**

This option controls the semantics of floating-point calculations.

The floating-point (FP) environment is a collection of registers that control the behavior of FP machine instructions and indicate the current FP status. The floating-point environment may include rounding-mode controls, exception masks, flush-to-zero controls, exception status flags, and other floating-point related features.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-fp-model=precise</code> or <code>/fp:precise</code></td>
<td>Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations. It disables optimizations that can change the result of floating-point calculations, which is required for strict ANSI conformance. These semantics ensure the reproducibility of floating-point computations for serial code, including code vectorized or auto-parallelized by the compiler, but they may slow performance. They do not ensure value safety or run-to-run reproducibility of other parallel code. Run-to-run reproducibility for floating-point reductions in OpenMP* code may be obtained for a fixed number of threads through the <code>KMP_DETERMINISTIC_REDUCTION</code> environment variable. For more information about this environment variable, see topic &quot;Supported Environment Variables&quot;. The compiler assumes the default floating-point environment; you are not allowed to modify it.</td>
</tr>
<tr>
<td>`-fp-model=fast[=1</td>
<td>2]<code>or</code>/fp:fast[=1</td>
</tr>
<tr>
<td><code>-fp-model=strict</code> or <code>/fp:strict</code></td>
<td>Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations and enables floating-point exception semantics. This is the strictest floating-point model. The compiler does not assume the default floating-point environment; you are allowed to modify it.</td>
</tr>
</tbody>
</table>

The `-fp-model` and `/fp` options determine the setting for the maximum allowable relative error for math library function results (max-error) if none of the following options are specified:

- `-fimf-accuracy-bits (Linux*) or /Qimf-accuracy-bits (Windows*)`
- `-fimf-max-error (Linux) or /Qimf-max-error (Windows)`
- `-fimf-precision (Linux) or /Qimf-precision (Windows)`

NOTE
In Microsoft* Visual Studio, when you create a Microsoft* Visual C++ project, option /fp:precise is set by default. It sets the floating-point model to improve consistency for floating-point operations by disabling certain optimizations that may reduce performance. To set the option back to the general default /fp:fast, change the IDE project property for Floating Point Model to Fast.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

**IDE Equivalent**

**Visual Studio**
Visual Studio: **Code Generation>Floating Point Model**

**Code Generation>Enable Floating Point Exceptions**

**Code Generation> Floating Point Expression Evaluation**

**Eclipse**
Eclipse: **Floating Point > Floating Point Model**

**Alternate Options**
None

**See Also**
- compiler option (specifically O0)
- compiler option
- compiler option
- compiler option
- compiler option
- compiler option

**Supported Environment Variables**
The article titled: Consistency of Floating-Point Results using the Intel® Compiler, which is located in https://software.intel.com/content/www/us/en/develop/articles/consistency-of-floating-point-results-using-the-intel-compiler.html

**fp-speculation, Qfp-speculation**
* Tells the compiler the mode in which to speculate on floating-point operations.

**Syntax**

**Linux OS:**
- `-fp-speculation=mode`

**Windows OS:**
- `/Qfp-speculation:mode`
Arguments

mode

Is the mode for floating-point operations. Possible values are:

- **fast**: Tells the compiler to speculate on floating-point operations.
- **safe**: Tells the compiler to disable speculation if there is a possibility that the speculation may cause a floating-point exception.
- **strict**: Tells the compiler to disable speculation on floating-point operations.
- **off**: This is the same as specifying strict.

Default

```
-fp-speculation=fast
or /Qfp-speculation:fast
```

The compiler speculates on floating-point operations. This is also the behavior when optimizations are enabled. However, if you specify no optimizations (-O0 on Linux*; /Od on Windows*), the default is `-fp-speculation=safe (Linux*) or /Qfp-speculation:safe (Windows*)`.

Description

This option tells the compiler the mode in which to speculate on floating-point operations.

Disabling speculation may prevent the vectorization of some loops containing conditionals. For an example, see the article titled: **Diagnostic 15326: loop was not vectorized: implied FP exception model prevents vectorization**, which is located in [https://software.intel.com/content/www/us/en/develop/articles/fdiag15326.html](https://software.intel.com/content/www/us/en/develop/articles/fdiag15326.html).

IDE Equivalent

**Visual Studio**

Visual Studio: **Optimization > Floating-Point Speculation**

**Eclipse**

Eclipse: **Floating Point > Floating-Point Speculation**

Alternate Options

None

Inlining Options

This section contains descriptions for compiler options that pertain to inlining.

**fgnu89-inline**

_Tells the compiler to use C89 semantics for inline functions when in C99 mode._

Syntax

**Linux OS:**

```
-fgnu89-inline
```
**Windows OS:**
None

**Arguments**
None

**Default**
OFF

**Description**
This option tells the compiler to use C89 semantics for inline functions when in C99 mode.

**IDE Equivalent**
None

**Alternate Options**
None

**finline**
*Tells the compiler to inline functions declared with \_\_inline and perform C++ inlining.*

**Syntax**

**Linux OS:**
-finline
-fno-inline

**Windows OS:**
None

**Arguments**
None

**Default**
-fno-inline

The compiler does not inline functions declared with __inline.

**Description**
This option tells the compiler to inline functions declared with __inline and perform C++ inlining.

**IDE Equivalent**
None

**Alternate Options**
None

**finline-functions**
*Enables function inlining for single file compilation.*
**Syntax**

**Linux OS:**
- `-finline-functions`
- `-fno-inline-functions`

**Windows OS:**
None

**Arguments**
None

**Default**

- `-finline-functions`  
  Interprocedural optimizations occur. However, if you specify `-O0`, the default is OFF.

**Description**

This option enables function inlining for single file compilation.

It enables the compiler to perform inline function expansion for calls to functions defined within the current source file.

The compiler applies a heuristic to perform the function expansion.

**IDE Equivalent**
None

**Alternate Options**
None

**Output, Debug, and Precompiled Header (PCH) Options**

This section contains descriptions for compiler options that pertain to output, debugging, or precompiled headers (PCH).

**c**

Prevents linking.

**Syntax**

**Linux OS:**
- `-c`

**Windows OS:**
- `/c`

**Arguments**
None

**Default**

OFF  
Linking is performed.
**Description**
This option prevents linking. Compilation stops after the object file is generated.
The compiler generates an object file for each C or C++ source file or preprocessed source file. It also takes an assembler file and invokes the assembler to generate an object file.

**IDE Equivalent**
None

**Alternate Options**
None

**debug (Linux*)**
*Enables or disables generation of debugging information. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**
- `debug [keyword]`

**Windows OS:**
None

**Arguments**
*keyword* is the type of debugging information to be generated. Possible values are:

- **none**: Disables generation of debugging information.
- **full or all**: Generates complete debugging information.
- **minimal**: Generates line number information for debugging.
- **[no]emit_column**: Determines whether the compiler generates column number information for debugging.
- **[no]expr-source-pos**: Determines whether the compiler generates source position information at the expression level of granularity.
- **[no]inline-debug-info**: Determines whether the compiler generates enhanced debug information for inlined code.
- **[no]pubnames**: Determines whether the compiler generates a DWARF debug_pubnames section.
- **[no]semantic-stepping**: Determines whether the compiler generates enhanced debug information useful for breakpoints and stepping.
- **[no]variable-locations**: Determines whether the compiler generates enhanced debug information useful in finding scalar local variables.
- **extended**: Generates complete debugging information and also sets keyword values semantic-stepping and variable-locations.
Determines whether the compiler generates parallel debug code instrumentations useful for thread data sharing and reentrant call detection.

For information on the non-default settings for these keywords, see the Description section.

Default

Normally, the default is `-debug none` and no debugging information is generated. However, on Linux*, the `-debug inline-debug-info` option will be enabled by default if you compile with optimizations (option `-O2` or higher) and debugging is enabled (option `-g`).

Description

This option enables or disables generation of debugging information.

By default, enabling debugging, will disable optimization. To enable both debugging and optimization use the `-debug` option together with one of the optimization level options (`-O3`, `-O2` or `-O3`).

Keywords `semantic-stepping`, `inline-debug-info`, `variable-locations`, and `extended` can be used in combination with each other. If conflicting keywords are used in combination, the last one specified on the command line has precedence.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-debug none</code></td>
<td>Disables generation of debugging information.</td>
</tr>
<tr>
<td><code>-debug full</code> or <code>-debug all</code></td>
<td>Generates complete debugging information. It is the same as specifying <code>-debug</code> with no keyword.</td>
</tr>
<tr>
<td><code>-debug minimal</code></td>
<td>Generates line number information for debugging.</td>
</tr>
<tr>
<td><code>-debug emit_column</code></td>
<td>Generates column number information for debugging.</td>
</tr>
<tr>
<td><code>-debug expr-source-pos</code></td>
<td>Generates source position information at the statement level of granularity.</td>
</tr>
<tr>
<td><code>-debug inline-debug-info</code></td>
<td>Generates enhanced debug information for inlined code. On inlined functions, symbols are (by default) associated with the caller. This option causes symbols for inlined functions to be associated with the source of the called function.</td>
</tr>
<tr>
<td><code>-debug pubnames</code></td>
<td>The compiler generates a DWARF debug_pubnames section. This provides a means to list the names of global objects and functions in a compilation unit.</td>
</tr>
<tr>
<td><code>-debug semantic-stepping</code></td>
<td>Generates enhanced debug information useful for breakpoints and stepping. It tells the debugger to stop only at machine instructions that achieve the final effect of a source statement. For example, in the case of an assignment statement, this might be a store instruction that assigns a value to a program variable; for a function call, it might be the machine instruction that executes the call. Other instructions generated for those source statements are not displayed during stepping. This option has no impact unless optimizations have also been enabled.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-debug variable-locations</td>
<td>Generates enhanced debug information useful in finding scalar local variables. It uses a feature of the Dwarf object module known as &quot;location lists&quot;. This feature allows the run-time locations of local scalar variables to be specified more accurately; that is, whether, at a given position in the code, a variable value is found in memory or a machine register.</td>
</tr>
<tr>
<td>-debug extended</td>
<td>Sets keyword values semantic-stepping and variable-locations. It also tells the compiler to include column numbers in the line information. Generates complete debugging information and also sets keyword values semantic-stepping and variable-locations. This is a more powerful setting than -debug full or -debug all.</td>
</tr>
<tr>
<td>-debug parallel</td>
<td>Generates parallel debug code instrumentations needed for the thread data sharing and reentrant call detection. This content is specific to C++; it does not apply to DPC++. For shared data and reentrancy detection, option -qopenmp must be set.</td>
</tr>
</tbody>
</table>

On Linux* systems, debuggers read debug information from executable images. As a result, information is written to object files and then added to the executable by the linker.

**IDE Equivalent**

**Windows**

Visual Studio: None

**Linux**

Eclipse: Advanced Debugging > Enable Parallel Debug Checks (-debug parallel)

Debug > Enable Expanded Line Number Information (-debug expr-source-pos)

**Alternate Options**

For -debug full, -debug all, or -debug

Linux: -g

Windows: /debug:full, /debug:all, or /debug

**See Also**

dbg (Windows*) compiler option
gopenmp, Qopenmp compiler option
ddebug (Windows*)

Enables or disables generation of debugging information. This content is specific to C++; it does not apply to DPC++.

**Syntax**

Linux OS:

None
Windows OS:

/debug[:keyword]

Arguments

keyword  Is the type of debugging information to be generated. Possible values are:

none       Disables generation of debugging information.
full or all Generates complete debugging information.
minimal    Generates line number information for debugging.
partial    Deprecated. Generates global symbol table information needed for linking.

[no]expr-source-pos  Determines whether the compiler generates source position information at the expression level of granularity.
[no]inline-debug-info  Determines whether the compiler generates enhanced debug information for inlined code.

For information on the non-default settings for these keywords, see the Description section.

Default

/debug:none  This is the default on the command line and for a release configuration in the IDE.
/debug:all   This is the default for a debug configuration in the IDE.

Description

This option enables or disables generation of debugging information. It is passed to the linker.

By default, enabling debugging, will disable optimization. To enable both debugging and optimization use the /debug option together with one of the optimization level options (/O3, /O2 or /O1).

If conflicting keywords are used in combination, the last one specified on the command line has precedence.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/debug:none</td>
<td>Disables generation of debugging information.</td>
</tr>
<tr>
<td>/debug:full or /debug:all</td>
<td>Generates complete debugging information. It produces symbol table information needed for full symbolic debugging of unoptimized code and global symbol information needed for linking. It is the same as specifying /debug with no keyword.</td>
</tr>
<tr>
<td>/debug:minimal</td>
<td>Generates line number information for debugging.</td>
</tr>
<tr>
<td>/debug:partial</td>
<td>Generates global symbol table information needed for linking, but not local symbol table information needed for debugging. This option is deprecated and is not available in the IDE.</td>
</tr>
<tr>
<td>/debug:expr-source-pos</td>
<td>Generates source position information at the statement level of granularity.</td>
</tr>
<tr>
<td>/debug:inline-debug-info</td>
<td>Generates enhanced debug information for inlined code.</td>
</tr>
</tbody>
</table>
### Option

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>On inlined functions, symbols are (by default) associated with the caller. This option causes symbols for inlined functions to be associated with the source of the called function.</td>
</tr>
</tbody>
</table>

### IDE Equivalent

**Windows**

Visual Studio: **Debugging > Enable Expanded Line Number Information** (`/debug:expr-source-pos`)

**Linux**

Eclipse: None

### Alternate Options

For `/debug:all` or `/debug`  

Linux: None  

Windows: `/Zi`

### See Also

`debug` (Linux*) compiler option

### Fa

Specifies that an assembly listing file should be generated.

### Syntax

**Linux OS:**

`-Fa[filename|dir]`

**Windows OS:**

`/Fa[filename|dir]`

### Arguments

- `filename`  
  Is the name of the assembly listing file.

- `dir`  
  Is the directory where the file should be placed. It can include `filename`.

### Default

**OFF**  
No assembly listing file is produced.

### Description

This option specifies that an assembly listing file should be generated (optionally named `filename`).

### IDE Equivalent

**Windows**

Visual Studio: **Output Files > ASM List Location**
Linux

Eclipse: **Output > Generate Assembler Source and Binary Files**

**Alternate Options**

Linux: `-S`

Windows: `/S`

**FA**

*Specifies the contents of an assembly listing file.*

**Syntax**

Linux OS:

None

Windows OS:

`/FA`

**Arguments**

None

**Default**

OFF

No source or machine code annotations appear in the assembly listing file, if one is produced.

**Description**

This option produces an assembly listing without source or machine code annotations.

To use this option, you must also specify option `/Fa`, which causes an assembly listing to be generated.

**IDE Equivalent**

**Windows**

Visual Studio: **Output Files > Assembler Output**

**Linux**

Eclipse: None

**Alternate Options**

None

**fasm-blocks**

*Enables the use of blocks and entire functions of assembly code within a C or C++ file.*

**Syntax**

Linux OS:

`-fasm-blocks`

Windows OS:

None
Arguments
None

Default
OFF The compiler allows a GNU*-style inline assembly format.

Description
This option enables the use of blocks and entire functions of assembly code within a C or C++ file.
It allows a Microsoft* MASM-style inline assembly block not a GNU*-style inline assembly block.

IDE Equivalent
None

Alternate Options
-use-msasm

FC
Displays the full path of source files passed to the compiler in diagnostics.

Syntax
Linux OS:
None

Windows OS:
/FC

Arguments
None

Default
OFF The compiler does not display the full path of source files passed to the compiler in diagnostics.

Description
Displays the full path of source files passed to the compiler in diagnostics. This option is supported with Microsoft Visual Studio .NET 2003* or newer.

IDE Equivalent
Windows
Visual Studio: Advanced > Use Full Paths

Alternate Options
None

Fd
Lets you specify a name for a program database (PDB) file created by the compiler.
Syntax

Linux OS:
None

Windows OS:
/Fd[:\filename]

Arguments

filename
Is the name for the PDB file. It can include a path. If you do not specify a file extension, the extension .pdb is used.

Default

OFF
No PDB file is created unless you specify option /Zi. If you specify option /Zi and /Fd, the default filename is vc\x\0.pdb, where \x\ represents the version of Visual C++, for example vc100.pdb.

Description

This option lets you specify a name for a program database (PDB) file that is created by the compiler. A program database (PDB) file holds debugging and project state information that allows incremental linking of a Debug configuration of your program. A PDB file is created when you build with option /Zi. Option /Fd has no effect unless you specify option /Zi.

IDE Equivalent

Windows
Visual Studio: Output Files > Program Database File Name

Linux
Eclipse: None

Alternate Options

None

See Also
Zi, Z7, ZI compiler option
pdbfile compiler option

FD
Generates file dependencies related to the Microsoft* C/C++ compiler.

Syntax

Linux OS:
None

Windows OS:
/FD
Arguments
None

Default
OFF The compiler does not generate Microsoft C/C++-related file dependencies.

Description
This option generates file dependencies related to the Microsoft* C/C++ compiler. It invokes the Microsoft C/C++ compiler and passes the option to it.

IDE Equivalent
None

Alternate Options
None

Fe
Specifies the name for a built program or dynamic-link library.

Syntax
Linux OS:
None

Windows OS:
/Fe[[:]filename|dir]

Arguments

filename Is the name for the built program or dynamic-link library.

dir Is the directory where the built program or dynamic-link library should be placed. It can include file.

Default
OFF The name of the file is the name of the first source file on the command line with file extension .exe, so file.f becomes file.exe.

Description
This option specifies the name for a built program (.EXE) or a dynamic-link library (.DLL).

You can use this option to specify an alternate name for an executable file. This is especially useful when compiling and linking a set of input files. You can use the option to give the resulting file a name other than that of the first input file (source or object) on the command line.

IDE Equivalent
None

Alternate Options
Linux: –o
Example

In the following example, the command produces an executable file named outfile.exe as a result of compiling and linking three files: one object file and two C++ source files.

```plaintext
prompt> icx /Feoutfile.exe file1.obj file2.cpp file3.cpp  ! specific to C++
prompt> dpcpp-cl /Feoutfile.exe file1.obj file2.cpp file3.cpp  ! specific to DPC++
```

This command produces an executable file named file1.exe when the /Fe option is not used.

See Also

- compiler option

Fo

Specifies the name for an object file.

Syntax

Linux OS:
See option o.

Windows OS:
/Fo[[:]filename|dir]

Arguments

- `filename`
  Is the name for the object file.

- `dir`
  Is the directory where the object file should be placed. It can include `filename`.

Default

OFF

An object file has the same name as the name of the first source file and a file extension of .obj.

Description

This option specifies the name for an object file.

IDE Equivalent

Windows

Visual Studio: Output Files > Object File Name

Alternate Options

None

See Also

- compiler option

Fp

Lets you specify an alternate path or file name for precompiled header files.
Syntax

Linux OS:
None

Windows OS:
/Fp{filename|dir}

Arguments

filename
Is the name for the precompiled header file.

dir
Is the directory where the precompiled header file should be placed. It can include filename.

Default

OFF
The compiler does not create or use precompiled headers unless you tell it to do so.

Description

This option lets you specify an alternate path or file name for precompiled header files.

IDE Equivalent

Windows

Visual Studio: Precompiled Headers > Precompiled Header Output File

Linux

Eclipse: None

Alternate Options

None

ftrapuv, Qtrapuv

Initializes stack local variables to an unusual value to aid error detection.

Syntax

Linux OS:
-ftrapuv

Windows OS:
/Qtrapuv (C++ only)

Windows OS:
None (DPC++ only)

Arguments

None
Default

OFF The compiler does not initialize local variables.

Description

This option initializes stack local variables to an unusual value to aid error detection. Normally, these local variables should be initialized in the application. It also unmasks the floating-point invalid exception.

The option sets any uninitialized local variables that are allocated on the stack to a value that is typically interpreted as a very large integer or an invalid address. References to these variables are then likely to cause run-time errors that can help you detect coding errors.

This option sets option -g (Linux*) and /zi or /z7 (Windows*), which changes the default optimization level from O2 to -O0 (Linux) or /Od (Windows). You can override this effect by explicitly specifying an O option setting.

For more details on using options -ftrapuv and /Qtrapuv (C++) with compiler option O, see the article in Intel® Developer Zone titled Don’t optimize when using -ftrapuv for uninitialized variable detection, which is located in https://software.intel.com/content/www/us/en/develop/articles/dont-optimize-when-using-ftrapuv-for-uninitialized-variable-detection.html.

Another way to detect uninitialized local scalar variables is by specifying keyword uninit for option check.

IDE Equivalent

Windows

Visual Studio: None

Linux

Eclipse: Run-Time > Initialize Stack Variables to an Unusual Value

Alternate Options

None

See Also

g compiler option
Zi, Z7, ZI compiler option
O compiler option

fverbose-asm

Produces an assembly listing with compiler comments, including options and version information.

Syntax

Linux OS:

-fverbose-asm
-fno-verbose-asm

Windows OS:

None

Arguments

None
Default

-fno-verbose-as

No source code annotations appear in the assembly listing file, if one is produced.

Description

This option produces an assembly listing file with compiler comments, including options and version information.

To use this option, you must also specify -S, which sets -fverbose-asm.

If you do not want this default when you specify -S, specify -fno-verbose-asm.

IDE Equivalent

None

Alternate Options

None

See Also

s compiler option

g

Tells the compiler to generate a level of debugging information in the object file.

Syntax

Linux OS:

-g[n]

Windows OS:

See option Zi, Z7, ZI.

Arguments

n

Is the level of debugging information to be generated. Possible values are:

0

Disables generation of symbolic debug information.

1

Produces minimal debug information for performing stack traces.

2

Produces complete debug information. This is the same as specifying -g with no n.

3

Produces extra information that may be useful for some tools.

Default

-g or -g2

The compiler produces complete debug information.
Description
Option `-g` tells the compiler to generate symbolic debugging information in the object file, which increases the size of the object file.

The compiler does not support the generation of debugging information in assemblable files. If you specify this option, the resulting object file will contain debugging information, but the assemblable file will not.

This option turns off option `-O2` and makes option `-O0` the default unless option `-O2` (or higher) is explicitly specified in the same command line.

Specifying the `-g` or `-O0` option sets the `-fno-omit-frame-pointer` option.

This content is specific to C++; it does not apply to DPC++. On Linux*, the `-debug inline-debug-info` option will be enabled by default if you compile with optimizations (option `-O2` or higher) and debugging is enabled (option `-g`).

**NOTE**
When option `-g` is specified, debugging information is generated in the DWARF Version 3 format. Older versions of some analysis tools may require applications to be built with the `-gdwarf-2` option to ensure correct operation.

IDE Equivalent

Visual Studio
Visual Studio: None

Eclipse
Eclipse: **General > Include Debug Information**

Alternate Options

Linux: None
Windows: `/Zi, /Z7, /ZI`

See Also
gdwarf compiler option  
`Zi, Z7, ZI` compiler option  
debug (Linux*) compiler option

gdwarf
*Lets you specify a DWARF Version format when generating debug information.*

Syntax

Linux OS:
- `-gdwarf-n`

Windows OS:
None
Arguments

\( n \)

Is a value denoting the DWARF Version format to use. Possible values are:

2

Generates debug information using the DWARF Version 2 format.

3

Generates debug information using the DWARF Version 3 format.

4

Generates debug information using the DWARF Version 4 format. This setting is only available on Linux*.

Default

OFF

No debug information is generated. However, if compiler option \(-g\) is specified, debugging information is generated in the DWARF Version 3 format.

Description

This option lets you specify a DWARF Version format when generating debug information.

Note that older versions of some analysis tools may require applications to be built with the \(-gdwarf-2\) option to ensure correct operation.

IDE Equivalent

None

Alternate Options

None

See Also

g compiler option

Gm

Enables a minimal rebuild.

Syntax

Linux OS:

None

Windows OS:

/Gm

Arguments

None

Default

OFF

Minimal rebuilds are disabled.

Description

This option enables a minimal rebuild.
IDE Equivalent

Windows
Visual Studio: Code Generation > Enable Minimal Rebuild

Linux
Eclipse: None

Alternate Options
None

grecord-gcc-switches
Causes the command line options that were used to invoke the compiler to be appended to the DW_AT_producer attribute in DWARF debugging information.

Syntax

Linux OS:
-grecord-gcc-switches

Windows OS:
None

Arguments
None

Default
OFF

The command line options that were used to invoke the compiler are not appended to the DW_AT_producer attribute in DWARF debugging information.

Description
This option causes the command line options that were used to invoke the compiler to be appended to the DW_AT_producer attribute in DWARF debugging information.
The options are concatenated with whitespace separating them from each other and from the compiler version.

IDE Equivalent
None

Alternate Options
None

gsplit-dwarf
Creates a separate object file containing DWARF debug information.
### Syntax

**Linux OS:**
- `gsplit-dwarf`

**Windows OS:**
None

### Arguments
None

### Default
- **OFF**
  No separate object file containing DWARF debug information is created.

### Description
This option creates a separate object file containing DWARF debug information. It causes debug information to be split between the generated object (.o) file and the new DWARF object (.dwo) file.

The DWARF object file is not used by the linker, so this reduces the amount of debug information the linker must process and it results in a smaller executable file.

For this option to perform correctly, you must use binutils-2.24 or later. To debug the resulting executable, you must use gdb-7.6.1 or later.

**NOTE**
If you use the split executable with a tool that does not support the split DWARF format, it will behave as though the DWARF debug information is absent.

### IDE Equivalent
None

### Alternate Options
None

### Syntax

**Linux OS:**
- `o filename`

**Windows OS:**
See option `Fo`.

### Arguments

*filename*

*Specify the name for an output file.*
Default

OFF           The compiler uses the default file name for an output file.

Description

This option specifies the name for an output file as follows:

- If -c is specified, it specifies the name of the generated object file.
- If -S is specified, it specifies the name of the generated assembly listing file.
- If -P is specified, it specifies the name of the generated preprocessor file.

Otherwise, it specifies the name of the executable file.

IDE Equivalent

None

Alternate Options

Linux: None

Windows: /Fe

See Also

Fo compiler option
Fe compiler option

pdbfile

Lets you specify the name for a program database (PDB) file created by the linker. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:

None

Windows OS:

/pdbfile[:filename]

Arguments

filename          Is the name for the PDB file. It can include a path. If you do not specify a file extension, the extension .pdb is used.

Default

OFF           No PDB file is created unless you specify option /zi. If you specify option /zi the default filename is executablename.pdb.

Description

This option lets you specify the name for a program database (PDB) file created by the linker. This option does not affect where the compiler outputs debug information.

To use this option, you must also specify option /debug:full or /zi.

If filename is not specified, the default file name used is the name of your file with an extension of .pdb.
IDE Equivalent
None

Alternate Options
None

See Also
Zi, Z7, ZI compiler option
debug compiler option
Fd compiler option

print-multi-lib
*Prints information about where system libraries should be found.*

Syntax
Linux OS:
- print-multi-lib

Windows OS:
None

Arguments
None

Default
OFF
No information is printed unless the option is specified.

Description
This option prints information about where system libraries should be found, but no compilation occurs. On Linux* systems, it is provided for compatibility with gcc.

IDE Equivalent
None

Alternate Options
None

RTC
*Enables checking for certain run-time conditions.*

Syntax
Linux OS:
None

Windows OS:
/RTCoption
Arguments

`option` Specifies the condition to check. Possible values are 1, s, u, or c.

Default

OFF No checking is performed for these run-time conditions.

Description

This option enables checking for certain run-time conditions. Using the `/RTC` option sets `__MSVC_RUNTIME_CHECKS = 1`.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/RTC1</code></td>
<td>This is the same as specifying <code>/RTCsu</code>.</td>
</tr>
<tr>
<td><code>/RTCs</code></td>
<td>Enables run-time checks of the stack frame.</td>
</tr>
<tr>
<td><code>/RTCu</code></td>
<td>Enables run-time checks for uninitialized variables.</td>
</tr>
<tr>
<td><code>/RTCc</code></td>
<td>Enables checks for converting to smaller types.</td>
</tr>
</tbody>
</table>

IDE Equivalent

Windows

Visual Studio: **Code Generation > Basic Runtime Checks / Smaller Type Check**

Linux

Eclipse: None

Alternate Options

None

S

*Causes the compiler to compile to an assembly file only and not link.*

Syntax

Linux OS:

```
-S
```

Windows OS:

```
/S
```

Arguments

None

Default

OFF Normal compilation and linking occur.
Description
This option causes the compiler to compile to an assembly file only and not link.
On Linux* systems, the assembly file name has a .s suffix. On Windows* systems, the assembly file name has an .asm suffix.

IDE Equivalent
Windows
Visual Studio: None

Linux
Eclipse: Output Files > Generate Assembler Source File

Alternate Options
Linux: None
Windows: /Fa

See Also
Fa compiler option

use-msasm
Enables the use of blocks and entire functions of assembly code within a C or C++ file.

Syntax
Linux OS:
-use-msasm

Windows OS:
None

Arguments
None

Default
OFF The compiler allows a GNU*-style inline assembly format.

Description
This option enables the use of blocks and entire functions of assembly code within a C or C++ file. It allows a Microsoft* MASM-style inline assembly block not a GNU*-style inline assembly block.

IDE Equivalent
None

Alternate Options
-fasm-blocks
Y-
*Tells the compiler to ignore all other precompiled header files.*

**Syntax**

**Linux OS:** None

**Windows OS:**

/Y-

**Arguments**

None

**Default**

OFF The compiler recognizes precompiled header files when certain compiler options are specified.

**Description**

This option tells the compiler to ignore all other precompiled header files.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

Yc compiler option

Yu compiler option

**Yc**
*Tells the compiler to create a precompiled header file.*

**Syntax**

**Linux OS:** None

**Windows OS:**

/Yc[filename]

**Arguments**

filename Is the name of a C/C++ header file, which is included in the source file using an #include preprocessor directive.

**Default**

OFF The compiler does not create or use precompiled headers unless you tell it to do so.
Description
This option tells the compiler to create a precompiled header (PCH) file. It is supported only for single source file compilations.

When filename is specified, the compiler creates a precompiled header file from the headers in the C/C++ program up to and including the C/C++ header specified.

If you do not specify filename, the compiler compiles all code up to the end of the source file, or to the point in the source file where a hdrstop occurs. The default name for the resulting file is the name of the source file with extension .pch.

This option cannot be used in the same compilation as the /Yu option.

IDE Equivalent
Windows
Visual Studio: Precompiled Headers > Precompiled Header File

Linux
Eclipse: None

Alternate Options
None

Example
If option /Fp is used, it names the PCH file. For example, consider the following command lines:

<table>
<thead>
<tr>
<th>Command Line</th>
<th>C++ Specific</th>
<th>DPC++ Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>icx /c /Ycheader.h /Fpprecomp foo.cpp</td>
<td>icx /c /Ycheader.h /Fpprecomp foo.cpp</td>
<td></td>
</tr>
<tr>
<td>icx /c /Yc /Fpprecomp foo.cpp</td>
<td>dpcpp-cl /c /Ycheader.h /Fpprecomp foo.cpp</td>
<td></td>
</tr>
<tr>
<td>dpcpp-cl /c /Yc /Fpprecomp foo.cpp</td>
<td>dpcpp-cl /c /Yc /Fpprecomp foo.cpp</td>
<td></td>
</tr>
</tbody>
</table>

In both cases, the name of the PCH file is "precomp.pch".

If the header file name is specified, the file name is based on the header file name. For example:

<table>
<thead>
<tr>
<th>Command Line</th>
<th>C++ Specific</th>
<th>DPC++ Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>icx /c /Ycheader.h foo.cpp</td>
<td>icx /c /Ycheader.h foo.cpp</td>
<td></td>
</tr>
<tr>
<td>dpcpp-cl /c /Ycheader.h foo.cpp</td>
<td>dpcpp-cl /c /Ycheader.h foo.cpp</td>
<td></td>
</tr>
</tbody>
</table>

In this case, the name of the PCH file is "header.pch".

If no header file name is specified, the file name is based on the source file name. For example:

<table>
<thead>
<tr>
<th>Command Line</th>
<th>C++ Specific</th>
<th>DPC++ Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>icx /c /Yc foo.cpp</td>
<td>icx /c /Yc foo.cpp</td>
<td></td>
</tr>
<tr>
<td>dpcpp-cl /c /Yc foo.cpp</td>
<td>dpcpp-cl /c /Yc foo.cpp</td>
<td></td>
</tr>
</tbody>
</table>

In this case, the name of the PCH file is "foo.pch".

See Also
Yu compiler option
Fp compiler option

Yu
Tell the compiler to use a precompiled header file.
Syntax

Linux OS:
None

Windows OS:
/Yu[filename]

Arguments

filename

Is the name of a C/C++ header file, which is included in the source file using an #include preprocessor directive.

Default

OFF
The compiler does not use precompiled header files unless it is told to do so.

Description

This option tells the compiler to use a precompiled header (PCH) file.

It is supported for multiple source files when all source files use the same .pch file.

The compiler treats all code occurring before the header file as precompiled. It skips to just beyond the #include directive associated with the header file, uses the code contained in the PCH file, and then compiles all code after filename.

If you do not specify filename, the compiler will use a PCH with a name based on the source file name. If you specify option /Fp, it will use the PCH specified by that option.

When this option is specified, the compiler ignores all text, including declarations preceding the #include statement of the specified file.

This option cannot be used in the same compilation as the /Yc option.

IDE Equivalent

Windows
Visual Studio: Precompiled Headers > Precompiled Header

Linux
Eclipse: None

Alternate Options

None

Example

Consider the following command line:

icx /c /Yuheader.h bar.cpp  ! specific to C++
dpcpp-cl /c /Yuheader.h bar.cpp  ! specific to DPC++

In this case, the name of the PCH file used is "header.pch".
In the following command line, no filename is specified:

```plaintext
icx /Yu bar.cpp   ! specific to C++
dpcpp-cl /Yu bar.cpp   ! specific to DPC++
```

In this case, the name of the PCH file used is "bar.pch".

In the following command line, no filename is specified, but option `/Fp` is specified:

```plaintext
icx /Yu /Fpprecomp bar.cpp   ! specific to C++
dpcpp-cl /Yu /Fpprecomp bar.cpp   ! specific to DPC++
```

In this case, the name of the PCH file used is "precomp.pch".

**See Also**

Yc compiler option

Zi, Z7, ZI

*Tells the compiler to generate full debugging information in either an object (.obj) file or a project database (PDB) file.*

**Syntax**

**Linux OS:**

See option g.

**Windows OS:**

/Zi
/Z7
/ZI

**Arguments**

None

**Default**

OFF No debugging information is produced.

**Description**

Option `/Z7` tells the compiler to generate symbolic debugging information in the object (.obj) file for use with the debugger. No .pdb file is produced by the compiler.

Option `/ZI` is a synonym for option `/Zi`.

The `/Zi` option tells the compiler to generate symbolic debugging information in a program database (PDB) file for use with the debugger. Type information is placed in the .pdb file, and not in the .obj file, resulting in smaller object files in comparison to option `/Z7`.

When option `/Zi` is specified, two PDB files are created:

- The compiler creates the program database project.pdb. If you compile a file without a project, the compiler creates a database named vcx0.pdb, where `x` represents the major version of Visual C++, for example vc140.pdb.

  This file stores all debugging information for the individual object files and resides in the same directory as the project makefile. If you want to change this name, use option `/Fd`.  

```
• The linker creates the program database `executablename.pdb`. This file stores all debug information for the .exe file and resides in the debug subdirectory. It contains full debug information, including function prototypes, not just the type information found in `vcx0.pdb`.

Both PDB files allow incremental updates. The linker also embeds the path to the .pdb file in the .exe or .dll file that it creates.

The compiler does not support the generation of debugging information in assemblable files. If you specify these options, the resulting object file will contain debugging information, but the assemblable file will not. These options turn off option `/O2` and make option `/Od` the default unless option `/O2` (or higher) is explicitly specified in the same command line.

For more information about the `/Z7`, `/Zi`, and `/ZI` options, see the Microsoft documentation.

**IDE Equivalent**

**Visual Studio**

Visual Studio: General > Generate Debug Information

**Eclipse**

Eclipse: None

**Alternate Options**

Linux: `-g`

Windows: None

**See Also**

`Fd` compiler option

`g` compiler option

`debug (Windows*)` compiler option

**Preprocessor Options**

This section contains descriptions for compiler options that pertain to preprocessing.

**B**

Specifies a directory that can be used to find include files, libraries, and executables.

**Syntax**

**Linux OS:**

`-Bdir`

**Windows OS:**

None

**Arguments**

`dir` Is the directory to be used. If necessary, the compiler adds a directory separator character at the end of `dir`.  

Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference
Default

OFF The compiler looks for files in the directories specified in your PATH environment variable.

Description

This option specifies a directory that can be used to find include files, libraries, and executables.

The compiler uses dir as a prefix.

For include files, the dir is converted to -I/dir/include. This command is added to the front of the includes passed to the preprocessor.

For libraries, the dir is converted to -L/dir. This command is added to the front of the standard -L inclusions before system libraries are added.

For executables, if dir contains the name of a tool, such as ld or as, the compiler will use it instead of those found in the default directories.

The compiler looks for include files in dir/include while library files are looked for in dir.

On Linux* systems, another way to get the behavior of this option is to use the environment variable GCC_EXEC_PREFIX.

IDE Equivalent

None

Alternate Options

None

C

Places comments in preprocessed source output.

Syntax

Linux OS:

-C

Windows OS:

/C

Arguments

None

Default

OFF No comments are placed in preprocessed source output.

Description

This option places (or preserves) comments in preprocessed source output. Comments following preprocessing directives, however, are not preserved.

IDE Equivalent

Windows

Visual Studio: Preprocessor > Keep Comments
Linux
Eclipse: None

Alternate Options
None

Example
The following commands cause the compiler to preserve comments in the prog1.i preprocessed file.

On Windows* systems:

- icx /C /P prog1.cpp prog2.cpp     ! specific to C++
- dpcpp-cl /C /P prog1.cpp prog2.cpp   ! specific to DPC++

On Linux* systems:

- icpx -C -P prog1.cpp prog2.cpp    ! specific to C++
- dpcpp -C -P prog1.cpp prog2.cpp   ! specific to DPC++

D
Defines a macro name that can be associated with an optional value.

Syntax

Linux OS:
-Dname[=value]

Windows OS:
/Dname[=value]

Arguments

name Is the name of the macro.
value Is an optional integer or an optional character string delimited by double quotes; for example, Dname=string.

Default

OFF Only default symbols or macros are defined.

Description

Defines a macro name that can be associated with an optional value. This option is equivalent to a #define preprocessor directive.

If a value is not specified, name is defined as "1".

IDE Equivalent

Windows

Visual Studio: Preprocessor > Preprocessor Definitions
Linux

Eclipse: **Preprocessor > Preprocessor Definitions**

**Alternate Options**

None

**Example**

To define a macro called SIZE with the value 100, enter the following command:

On Windows* systems:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icx /DSIZE=100 prog1.cpp</td>
<td>specific to C++</td>
</tr>
<tr>
<td>dpcpp-cl /DSIZE=100 prog1.cpp</td>
<td>specific to DPC++</td>
</tr>
</tbody>
</table>

On Linux* systems:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icpx -DSIZE=100 prog1.cpp</td>
<td>specific to C++</td>
</tr>
<tr>
<td>dpcpp -DSIZE=100 prog1.cpp</td>
<td>specific to DPC++</td>
</tr>
</tbody>
</table>

If you define a macro, but do not assign a value, the compiler defaults to 1 for the value of the macro.

**See Also**

Additional Predefined Macros

**dD, QdD**

Same as option `-dM`, but outputs `#define` directives in preprocessed source.

**Syntax**

Linux OS:

- `dD`

Windows OS:

- `/QdD`

**Arguments**

None

**Default**

OFF The compiler does not output `#define` directives.

**Description**

Same as `-dM`, but outputs `#define` directives in preprocessed source. To use this option, you must also specify the E option.

**IDE Equivalent**

None

**Alternate Options**

None
**dM, QdM**  
*Tells the compiler to output macro definitions in effect after preprocessing.*

**Syntax**

**Linux OS:**

- `dM`

**Windows OS:**

/ `QdM`

**Arguments**

None

**Default**

OFF  
The compiler does not output macro definitions after preprocessing.

**Description**

This option tells the compiler to output macro definitions in effect after preprocessing. To use this option, you must also specify option `E`.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

E  compiler option

**E**  
*Causes the preprocessor to send output to stdout.*

**Syntax**

**Linux OS:**

- `E`

**Windows OS:**

/ `E`

**Arguments**

None

**Default**

OFF  
Preprocessed source files are output to the compiler.
Description
This option causes the preprocessor to send output to stdout. Compilation stops when the files have been preprocessed.

When you specify this option, the compiler’s preprocessor expands your source module and writes the result to stdout. The preprocessed source contains #line directives, which the compiler uses to determine the source file and line number.

IDE Equivalent
None

Alternate Options
None

Example
To preprocess two source files and write them to stdout, enter the following command:

On Windows* systems:
  icx /E prog1.cpp prog2.cpp   ! specific to C++
  dpcpp-cl /E prog1.cpp prog2.cpp   ! specific to DPC++

On Linux* systems:
  icpx -E prog1.cpp prog2.cpp   ! specific to C++
  dpcpp -E prog1.cpp prog2.cpp   ! specific to DPC++

EP
Causes the preprocessor to send output to stdout, omitting #line directives.

Syntax
Linux OS:
-EP

Windows OS:
/EP

Arguments
None

Default
OFF  Preprocessed source files are output to the compiler.

Description
This option causes the preprocessor to send output to stdout, omitting #line directives.

If you also specify option P or Linux* option P, the preprocessor will write the results (without #line directives) to a file instead of stdout.
IDE Equivalent

Windows
Visual Studio: **Preprocessor > Preprocess Suppress Line Numbers**

Linux
Eclipse: None

Alternate Options
None

Example
To preprocess to stdout omitting #line directives, enter the following command:

On Windows* systems:
```
icx /EP prog1.cpp prog2.cpp   ! specific to C++
dpcpp-cl /EP prog1.cpp prog2.cpp   ! specific to DPC++
```

On Linux* systems:
```
icpx -EP prog1.cpp prog2.cpp   ! specific to C++
dpccpp -EP prog1.cpp prog2.cpp   ! specific to DPC++
```

**FI**
*Tells the preprocessor to include a specified file name as the header file.*

Syntax

Linux OS:
None

Windows OS:
```
/IFI filename
```

Arguments

*filename*  Is the file name to be included as the header file.

Default

**OFF**  The compiler uses default header files.

Description

This option tells the preprocessor to include a specified file name as the header file.
The file specified with **/FI** is included in the compilation before the first line of the primary source file.

IDE Equivalent

Windows
Visual Studio: **Advanced > Forced Include File**
Linux
Eclipse: None

Alternate Options
None

H, QH
*Tells the compiler to display the include file order and continue compilation.*

Syntax
Linux OS:

- H

Windows OS:

/QH

Arguments
None

Default
OFF  Compilation occurs as usual.

Description
This option tells the compiler to display the include file order and continue compilation.

IDE Equivalent
None

Alternate Options
None

I
*Specify an additional directory to search for include files.*

Syntax
Linux OS:

-Idir

Windows OS:

/I dir

Arguments

dir  Is the additional directory for the search.

Default
OFF  The default directory is searched for include files.
Description
This option specifies an additional directory to search for include files. To specify multiple directories on the command line, repeat the include option for each directory.

IDE Equivalent

Windows
Visual Studio: General > Additional Include Directories

Linux
Eclipse: Preprocessor > Additional Include Directories

Alternate Options
None

I-
Splits the include path.

Syntax

Linux OS:
-\-I-

Windows OS:
/\-I-

Arguments
None

Default
OFF  The default directory is searched for include files.

Description
This option splits the include path. It prevents the use of the current directory as the first search directory for '#include "file"'.

If you specify directories using the I option before you specify option I-, the directories are searched only for the case of '#include "file"'; they are not searched for '#include <file>'.

If you specify directories using the I option after you specify option I-, these directories are searched for all '#include' directives.

This option has no effect on option nostdinc++, which searches the standard system directories for header files.

This option is provided for compatibility with gcc.

IDE Equivalent
None

Alternate Options
None
See Also

I compiler option
nostdinc++ compiler option

idirafter

Adds a directory to the second include file search path.

Syntax

Linux OS:
-idirafterdir

Windows OS:
None

Arguments

dir

Is the name of the directory to add.

Default

OFF

Include file search paths include certain default directories.

Description

This option adds a directory to the second include file search path (after -I).

IDE Equivalent

None

Alternate Options

None

imacros

Allows a header to be specified that is included in front of the other headers in the translation unit.

Syntax

Linux OS:
-imacros filename

Windows OS:
None

Arguments

filename

Name of header file.

Default

OFF
Description
Allows a header to be specified that is included in front of the other headers in the translation unit.

IDE Equivalent
None

Alternate Options
None

iprefix
Lets you indicate the prefix for referencing directories that contain header files.

Syntax
Linux OS:
-iprefix prefix

Windows OS:
None

Arguments
prefix
Is the prefix to use.

Default
OFF
No prefix is included.

Description
Options for indicating the prefix for referencing directories containing header files. Use prefix with option -iwithprefix as a prefix.

IDE Equivalent
None

Alternate Options
None

iquote
Adds a directory to the front of the include file search path for files included with quotes but not brackets.

Syntax
Linux OS:
-iquote dir

Windows OS:
None
Arguments

dir Is the name of the directory to add.

Default

OFF The compiler does not add a directory to the front of the include file search path.

Description

Add directory to the front of the include file search path for files included with quotes but not brackets.

IDE Equivalent

None

Alternate Options

None

isystem

Specifies a directory to add to the start of the system include path.

Syntax

Linux OS:

-ismystemdir

Windows OS:

None

Arguments

dir Is the directory to add to the system include path.

Default

OFF The default system include path is used.

Description

This option specifies a directory to add to the system include path. The compiler searches the specified directory for include files after it searches all directories specified by the -I compiler option but before it searches the standard system directories.

On Linux* systems, this option is provided for compatibility with gcc.

IDE Equivalent

None

Alternate Options

None
**iwthprefix**

Appends a directory to the prefix passed in by `-iprefix` and puts it on the include search path at the end of the include directories.

**Syntax**

**Linux OS:**

```
-iwthprefixdir
```

**Windows OS:**

None

**Arguments**

*dir*  
Is the include directory.

**Default**

OFF

**Description**

This option appends a directory to the prefix passed in by `-iprefix` and puts it on the include search path at the end of the include directories.

**IDE Equivalent**

None

**Alternate Options**

None

**iwthprefixbefore**

Similar to `-iwthprefix` except the include directory is placed in the same place as `-I` command-line include directories.

**Syntax**

**Linux OS:**

```
- iwthprefixbefore dir
```

**Windows OS:**

None

**Arguments**

*dir*  
Is the include directory.

**Default**

OFF
**Description**

Similar to `-iwithprefix` except the include directory is placed in the same place as `-I` command-line include directories.

**IDE Equivalent**

None

**Alternate Options**

None

**Kc++, TP**

*Tells the compiler to process all source or unrecognized file types as C++ source files. This is a deprecated option. The replacement option for Kc++ is `x c++`; the replacement option for /TP is `/Tp<file>`.*

This content is specific to C++; it does not apply to DPC++.

**Syntax**

**Linux OS:**

`-Kc++`

**Windows OS:**

`/TP`

**Arguments**

None

**Default**

OFF

The compiler uses default rules for determining whether a file is a C++ source file.

**Description**

This option tells the compiler to process all source or unrecognized file types as C++ source files.

**IDE Equivalent**

**Windows**

Visual Studio: Advanced > Compile As

**Linux**

Eclipse: None

**Alternate Options**

Linux: `-x c++`

Windows: `/Tp`

**M, QM**

*Tells the compiler to generate makefile dependency lines for each source file.*
Syntax
Linux OS:
-M

Windows OS:
/QM

Arguments
None

Default
OFF The compiler does not generate makefile dependency lines for each source file.

Description
This option tells the compiler to generate makefile dependency lines for each source file, based on the #include lines found in the source file.

IDE Equivalent
None

Alternate Options
None

MD, QMD
Preprocess and compile, generating output file containing dependency information ending with extension .d.

Syntax
Linux OS:
-MD

Windows OS:
/QMD

Arguments
None

Default
OFF The compiler does not generate dependency information.

Description
Preprocess and compile, generating output file containing dependency information ending with extension .d.

IDE Equivalent
None
**Alternate Options**

None

**MF, QMF**  
*Tells the compiler to generate makefile dependency information in a file.*

**Syntax**

**Linux OS:**  
- `MF filename`

**Windows OS:**  
/`QMF filename`

**Arguments**

`filename`  
Is the name of the file where the makefile dependency information should be placed.

**Default**

OFF  
The compiler does not generate makefile dependency information in files.

**Description**

This option tells the compiler to generate makefile dependency information in a file. To use this option, you must also specify `/QM` or `/QMM`.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

QM  
compiler option

QMM  
compiler option

**MG, QMG**

*Tells the compiler to generate makefile dependency lines for each source file.*

**Syntax**

**Linux OS:**  
- `MG`

**Windows OS:**  
/`QMG`

**Arguments**

None
Default

OFF The compiler does not generate makefile dependency information in files.

Description

This option tells the compiler to generate makefile dependency lines for each source file. It is similar to /QM, but it treats missing header files as generated files.

IDE Equivalent

None

Alternate Options

None

See Also

/QM compiler option

/MM, /QMM

Tells the compiler to generate makefile dependency lines for each source file.

Syntax

Linux OS:

-illum

Windows OS:

/QMM

Arguments

None

Default

OFF The compiler does not generate makefile dependency information in files.

Description

This option tells the compiler to generate makefile dependency lines for each source file. It is similar to /QM, but it does not include system header files.

IDE Equivalent

None

Alternate Options

None

See Also

/QM compiler option
**MMD, QMMD**
*Tells the compiler to generate an output file containing dependency information.*

**Syntax**
- **Linux OS:**
  - `-MMD`
- **Windows OS:**
  - `/QMMD`

**Arguments**
None

**Default**
OFF  
The compiler does not generate an output file containing dependency information.

**Description**
This option tells the compiler to preprocess and compile a file, then generate an output file (with extension `.d`) containing dependency information.

It is similar to `/QMD`, but it does not include system header files.

**IDE Equivalent**
None

**Alternate Options**
None

**MP**
*Tells the compiler to add a phony target for each dependency.*

**Syntax**
- **Linux OS:**
  - `-MP`
- **Windows OS:**
  - None (see below)

**Arguments**
None

**Default**
OFF  
The compiler does not generate dependency information unless it is told to do so.

**Description**
This option tells the compiler to add a phony target for each dependency.
This content is specific to C++; it does not apply to DPC++. Note that this option is not related to Windows* option /MP.

**IDE Equivalent**
None

**Alternate Options**
None

**MQ**
*Changes the default target rule for dependency generation.*

**Syntax**

**Linux OS:**
- `MQtargertarget`

**Windows OS:**
None

**Arguments**

`target` Is the target rule to use.

**Default**

OFF The default target rule applies to dependency generation.

**Description**
This option changes the default target rule for dependency generation. It is similar to `−MT`, but quotes special Make characters.

**IDE Equivalent**
None

**Alternate Options**
None

**MT, QMT**
*Changes the default target rule for dependency generation.*

**Syntax**

**Linux OS:**
- `MQtargertarget`

**Windows OS:**
`/QMTtarget`
Arguments

target

Is the target rule to use.

Default

OFF

The default target rule applies to dependency generation.

Description

This option changes the default target rule for dependency generation.

IDE Equivalent

None

Alternate Options

None

nostdinc++

Do not search for header files in the standard directories for C++, but search the other standard directories.

Syntax

Linux OS:

-nostdinc++

Windows OS:

None

Arguments

None

Default

OFF

Description

Do not search for header files in the standard directories for C++, but search the other standard directories.

IDE Equivalent

None

Alternate Options

None

P

Tells the compiler to stop the compilation process and write the results to a file.
Syntax
Linux OS:

-\(P\)

Windows OS:

/\(P\)

Arguments
None

Default
OFF Normal compilation is performed.

Description
This option tells the compiler to stop the compilation process after C or C++ source files have been preprocessed and write the results to files named according to the compiler's default file-naming conventions.

On Linux systems, this option causes the preprocessor to expand your source module and direct the output to a .i file instead of stdout. Unlike the -E option, the output from -\(P\) on Linux does not include #line number directives. By default, the preprocessor creates the name of the output file using the prefix of the source file name with a .i extension. You can change this by using the -o option.

IDE Equivalent
Windows
Visual Studio: Preprocessor > Generate Preprocessed File

Linux
Eclipse: None

Alternate Options
Linux: -\(F\)
Windows: None

**pragma-optimization-level**
Specifies which interpretation of the optimization_level pragma should be used if no prefix is specified. This content is specific to C++; it does not apply to DPC++.

Syntax
Linux OS:

-pragma-optimization-level=interpretation

Windows OS:

None
Arguments

*interpretation*

Compiler-specific interpretation of `optimization_level pragma`. Possible values are:

- **Intel**: Specify the Intel interpretation.
- **GCC**: Specify the GCC interpretation.

Default

```
-pragma-optimization-level=Intel
```

Use the Intel interpretation of the `optimization_level pragma`.

Description

Specifies which interpretation of the `optimization_level pragma` should be used if no prefix is specified.

IDE Equivalent

None

Alternate Options

None

*u* (Windows*)

Disables all predefined macros and assertions. This content is specific to C++; it does not apply to DPC++.

Syntax

**Linux OS:**

None

**Windows OS:**

```
/u
```

Arguments

None

Default

OFF Defined preprocessor values are in effect until they are undefined.

Description

This option disables all predefined macros and assertions.

IDE Equivalent

**Windows**

Visual Studio: **Preprocessor > Undefine All Preprocessor Definitions**

**Linux**

Eclipse: None
Alternate Options
None

U
Undefines any definition currently in effect for the specified macro.

Syntax
Linux OS:
-\texttt{Uname}

Windows OS:
/\texttt{Uname}

Arguments
name Is the name of the macro to be undefined.

Default
OFF Macro definitions are in effect until they are undefined.

Description
This option undefines any definition currently in effect for the specified macro. It is equivalent to an \texttt{#undef}
preprocessing directive.

On Windows systems, use the /u option to undefine all previously defined preprocessor values.

IDE Equivalent
Windows
Visual Studio: Preprocessor > Undefine Preprocessor Definitions

Linux
Eclipse: Preprocessor > Undefine Preprocessor Definitions

Alternate Options
None

Example
To undefine a macro, enter the following command:

On Windows* systems:

\begin{verbatim}
icx /Uia64 progl.cpp   ! specific to C++
dcpp-cl /Uia64 progl.cpp   ! specific to DPC++
\end{verbatim}

On Linux* systems:

\begin{verbatim}
icpx -Uia64 progl.cpp   ! specific to C++
dcpp -Uia64 progl.cpp   ! specific to DPC++
\end{verbatim}
If you attempt to undefine an ANSI C macro, the compiler will emit an error:

```
invalid macro undefinition: <name of macro>
```

**See Also**

**undef**

Disables all predefined macros.

**Syntax**

**Linux OS:**

```
-undef
```

**Windows OS:**

None

**Arguments**

None

**Default**

OFF Defined macros are in effect until they are undefined.

**Description**

This option disables all predefined macros.

**IDE Equivalent**

None

**Alternate Options**

None

**X**

Removes standard directories from the include file search path.

**Syntax**

**Linux OS:**

```
-X
```

**Windows OS:**

```
/X
```

**Arguments**

None

**Default**

OFF Standard directories are in the include file search path.
**Description**

This option removes standard directories from the include file search path. It prevents the compiler from searching the default path specified by the INCLUDE environment variable.

On Linux* systems, specifying -X (or -noinclude) prevents the compiler from searching in /usr/include for files specified in an INCLUDE statement.

You can use this option with the I option to prevent the compiler from searching the default path for include files and direct it to use an alternate path.

**IDE Equivalent**

**Windows**
Visual Studio: *Preprocessor > Ignore Standard Include Path*

**Linux**
Eclipse: *Preprocessor > Ignore Standard Include Path*

**Alternate Options**

Linux: `-nostdinc`

Windows: None

**See Also**

I compiler option

**Component Control Options**

This section contains descriptions for compiler options that pertain to component control.

**Qinstall**

Specifies the root directory where the compiler installation was performed. This content is specific to C++; it does not apply to DPC++.

**Syntax**

**Linux OS:**

`-Qinstall` *dir*

**Windows OS:**

None

**Arguments**

*dir* Is the root directory where the installation was performed.

**Default**

OFF The default root directory for compiler installation is searched for the compiler.

**Description**

This option specifies the root directory where the compiler installation was performed. It is useful if you want to use a different compiler or if you did not use a shell script to set your environment variables.
**IDE Equivalent**
None

**Alternate Options**
None

**Qlocation**
Specifies the directory for supporting tools. This content is specific to C++; it does not apply to DPC++.

**Syntax**

**Linux OS:**
- `Qlocation, string, dir`

**Windows OS:**
/`Qlocation, string, dir`

**Arguments**

`string`  
Is the name of the tool.

`dir`  
Is the directory (path) where the tool is located.

**Default**
OFF  
The compiler looks for tools in a default area.

**Description**
This option specifies the directory for supporting tools.

`string` can be any of the following:

- c - Indicates the Intel® oneAPI DPC++/C++ Compiler.
- cpp (or fpp) - Indicates the Intel® C++ preprocessor.
- cxxinc - Indicates C++ header files.
- cinc - Indicates C header files.
- asm - Indicates the assembler.
- link - Indicates the linker.
- prof - Indicates the profiler.
- On Windows* systems, the following is also available:
  - masm - Indicates the Microsoft assembler.
- On Linux* systems, the following are also available:
  - as - Indicates the assembler.
  - gas - Indicates the GNU assembler. This setting is for Linux* only.
  - ld - Indicates the loader.
  - gld - Indicates the GNU loader. This setting is for Linux* only.
  - lib - Indicates an additional library.
  - crt - Indicates the crt%.o files linked into executables to contain the place to start execution.

On Windows systems, you can also specify a tool command name.
The following shows an example on Windows® systems:

```
/Qlocation,link,"c:\Program Files\tools\"            ! This tells the driver to use c:\Program
Files\tools\link.exe for the loader
/Qlocation,link,"c:\Program Files\tools\my_link.exe" ! This tells the driver to use c:\Program
Files\tools\my_link.exe as the loader
```

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

Qoption  compiler option

**Qoption**

*Passes options to a specified tool. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**

```
-Qoption,string,options
```

**Windows OS:**

```
/Qoption,string,options
```

**Arguments**

- **string**
  
  Is the name of the tool.

- **options**
  
  Are one or more comma-separated, valid options for the designated tool.

  Note that certain tools may require that options appear within quotation marks (" ").

**Default**

OFF  No options are passed to tools.

**Description**

This option passes options to a specified tool.

If an argument contains a space or tab character, you must enclose the entire argument in quotation marks (" "). You must separate multiple arguments with commas.

**string** can be any of the following:

- cpp - Indicates the preprocessor for the compiler.
- c - Indicates the Intel® oneAPI DPC++/C++ Compiler.
- asm - Indicates the assembler.
- link - Indicates the linker.
- prof - Indicates the profiler.
- On Windows* systems, the following is also available:
• masm - Indicates the Microsoft assembler.
• On Linux* systems, the following are also available:
  • as - Indicates the assembler.
  • gas - Indicates the GNU assembler.
  • ld - Indicates the loader.
  • gld - Indicates the GNU loader.
  • lib - Indicates an additional library.
  • crt - Indicates the crt%.o files linked into executables to contain the place to start execution.

IDE Equivalent
None

Alternate Options
None

See Also
Qlocation  compiler option

Language Options
This section contains descriptions for compiler options that pertain to language compatibility, conformity, etc.

ansi
Enables language compatibility with the gcc option -ansi.

Syntax
Linux OS:
-ansi

Windows OS:
None

Arguments
None

Default
OFF  GNU C++ is more strongly supported than ANSI C.

Description
This option enables language compatibility with the gcc option -ansi and provides the same level of ANSI standard conformance as that option.

This option sets option fmath-errno.

This content is specific to C++; it does not apply to DPC++. If you want strict ANSI conformance, use the -strict-ansi option.

IDE Equivalent
Windows
Visual Studio: None
Linux

Eclipse: Language > ANSI Conformance

Alternate Options

None

fno-gnu-keywords

_Tells the compiler to not recognize typeof as a keyword._

Syntax

Linux OS:

-`fno-gnu-keywords`

Windows OS:

None

Arguments

None

Default

OFF  _Keyword typeof is recognized._

Description

Tells the compiler to not recognize `typeof` as a keyword.

IDE Equivalent

None

Alternate Options

None

fno-operator-names

_Disables support for the operator names specified in the standard._

Syntax

Linux OS:

-`fno-operator-names`

Windows OS:

None

Arguments

None

Default

OFF
**Description**  
Disables support for the operator names specified in the standard.

**IDE Equivalent**  
None

**Alternate Options**  
None

**fno-rtti**  
*Disables support for run-time type information (RTTI).*

**Syntax**  
**Linux OS:**  
- `fno-rtti`

**Windows OS:**  
None

**Arguments**  
None

**Default**  
OFF  
Support for run-time type information (RTTI) is enabled.

**Description**  
This option disables support for run-time type information (RTTI).

**IDE Equivalent**  
**Windows**  
Visual Studio: None

**Linux**  
Eclipse: None

**Alternate Options**  
None

**fpermissive**  
*Tells the compiler to allow for non-conformant code.*

**Syntax**  
**Linux OS:**  
- `fpermissive`

**Windows OS:**  
None
Arguments
None

Default
OFF

Description
Tells the compiler to allow for non-conformant code.

IDE Equivalent
None

Alternate Options
None

`fshort-enums`
*Tells the compiler to allocate as many bytes as needed for enumerated types.*

Syntax

**Linux OS:**
- `fshort-enums`

**Windows OS:**
None

Arguments
None

Default
OFF

Description
The compiler allocates a default number of bytes for enumerated types.

IDE Equivalent

Windows
Visual Studio: None

Linux
Eclipse: **Data > Associate as Many Bytes as Needed for Enumerated Types**

Alternate Options
None

`fsyntax-only`
*Tell the compiler to check only for correct syntax.*
**Syntax**

**Linux OS:**
-fsyntax-only

**Windows OS:**
None

**Arguments**
None

**Default**
OFF Normal compilation is performed.

**Description**
This option tells the compiler to check only for correct syntax. No object file is produced.

**IDE Equivalent**
None

**Alternate Options**
Linux: None
Windows: /Zs

**funsigned-char**
*Change default char type to unsigned.*

**Syntax**

**Linux OS:**
-funsigned-char

**Windows OS:**
None

**Arguments**
None

**Default**
OFF Do not change default char type to unsigned.

**Description**
Change default char type to unsigned.

**IDE Equivalent**

**Windows**
Visual Studio: None
Linux

Eclipse: **Data > Change default char type to unsigned**

**Alternate Options**
None

**GZ**

*Initializes all local variables. This is a deprecated option. The replacement option is /RTC1. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**
None

**Windows OS:**
/J

**Arguments**
None

**Default**
OFF

The compiler does not initialize local variables.

**Description**

This option initializes all local variables to a non-zero value. To use this option, you must also specify option /Od.

**IDE Equivalent**
None

**Alternate Options**

Linux: None
Windows: /RTC1

**J**

*Sets the default character type to unsigned.*

**Syntax**

**Linux OS:**
None

**Windows OS:**
/J

**Arguments**
None
Default

OFF The default character type is signed

Description
This option sets the default character type to unsigned. This option has no effect on character values that are explicitly declared signed. This option sets _CHAR_UNSIGNED = 1.

IDE Equivalent

Windows
Visual Studio: Language > Default Char Unsigned

Linux
Eclipse: None

Alternate Options
None

std, Qstd
Tells the compiler to conform to a specific language standard.

Syntax

Linux OS:
-std=val

Windows OS:
/Qstd:val
/std:val (For Microsoft* compatibility)

Arguments

val Specifies the specific language standard to conform to.
The following values apply to Linux* -std and Windows* /Qstd:

<table>
<thead>
<tr>
<th>std</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c++2b</td>
<td>Enables support for the Working Draft for ISO C++ 2023 DIS standard.</td>
</tr>
<tr>
<td>c++20</td>
<td>Enables support for the 2020 ISO C++ DIS standard.</td>
</tr>
<tr>
<td>c++17</td>
<td>Enables support for the 2017 ISO C++ standard with amendments.</td>
</tr>
<tr>
<td>c++14</td>
<td>Enables support for the 2014 ISO C++ standard with amendments.</td>
</tr>
<tr>
<td>c++11</td>
<td>Enables support for the 2011 ISO C++ standard with amendments.</td>
</tr>
<tr>
<td>c++98 and c++03</td>
<td>Enables support for the 1998 ISO C++ standard with amendments.</td>
</tr>
</tbody>
</table>
Enables support for the Working Draft for ISO C2x standard.

c18 and c17
Enables support for the 2017 ISO C standard.
Support for c17 can also be enabled by value iso9899:2017.
Support for c18 can also be enabled by value iso9899:2018.

c11
Enables support for the 2011 ISO C standard.
Support for this standard can also be enabled by value iso9899:2011.

c99
Enables support for the 1999 ISO C standard.
Support for this standard can also be enabled by value iso9899:1999.

c90 and c89
Enables support for the 1990 ISO C standard.
Support for this standard can also be enabled by value iso9899:1990.

The following values apply only to Linux -std:

gnu++2b
Enables support for the Working Draft for ISO C++ 2023 DIS standard plus GNU extensions.

gnu++20
Enables support for the 2020 ISO C++ DIS standard plus GNU extensions.

gnu++17
Enables support for the 2017 ISO C++ standard with amendments plus GNU extensions.

gnu++14
Enables support for the 2014 ISO C++ standard with amendments plus GNU extensions.

gnu++11
Enables support for the 2011 ISO C++ standard with amendments plus GNU extensions.

gnu++98 and gnu++03
Enables support for the 1998 ISO C++ standard with amendments plus GNU extensions.

gnu2x
Enables support for the Working Draft for ISO C2x standard plus GNU extensions.

gnu18 and gnu17
Enables support for the 2017 ISO C standard plus GNU extensions.

gnu11
Enables support for the 2011 ISO C standard plus GNU extensions.

gnu99
Enables support for the 1999 ISO C standard plus GNU extensions.

gnu90 and gnu89
Enables support for the 1990 ISO C standard plus GNU extensions.

For possible values for Microsoft*-compatible Windows* /std, see the Microsoft* documentation.
Default

Default for Windows option /Qstd:
OFF

Default for Windows option /std:
c++14

Default for Linux option -std on
dpcpp:
c++17

Default for Linux option -std on icx:
c++14

Description
This option tells the compiler to conform to a specific language standard.

IDE Equivalent

Visual Studio
Visual Studio: Language > C/C++ Language Support

Eclipse
Eclipse: Language > ANSI Conformance

Alternate Options
None

strict-ansi
Tells the compiler to implement strict ANSI conformance dialect. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
-strict-ansi

Windows OS:
None

Arguments
None

Default

OFF

The compiler conforms to default standards.

Description
This option tells the compiler to implement strict ANSI conformance dialect. On Linux* systems, if you need to be compatible with gcc, use the -ansi option.
This option sets option `fmath_errno`, which tells the compiler to assume that the program tests `errno` after calls to math library functions. This restricts optimization because it causes the compiler to treat most math functions as having side effects.

**IDE Equivalent**

**Windows**
Visual Studio: None

**Linux**
Eclipse: **Language > ANSI Conformance**

**Alternate Options**
None

**vd**

*Enables or suppresses hidden vtordisp members in C++ objects.*

**Syntax**

**Linux OS:**
None

**Windows OS:**
`/vd n`

**Arguments**

<table>
<thead>
<tr>
<th>n</th>
<th>Possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Suppresses the creation of the hidden vtordisp members in C++ objects.</td>
</tr>
<tr>
<td>1</td>
<td>Enables the creation of hidden vtordisp members in C++ objects when they are necessary.</td>
</tr>
</tbody>
</table>
| 2  | Enables the hidden vtordisp members for all virtual base classes with virtual functions. This setting is recommended in the following cases:  
  - When the only virtual function in your virtual base class is a destructor  
  - When you want to ensure correct performance of the dynamic_cast operator on a partially-constructed object |

**Default**

`/vd1`  
The compiler enables the creation of hidden vtordisp members in C++ objects when they are necessary.

**Description**

This option enables or suppresses hidden vtordisp members in C++ objects.

This is a compatibility option for the Microsoft Visual C++* option `/vdn`. For full details about this compiler option, see the Microsoft* documentation.
**IDE Equivalent**
None

**Alternate Options**
None

**vmg**
*Selects the general representation that the compiler uses for pointers to members.*

**Syntax**

**Linux OS:**
None

**Windows OS:**
/vmg

**Arguments**
None

**Default**
OFF  The compiler uses default rules to represent pointers to members.

**Description**
This option selects the general representation that the compiler uses for pointers to members. Use this option if you declare a pointer to a member before you define the corresponding class.

**IDE Equivalent**
None

**Alternate Options**
None

**x (type option)**
*All source files found subsequent to -x type will be recognized as a particular type.*

**Syntax**

**Linux OS:**
-x type

**Windows OS:**
None

**Arguments**

*type*  is the type of source file. Possible values are:

* c++  C++ source file
c++-header | C++ header file
---|---
c++-cpp-output | C++ pre-processed file
c | C source file
c-header | C header file
cpp-output | C pre-processed file
assembler | Assembly file
assembler-with-cpp | Assembly file that needs to be preprocessed
none | Disable recognition, and revert to file extension

**Default**
none | Disable recognition and revert to file extension.

**Description**
All source files found subsequent to `-x type` will be recognized as a particular type.

**IDE Equivalent**
None

**Alternate Options**
None

**Example**
Suppose you want to compile the following C and C++ source files whose extensions are not recognized by the compiler:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>file1.c99</td>
<td>C</td>
</tr>
<tr>
<td>file2.cplusplus</td>
<td>C++</td>
</tr>
</tbody>
</table>

We will also include these files whose extensions are recognized:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>file3.c</td>
<td>C</td>
</tr>
<tr>
<td>file4.cpp</td>
<td>C++</td>
</tr>
</tbody>
</table>

The command-line invocation using the `-x` option follows:

```bash
icpx -x c file1.c99 -x c++ file2.cplusplus -x none file3.c file4.cpp   ! specific to C++
dpcpp -x c file1.c99 -x c++ file2.cplusplus -x none file3.c file4.cpp   ! specific to DPC++
```

**Zc**
Lets you specify ANSI C standard conformance for certain language features.
Syntax

Linux OS:
None

Windows OS:
/Zc:arg1[,arg2]

Arguments

arg

Is the language feature for which you want standard conformance.
The settings are compatible with Microsoft* settings for option /Zc.
For a list of supported settings, see the table in the Description section of this topic.

Default

varies

See the table in the Description section of this topic.

Description

This option lets you specify ANSI C standard conformance for certain language features.
If you do not want the default behavior for one or more of the settings, you must specify the negative form
of the setting. For example, if you do not want the threadSafeInit or sizedDealloc default behavior, you
should specify /Zc:threadSafeInit-,sizedDealloc-.

The following table shows the supported Microsoft settings for option /Zc.

<table>
<thead>
<tr>
<th>/Zc setting name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alignedNew[-]</td>
<td>Enables C++17 aligned allocation functions (default for C++17). Disabled by /Zc:alignedNew-.</td>
</tr>
<tr>
<td>char8_t[-]</td>
<td>Enables char8_t from C++2a. Disabled by /Zc:char8_t- (default).</td>
</tr>
<tr>
<td>dllexportInlines[-]</td>
<td>Enables dllexport/dllimport inline member functions of dllexport/import classes (default). Disabled by /Zc:dllexportInlines-.</td>
</tr>
<tr>
<td>sizedDealloc[-]</td>
<td>Enables C++14 sized global deallocation functions (default). Disabled by /Zc:sizedDealloc-.</td>
</tr>
<tr>
<td>strictStrings[-]</td>
<td>Enforces const qualification for string literals. Disabled by /Zc:strictStrings- (default).</td>
</tr>
<tr>
<td>threadSafeInit[-]</td>
<td>Enables thread-safe initialization of local statics (default). Disabled by /Zc:threadSafeInit-.</td>
</tr>
<tr>
<td>trigraphs[-]</td>
<td>Enables trigraph character sequences. Disabled by /Zc:trigraphs- (default).</td>
</tr>
<tr>
<td>twoPhase[-]</td>
<td>Enables two-phase name lookup in templates. Disabled by /Zc:twoPhase- (default).</td>
</tr>
</tbody>
</table>
IDE Equivalent

Windows

Visual Studio: Language > Treat wchar_t as Built-in Type / Force Conformance In For Loop Scope
Language > Enforce type conversion rules (rvalueCast)

Linux

Eclipse: None

Alternate Options

None

Zg

Tells the compiler to generate function prototypes.
This is a deprecated option. There is no replacement option. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
None

Windows OS:
/Zg

Arguments

None

Default

OFF The compiler does not create function prototypes.

Description

This option tells the compiler to generate function prototypes.

IDE Equivalent

None

Alternate Options

None

Zp

Specifies alignment for structures on byte boundaries.

Syntax

Linux OS:
-Zp[n]

Windows OS:
/Zp[n]
Arguments

\[ n \]  
Is the byte size boundary. Possible values are 1, 2, 4, 8, or 16.

Default

Zp16  
Structures are aligned on either size boundary 16 or the boundary that will naturally align them.

Description

This option specifies alignment for structures on byte boundaries.

If you do not specify \( n \), you get Zp16.

IDE Equivalent

Windows

Visual Studio: Code Generation > Struct Member Alignment

Linux

Eclipse: Data > Structure Member Alignment

Alternate Options

None

Zs

_Tells the compiler to check only for correct syntax._

Syntax

Linux OS:  
None

Windows OS:  
/Zs

Arguments

None

Default

OFF  
Normal compilation is performed.

Description

This option tells the compiler to check only for correct syntax.

IDE Equivalent

None

Alternate Options

Linux: -syntax, -fsyntax-only

Windows: None
Data Options
This section contains descriptions for compiler options that pertain to the treatment of data.

align
Determines whether variables and arrays are naturally aligned. This content is specific to C++; it does not apply to DPC++.

Architecture Restrictions
Only available on IA-32 architecture

Syntax
Linux OS:
-align
-noalign

Windows OS:
None

Arguments
None

Default
-noalign Variables and arrays are aligned according to the gcc model, which means they are aligned to 4-byte boundaries.

Description
This option determines whether variables and arrays are naturally aligned. Option -align forces the following natural alignment:

<table>
<thead>
<tr>
<th>Type</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>8 bytes</td>
</tr>
<tr>
<td>long long</td>
<td>8 bytes</td>
</tr>
<tr>
<td>long double</td>
<td>16 bytes</td>
</tr>
</tbody>
</table>

If you are not interacting with system libraries or other libraries that are compiled without -align, this option can improve performance by reducing misaligned accesses.

This option can also be specified as -m[no-]align-double. The options are equivalent.

Caution
If you are interacting with compatible libraries, this option can improve performance by reducing misaligned accesses. However, if you are interacting with noncompatible libraries or libraries that are compiled without option -align, your application may not perform as expected.
**IDE Equivalent**
None

**Alternate Options**
None

**fcommon**
*Determines whether the compiler treats common symbols as global definitions.*

**Syntax**

**Linux OS:**
- `fcommon`
- `fno-common`

**Windows OS:**
None

**Arguments**
None

**Default**
- `fcommon` The compiler does not treat common symbols as global definitions.

**Description**
This option determines whether the compiler treats common symbols as global definitions and to allocate memory for each symbol at compile time.

Option `-fno-common` tells the compiler to treat common symbols as global definitions. When using this option, you can only have a common variable declared in one module; otherwise, a link time error will occur for multiple defined symbols.

Normally, a file-scope declaration with no initializer and without the `extern` or `static` keyword "int i;" is represented as a common symbol. Such a symbol is treated as an external reference. However, if no other compilation unit has a global definition for the name, the linker allocates memory for it.

**IDE Equivalent**

**Windows**
Visual Studio: None

**Linux**
Eclipse: **Data > Allow gprel Addressing of Common Data Variables**

**Alternate Options**
None

**fkeep-static-consts, Qkeep-static-consts**
*Tells the compiler to preserve allocation of variables that are not referenced in the source.*
Syntax

Linux OS:
- fkeep-static-consts
- fno-keep-static-consts

Windows OS:
/Qkeep-static-consts (C++ only)
/Qkeep-static-consts- (C++ only)

Arguments
None

Default
- fno-keep-static-consts

C++: or /Qkeep-static-consts-

If a variable is never referenced in a routine, the variable is discarded unless optimizations are disabled by option -O0 (Linux*) or /Od (Windows*).

Description
This option tells the compiler to preserve allocation of variables that are not referenced in the source.
The negated form can be useful when optimizations are enabled to reduce the memory usage of static data.

IDE Equivalent
None

Alternate Options
None

fmath-errno
Tells the compiler that errno can be reliably tested after calls to standard math library functions.

Syntax

Linux OS:
- fmath-errno
- fno-math-errno

Windows OS:
None

Arguments
None

Default
- fno-math-errno

The compiler assumes that the program does not test errno after calls to standard math library functions.
Description
This option tells the compiler to assume that the program tests `errno` after calls to math library functions. This restricts optimization because it causes the compiler to treat most math functions as having side effects. Option `-fno-math-errno` tells the compiler to assume that the program does not test `errno` after calls to math library functions. This frequently allows the compiler to generate faster code. Floating-point code that relies on IEEE exceptions instead of `errno` to detect errors can safely use this option to improve performance.

IDE Equivalent
None

Alternate Options
None

`fpack-struct`
Specifies that structure members should be packed together.

Syntax
Linux OS:
- `-fpack-struct`

Windows OS:
None

Arguments
None

Default
OFF

Description
Specifies that structure members should be packed together.

NOTE
Using this option may result in code that is not usable with standard (system) c and C++ libraries.

IDE Equivalent
None

Alternate Options
Linux: `-Zp1`
Windows: None
**fpascal-strings**
*Tells the compiler to allow for Pascal-style string literals. This content is specific to C++; it does not apply to DPC++.*

**Architecture Restrictions**
Only available on IA-32 architecture

**Syntax**

**Linux OS:**
- `fpascal-strings`

**Windows OS:**
None

**Arguments**
None

**Default**
OFF  
The compiler does not allow for Pascal-style string literals.

**Description**
Tells the compiler to allow for Pascal-style string literals.

**IDE Equivalent**

**Windows**
Visual Studio: None

**Linux**
Eclipse: None

**Alternate Options**
None

**fpic**
*Determines whether the compiler generates position-independent code.*

**Syntax**

**Linux OS:**
- `fpic`
- `fno-pic`

**Windows OS:**
None
Arguments
None

Default
-fno-pic  The compiler does not generate position-independent code.

Description
This option determines whether the compiler generates position-independent code.
Option -fpic specifies full symbol preemption. Global symbol definitions as well as global symbol references get default (that is, preemptable) visibility unless explicitly specified otherwise.
Option -fpic must be used when building shared objects.
This option can also be specified as -fPIC.

IDE Equivalent
Windows
Visual Studio: None

Linux
Eclipse: Code Generation > Generate Position Independent Code

Alternate Options
None

fpie
 Tells the compiler to generate position-independent code. The generated code can only be linked into executables.

Syntax
Linux OS:
-fpie

Windows OS:
None

Arguments
None

Default
OFF  The compiler does not generate position-independent code for an executable-only object.

Description
This option tells the compiler to generate position-independent code. It is similar to -fpic, but code generated by -fpie can only be linked into an executable.
Because the object is linked into an executable, this option causes better optimization of some symbol references.

To ensure that run-time libraries are set up properly for the executable, you should also specify option -pie to the compiler driver on the link command line.

Option -fpie can also be specified as -fPIE.

**IDE Equivalent**
None

**Alternate Options**
None

**See Also**
fpic compiler option
pie compiler option

**freg-struct-return**
*Tells the compiler to return struct and union values in registers when possible. This content is specific to C++; it does not apply to DPC++.*

**Architecture Restrictions**
Only available on IA-32 architecture

**Syntax**

**Linux OS:**
-freg-struct-return

**Windows OS:**
None

**Arguments**
None

**Default**
OFF

**Description**
This option tells the compiler to return struct and union values in registers when possible.

**IDE Equivalent**
None

**Alternate Options**
None

**fstack-protector**
*Enables or disables stack overflow security checks for certain (or all) routines.*
Syntax

Linux OS:
-fstack-protector[-keyword]
-fno-stack-protector[-keyword]

Windows OS:
None

Arguments

keyword Possible values are:

strong When option -fstack-protector-strong is specified, it enables stack overflow security checks for routines with any type of buffer.

all When option -fstack-protector-all is specified, it enables stack overflow security checks for every routine.

If no -keyword is specified, option -fstack-protector enables stack overflow security checks for routines with a string buffer.

Default

-fno-stack-protector, -fno-stack-protector-strong No stack overflow security checks are enabled for the relevant routines.
-fno-stack-protector-all No stack overflow security checks are enabled for any routines.

Description

This option enables or disables stack overflow security checks for certain (or all) routines. A stack overflow occurs when a program stores more data in a variable on the execution stack than is allocated to the variable. Writing past the end of a string buffer or using an index for an array that is larger than the array bound could cause a stack overflow and security violations.

The -fstack-protector options are provided for compatibility with gcc. They use the gcc/glibc implementation when possible. If the gcc/glibc implementation is not available, they use the Intel implementation.

This content is specific to C++; it does not apply to DPC++.
For an Intel-specific version of this feature, see option -fstack-security-check.

IDE Equivalent
None

Alternate Options
None

See Also
fstack-security-check compiler option
GS compiler option
**fstack-security-check**

Determines whether the compiler generates code that detects some buffer overruns. This content is specific to C++; it does not apply to DPC++.

**Syntax**

**Linux OS:**

- `-fstack-security-check`
- `-fno-stack-security-check`

**Windows OS:**

None

**Arguments**

None

**Default**

- `-fno-stack-security-check`
  The compiler does not detect buffer overruns.

**Description**

This option determines whether the compiler generates code that detects some buffer overruns that overwrite the return address. This is a common technique for exploiting code that does not enforce buffer size restrictions.

This option always uses an Intel implementation.

For a gcc-compliant version of this feature, see option `fstack-protector`.

**IDE Equivalent**

None

**Alternate Options**

Linux: None

Windows: `/GS`

**See Also**

`fstack-protector` compiler option

`GS` compiler option

**fvisibility**

Specifies the default visibility for global symbols or the visibility for symbols in declarations, functions, or variables. This content is specific to C++; it does not apply to DPC++.

**Syntax**

**Linux OS:**

- `-fvisibility=arg`
- `-fvisibility-global-new-delete-hidden`
-fvisibility-inlines-hidden
-f[no]visibility-inlines-hidden-static-local-var
-fvisibility-ms-compat

Windows OS:
None

Arguments

arg Specifies the visibility setting. Possible values are:

default Sets visibility to default. The symbol is visible outside this shared object.

  This means that other components can reference the symbols, and the symbol definitions can be overridden (preempted) by a definition of the same name in another component.

hidden Sets visibility to hidden. The symbol is not visible outside this shared object.

  This means that other components cannot directly reference the symbol.

internal This is the same as specifying hidden.

protected Sets visibility to protected. The symbol is seen by the dynamic linker but always dynamically resolves to an object within this shared object.

  This means that other components can reference the symbol, but it cannot be overridden by a definition of the same name in another component.

  This value is not supported on all targets.

Default

-fvisibility=default The compiler sets visibility of symbols to default.

Description

This option specifies the default visibility for global symbols (syntax -fvisibility=arg) or the visibility for symbols in declarations, functions, or variables.

The following table shows supported -fvisibility options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fvisibility=arg</td>
<td>Sets visibility of symbols for all global declarations.</td>
</tr>
<tr>
<td></td>
<td>As specified above in Arguments, arg can be one of the following: hidden</td>
</tr>
<tr>
<td></td>
<td>internal default protected.</td>
</tr>
<tr>
<td>-fvisibility-inlines-hidden</td>
<td>Sets hidden visibility by default for inline C++ member functions.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>-fvisibility-inlines-hidden-static-local-var</td>
<td>When -fvisibility-inlines-hidden is enabled, static variables in inline C++ member functions will also be given hidden visibility by default. To disable option -fvisibility-inlines-hidden-static-local-var, specify option -fno-visibility-inlines-hidden-static-local-var.</td>
</tr>
<tr>
<td>-fvisibility-ms-compat</td>
<td>Sets default visibility for global types and sets hidden visibility for global functions and variables.</td>
</tr>
</tbody>
</table>

If an -fvisibility option is specified more than once on the command line, the last specification takes precedence over any others.

The following shows the precedence of the visibility settings (from greatest to least visibility):

- default
- protected
- hidden

**IDE Equivalent**

None

**Alternate Options**

None

**fzero-initialized-in-bss, Qzero-initialized-in-bss**

Determines whether the compiler places in the DATA section any variables explicitly initialized with zeros.

**Syntax**

**Linux OS:**

- -fzero-initialized-in-bss
- -fno-zero-initialized-in-bss

**Windows OS:**

- /Qzero-initialized-in-bss
- /Qzero-initialized-in-bss-

**Arguments**

None

**Default**

- -fno-zero-initialized-in-bss or /Qzero-initialized-in-bss-

Variables explicitly initialized with zeros are placed in the BSS section. This can save space in the resulting code.
**Description**

This option determines whether the compiler places in the DATA section any variables explicitly initialized with zeros.

If option `-fno-zero-initialized-in-bss (Linux*)` or `/Qzero-initialized-in-bss- (Windows*)` is specified, the compiler places in the DATA section any variables that are initialized to zero.

**IDE Equivalent**

**Windows**

Visual Studio: None

**Linux**

Eclipse: **Data > Disable Placement of Zero-Initialized Variables in .bss - place in .data instead**

**Alternate Options**

None

**GA**

*Enables faster access to certain thread-local storage (TLS) variables.*

**Syntax**

**Linux OS:**

None

**Windows OS:**

/GA

**Arguments**

None

**Default**

OFF Default access to TLS variables is in effect.

**Description**

This option enables faster access to certain thread-local storage (TLS) variables. When you compile your main executable (.EXE) program with this option, it allows faster access to TLS variables declared with the `__declspec(thread)` specification.

Note that if you use this option to compile .DLLs, you may get program errors.

**IDE Equivalent**

**Windows**

Visual Studio: **Optimization > Optimize for Windows Applications**

**Linux**

Eclipse: None
Alternate Options
None

Gs
*Lets you control the threshold at which the stack checking routine is called or not called.*

**Syntax**

**Linux OS:**
None

**Windows OS:**
/Gs\[n\]

**Arguments**

\( n \)
Is the number of bytes that local variables and compiler temporaries can occupy before stack checking is activated. This is called the *threshold*.

**Default**

/Gs
Stack checking occurs for routines that require more than 4KB (4096 bytes) of stack space. This is also the default if you do not specify \( n \).

**Description**

This option lets you control the threshold at which the stack checking routine is called or not called. If a routine's local stack allocation exceeds the threshold \( (n) \), the compiler inserts a __chkstk() call into the prologue of the routine.

**IDE Equivalent**
None

**Alternate Options**
None

GS
*Determines whether the compiler generates code that detects some buffer overruns.*

**Syntax**

**Linux OS:**
None

**Windows OS:**
/GS[:keyword]
/GS-

**Arguments**

\( keyword \)
Specifies the level of stack protection heuristics used by the compiler. Possible values are:
Tells the compiler to ignore buffer overruns. This is the same as specifying /GS-.

partial

Tells the compiler to provide a stack protection level that is compatible with Microsoft* Visual Studio 2008.

strong

Tells the compiler to provide full stack security level checking. This setting is compatible with more recent Microsoft* Visual Studio stack protection heuristics. This is the same as specifying /GS with no keyword.

Default

/GS- The compiler does not detect buffer overruns.

Description

This option determines whether the compiler generates code that detects some buffer overruns that overwrite a function's return address, exception handler address, or certain types of parameters. This option has been added for Microsoft compatibility.

Following Visual Studio 2008, the Microsoft implementation of option /GS became more extensive (for example, more routines are protected). The performance of some programs may be impacted by the newer heuristics. In such cases, you may see better performance if you specify /GS:partial.

For more details about option /GS, see the Microsoft documentation.

IDE Equivalent

Visual Studio

Visual Studio: Code Generation > Security Check

Eclipse

Eclipse: None

Alternate Options

DPC++: None

C++: Linux: -fstack-security-check

C++: Windows: None

See Also

fstack-security-check compiler option
fstack-protector compiler option

malign-double

Determines whether double, long double, and long long types are naturally aligned. This option is equivalent to specifying option align. This content is specific to C++; it does not apply to DPC++.

Architecture Restrictions

Only available on IA-32 architecture
Syntax

Linux OS:
- -malign-double
  -mno-align-double

Windows OS:
None

Arguments
None

Default
- -mno-align-double

Types are aligned according to the gcc model, which means they are aligned to 4-byte boundaries.

Description

This content is specific to C++; it does not apply to DPC++. For details, see the align option.

IDE Equivalent
None

Alternate Options
None

mcmmodel

Tells the compiler to use a specific memory model to generate code and store data.

Architecture Restrictions

Only available on Intel® 64 architecture

Syntax

Linux OS:
- -mcmmodel=mem_model

Windows OS:
None

Arguments

mem_model

Is the memory model to use. Possible values are:

small

Tells the compiler to restrict code and data to the first 2GB of address space. All accesses of code and data can be done with Instruction Pointer (IP)-relative addressing.
medium

Tells the compiler to restrict code to the first 2GB; it places no memory restriction on data. Accesses of code can be done with IP-relative addressing, but accesses of data must be done with absolute addressing.

large

Places no memory restriction on code or data. All accesses of code and data must be done with absolute addressing.

Default

- \texttt{mcmodel=small}
  
  On systems using Intel® 64 architecture, the compiler restricts code and data to the first 2GB of address space. Instruction Pointer (IP)-relative addressing can be used to access code and data.

Description

This option tells the compiler to use a specific memory model to generate code and store data. It can affect code size and performance. If your program has global and static data with a total size smaller than 2GB, \texttt{mcmodel=small} is sufficient. Global and static data larger than 2GB requires \texttt{mcmodel=medium} or \texttt{mcmodel=large}. Allocation of memory larger than 2GB can be done with any setting of \texttt{mcmodel}.

IP-relative addressing requires only 32 bits, whereas absolute addressing requires 64-bits. IP-relative addressing is somewhat faster. So, the \texttt{small} memory model has the least impact on performance.

NOTE

This content is specific to C++; it does not apply to DPC++.

When you specify option \texttt{mcmodel=medium} or \texttt{mcmodel=large}, it sets option \texttt{shared-intel}. This ensures that the correct dynamic versions of the Intel run-time libraries are used.

If you specify option \texttt{static-intel} \textbf{while} \texttt{mcmodel=medium} or \texttt{mcmodel=large} is set, an error will be displayed.

IDE Equivalent

None

Alternate Options

None

Example

The following example shows how to compile using \texttt{mcmodel}:

This content is specific to C++; it does not apply to DPC++.

\texttt{icx -shared-intel -mcmodel=medium -o prog prog.c}

See Also

\texttt{shared-intel} compiler option

\texttt{fpic} compiler option

Qlong-double

\textit{Changes the default size of the long double data type.}
Syntax

Linux OS:
None

Windows OS:
/Qlong-double

Arguments
None

Default
OFF The default size of the long double data type is 64 bits.

Description
This option changes the default size of the long double data type to 80 bits. However, the alignment requirement of the data type is 16 bytes, and its size must be a multiple of its alignment, so the size of a long double on Windows* is also 16 bytes. Only the lower 10 bytes (80 bits) of the 16 byte space will have valid data stored in it.

NOTE
Using the Qlong-double command-line option on Windows* platforms requires that any source code using double extended precision floating-point types (FP80) be carefully segregated from source code that was not written in a way that considers or supports their use. When this option is used, source code that makes assumptions or has requirements on the size or layout of an FP80 value may experience a variety of failures at compile time, link time, or run time.

The Microsoft* C Standard Library and Microsoft* C++ Standard Template Library do not support FP80 datatypes. In all circumstances where you want to use this option, please check with your library vendor to determine whether they support FP80 datatype formats.

For example, the Microsoft* compiler and Microsoft*-provided library routines (such as printf or long double math functions) do not provide support for 80-bit floating-point values and should not be called from code compiled with the Qlong-double command-line option.

Starting with the Microsoft Visual Studio 2019 version 16.10 release, you may get compilation errors when using options /std:c++latest together with /Qlong-double in programs that directly or indirectly include the <complex> header, <xutility> header, or the <cmath> header. To see an example of this, see the Example section below.

IDE Equivalent
None

Alternate Options
None
Example

In the Note above, we mention an issue with using the options /std:c++latest together with /Qlong-double in programs that directly or indirectly include the <complex>, <xutility>, or the <cmath> headers. The following shows an example of this issue:

```cpp
#include <iostream>
#include <complex>

int main()
{long double ld2 = 1256789.98765432106L;int iNan = isnan(ld2);std::cout << "Hello World!\n"; }
ksh-3.2$ icl -c -EHsc -GR -std:c++latest /Qlong-double /MD test1.cpp
Intel(R) C++ Intel(R) 64 Compiler Classic for applications running on Intel(R) 64, Version xxx Build xxxx
Copyright (C) 1985-2021 Intel Corporation. All rights reserved.
test1.cpp
c:/Program files/Microsoft Visual Studio/2022/Preview/VC/Tools/MSVC/14.29.30130/include/
xutility(5918): error: no instance of function template "std::_Bit_cast" matches the argument list
argument types are: (const long double)
const auto _Bits = _Bit_cast<_Uint_type>(_Xx);
c:/Program files/Microsoft Visual Studio/2022/Preview/VC/Tools/MSVC/14.29.30130/include/
xutility(67): note: this candidate was rejected because at least one template argument could not be deduced
_NODISCARD _CONSTEXPR_BIT_CAST _To _Bit_cast(const _From& _Val) noexcept {
    detected during:
    instantiation of "auto std::_Float_abs_bits(const _Ty &)[with _Ty=long double," at line 5967
    instantiation of "bool std::_Is_finite(_Ty)[with _Ty=long double," at line 1307 of "c:/Program files/Microsoft Visual Studio/2022/Preview/VC/Tools/MSVC/14.29.30130/include/cmath"
    instantiation of ":_Ty std::_Common_lerp(_Ty, _Ty, _Ty) noexcept [with _Ty=long double]" at line 1392 of "c:/Program files/Microsoft Visual Studio/2022/Preview/VC/Tools/MSVC/14.29.30130/include/cmath"
```

Compiler Diagnostic Options

This section contains descriptions for compiler options that pertain to compiler diagnostics.

w

Disables all warning messages.

Syntax

Linux OS:

-w

Windows OS:

/w
Arguments
None

Default
OFF  Default warning messages are enabled.

Description
This option disables all warning messages.

IDE Equivalent

Windows
Visual Studio: **General > Warning Level**

Linux
Eclipse: **General > Warning Level**

Alternate Options

Linux: `-w0`
Windows: `/W0`

w, W
*Specifies the level of diagnostic messages to be generated by the compiler.*

Syntax

Linux OS:
`-wn`

Windows OS:
`/wn`

Arguments

Is the level of diagnostic messages to be generated. Possible values are:

- **0**: Enables diagnostics for errors. Disables diagnostics for warnings.
- **1**: Enables diagnostics for warnings and errors. On Linux* systems, additional warnings are enabled. On Windows* systems, this setting is equivalent to level 1 (n = 1).
- **2**: Enables diagnostics for remarks, warnings, and errors. Additional warnings are also enabled above level 2 (n = 2). This level is recommended for production purposes.
Enables diagnostics for all level 3 (n = 3) warnings plus informational warnings and remarks, which in most cases can be safely ignored. This value is only available on Windows* systems.

Enables diagnostics for all remarks, warnings, and errors. This setting produces the most diagnostic messages. This value is only available on Windows* systems.

**Default**

n=1

The compiler displays diagnostics for warnings and errors.

**Description**

This option specifies the level of diagnostic messages to be generated by the compiler.

On Windows systems, option /W4 is equivalent to option /Wall.

The -wn, /wn, and Wall options can override each other. The last option specified on the command line takes precedence.

**IDE Equivalent**

**Windows**

Visual Studio: General > Warning Level

**Linux**

Eclipse: General > Warning Level

**Alternate Options**

None

**See Also**

Wall compiler option

**Wabi**

*Determines whether a warning is issued if generated code is not C++ ABI compliant.*

**Syntax**

**Linux OS:**

-Wabi

-Wno-abi

**Windows OS:**

None

**Arguments**

None
Default

-Wno-abi

No warning is issued when generated code is not C++ ABI compliant.

Description

This option determines whether a warning is issued if generated code is not C++ ABI compliant.

IDE Equivalent

None

Alternate Options

None

Wall

* Enables warning and error diagnostics.*

Syntax

Linux OS:

-Wall

Windows OS:

/Wall

Arguments

None

Default

OFF Only default warning diagnostics are enabled.

Description

This option enables many warning and error diagnostics.

On Windows* systems, this option is equivalent to the /W4 option. It enables diagnostics for all level 3 warnings plus informational warnings and remarks.

However, on Linux* systems, this option is similar to gcc option -Wall. It displays all errors and some of the warnings that are typically reported by gcc option -Wall. If you want to display all warnings, specify the -w2 or -w3 option.

The Wall, -wn, and /Wn options can override each other. The last option specified on the command line takes precedence.

IDE Equivalent

None

Alternate Options

None

See Also

w, W  compiler option
**Wcomment**

_Determines whether a warning is issued when /* appears in the middle of a /* */ comment._

**Syntax**

**Linux OS:**

- `Wcomment`
- `Wno-comment`

**Windows OS:**

None

**Arguments**

None

**Default**

`-Wno-comment`  
_No warning is issued when /* appears in the middle of a /* */ comment._

**Description**

This option determines whether a warning is issued when /* appears in the middle of a /* */ comment.

**IDE Equivalent**

None

**Alternate Options**

None

**Wdeprecated**

_Determines whether warnings are issued for deprecated C++ headers._

**Syntax**

**Linux OS:**

- `Wdeprecated`
- `Wno-deprecated`

**Windows OS:**

None

**Arguments**

None

**Default**

`-Wdeprecated`  
_The compiler issues warnings for deprecated C++ headers._


**Description**

This option determines whether warnings are issued for deprecated C++ headers. It has no effect in C compilation mode.

Option `-Wdeprecated` enables these warnings by defining the __DEPRECATED macro for preprocessor.

To disable warnings for deprecated C++ headers, specify `-Wno-deprecated`.

**IDE Equivalent**

None

**Alternate Options**

None

**Weffc++, Qeffc++**

*Enables warnings based on certain C++ programming guidelines.*

**Syntax**

**Linux OS:**

- `-Weffc++`

**Windows OS:**

- `/Qeffc++`

**Arguments**

None

**Default**

OFF  

Diagnostics are not enabled.

**Description**

This option enables warnings based on certain programming guidelines developed by Scott Meyers in his books on effective C++ programming. With this option, the compiler emits warnings for these guidelines:

- Use `const` and `inline` rather than `#define`. Note that you will only get this in user code, not system header code.
- Use `<iostream>` rather than `<stdio.h>`.
- Use `new` and `delete` rather than `malloc` and `free`.
- Use C++ style comments in preference to C style comments. C comments in system headers are not diagnosed.
- Use `delete` on pointer members in destructors. The compiler diagnoses any pointer that does not have a `delete`.
- Make sure you have a user copy constructor and assignment operator in classes containing pointers.
- Use initialization rather than assignment to members in constructors.
- Make sure the initialization list ordering matches the declaration list ordering in constructors.
- Make sure base classes have virtual destructors.
- Make sure `operator=` returns `*this`.
- Make sure prefix forms of increment and decrement return a `const` object.
- Never overload operators `&`, `|`, and `.`
NOTE
The warnings generated by this compiler option are based on the following books from Scott Meyers:

- Effective C++ Second Edition - 50 Specific Ways to Improve Your Programs and Designs
- More Effective C++ - 35 New Ways to Improve Your Programs and Designs

IDE Equivalent

Windows
Visual Studio: None

Linux
Eclipse: Compilation Diagnostics > Enable Warnings for Style Guideline Violations

Alternate Options
None

Werror, WX
Changes all warnings to errors.

Syntax

Linux OS:
-Werror

Windows OS:
/WX

Arguments
None

Default
OFF  The compiler returns diagnostics as usual.

Description
This option changes all warnings to errors.

IDE Equivalent

Windows
Visual Studio: General > Treat Warnings As Errors

Linux
Eclipse: Compilation Diagnostics > Treat Warnings As Errors

Alternate Options
None
**Werror-all**

*Causes all warnings and currently enabled remarks to be reported as errors.*

**Syntax**

**Linux OS:**

- `Werror-all`

**Windows OS:**

/`Werror-all`

**Arguments**

None

**Default**

OFF

The compiler returns diagnostics as usual.

**Description**

This option causes all warnings and currently enabled remarks to be reported as errors.

**IDE Equivalent**

None

**Alternate Options**

None

**Wextra-tokens**

*Determines whether warnings are issued about extra tokens at the end of preprocessor directives.*

**Syntax**

**Linux OS:**

- `Wextra-tokens`
- `Wno-extra-tokens`

**Windows OS:**

None

**Arguments**

None

**Default**

- `Wno-extra-tokens`

The compiler does not warn about extra tokens at the end of preprocessor directives.

**Description**

This option determines whether warnings are issued about extra tokens at the end of preprocessor directives.
IDE Equivalent
None

Alternate Options
None

Wformat
Determines whether argument checking is enabled for calls to printf, scanf, and so forth.

Syntax
Linux OS:
-Wformat
-Wno-format

Windows OS:
None

Arguments
None

Default
-Wno-format Argument checking is not enabled for calls to printf, scanf, and so forth.

Description
This option determines whether argument checking is enabled for calls to printf, scanf, and so forth.

IDE Equivalent
None

Alternate Options
None

Wformat-security
Determines whether the compiler issues a warning when the use of format functions may cause security problems.

Syntax
Linux OS:
-Wformat-security
-Wno-format-security

Windows OS:
None
**Arguments**
None

**Default**

- `Wno-format-security`  
  No warning is issued when the use of format functions may cause security problems.

**Description**
This option determines whether the compiler issues a warning when the use of format functions may cause security problems.

When `Wformat-security` is specified, it warns about uses of format functions where the format string is not a string literal and there are no format arguments.

**IDE Equivalent**
None

**Alternate Options**
None

**Wmain**
*Determine whether a warning is issued if the return type of main is not expected.*

**Syntax**

**Linux OS:**

- `Wmain`
- `Wno-main`

**Windows OS:**
None

**Arguments**
None

**Default**

- `Wno-main`  
  No warning is issued if the return type of `main` is not expected.

**Description**
This option determines whether a warning is issued if the return type of `main` is not expected.

**IDE Equivalent**
None

**Alternate Options**
None
**Wmissing-declarations**

*Determines whether warnings are issued for global functions and variables without prior declaration.*

### Syntax

**Linux OS:**

- `-Wmissing-declarations`
- `-Wno-missing-declarations`

**Windows OS:**

None

### Arguments

None

### Default

- `-Wno-missing-declarations`

No warnings are issued for global functions and variables without prior declaration.

### Description

This option determines whether warnings are issued for global functions and variables without prior declaration.

### IDE Equivalent

None

### Alternate Options

None

**Wmissing-prototypes**

*Determines whether warnings are issued for missing prototypes.*

### Syntax

**Linux OS:**

- `-Wmissing-prototypes`
- `-Wno-missing-prototypes`

**Windows OS:**

None

### Arguments

None

### Default

- `-Wno-missing-prototypes`

No warnings are issued for missing prototypes.
**Description**
Determines whether warnings are issued for missing prototypes.

If `-Wmissing-prototypes` is specified, it tells the compiler to detect global functions that are defined without a previous prototype declaration.

**IDE Equivalent**
None

**Alternate Options**
None

**Wpointer-arith**
Determines whether warnings are issued for questionable pointer arithmetic.

**Syntax**

**Linux OS:**
- `-Wpointer-arith`
- `-Wno-pointer-arith`

**Windows OS:**
None

**Arguments**
None

**Default**
- `-Wno-pointer-arith`  
  No warnings are issued for questionable pointer arithmetic.

**Description**
Determines whether warnings are issued for questionable pointer arithmetic.

**IDE Equivalent**
None

**Alternate Options**
None

**Wreorder**
*Tells the compiler to issue a warning when the order of member initializers does not match the order in which they must be executed.*

**Syntax**

**Linux OS:**
- `-Wreorder`
**Windows OS:**
None

**Arguments**
None

**Default**
OFF The compiler does not issue a warning.

**Description**
This option tells the compiler to issue a warning when the order of member initializers does not match the order in which they must be executed. This option is supported for C++ only.

**IDE Equivalent**
None

**Alternate Options**
None

**Wreturn-type**
*Determines whether warnings are issued when a function is declared without a return type, when the definition of a function returning void contains a return statement with an expression, or when the closing brace of a function returning non-void is reached.*

**Syntax**

**Linux OS:**
-Wreturn-type
-Wno-return-type

**Windows OS:**
None

**Arguments**
None

**Default**
ON for one condition A warning is issued when the closing brace of a function returning non-void is reached.

**Description**
This option determines whether warnings are issued for the following:

- When a function is declared without a return type
- When the definition of a function returning void contains a return statement with an expression
- When the closing brace of a function returning non-void is reached

Specify -Wno-return-type if you do not want to see warnings about the above diagnostics.
IDE Equivalent
None

Alternate Options
None

Wshadow
Determines whether a warning is issued when a variable declaration hides a previous declaration.

Syntax
Linux OS:
-Wshadow
-Wno-shadow

Windows OS:
None

Arguments
None

Default
-Wno-shadow
No warning is issued when a variable declaration hides a previous declaration.

Description
This option determines whether a warning is issued when a variable declaration hides a previous declaration. Same as -ww1599.

IDE Equivalent
None

Alternate Options
None

Wsing-compare
Determines whether warnings are issued when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.

Syntax
Linux OS:
-Wsing-compare
-Wno-sing-compare

Windows OS:
None
Arguments
None

Default

-Wno-sign-compare

The compiler does not issue these warnings

Description
This option determines whether warnings are issued when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.

On Linux* systems, this option is provided for compatibility with gcc.

IDE Equivalent
None

Alternate Options
None

Wstrict-aliasing
Determines whether warnings are issued for code that might violate the optimizer's strict aliasing rules.

Syntax
Linux OS:
-Wstrict-aliasing
-Wno-strict-aliasing

Windows OS:
None

Arguments
None

Default

-Wno-strict-aliasing

No warnings are issued for code that might violate the optimizer's strict aliasing rules.

Description
This option determines whether warnings are issued for code that might violate the optimizer's strict aliasing rules. These warnings will only be issued if you also specify option -fstrict-aliasing.

IDE Equivalent
None

Alternate Options
None
**Wstrict-prototypes**
*Determines whether warnings are issued for functions declared or defined without specified argument types.*

**Syntax**

**Linux OS:**
- `Wstrict-prototypes`
- `Wno-strict-prototypes`

**Windows OS:**
None

**Arguments**
None

**Default**
- `Wno-strict-prototypes`
No warnings are issued for functions declared or defined without specified argument types.

**Description**
This option determines whether warnings are issued for functions declared or defined without specified argument types.

**IDE Equivalent**
None

**Alternate Options**
None

**Wtrigraphs**
*Determines whether warnings are issued if any trigraphs are encountered that might change the meaning of the program.*

**Syntax**

**Linux OS:**
- `Wtrigraphs`
- `Wno-trigraphs`

**Windows OS:**
None

**Arguments**
None
Default
-Wno-trigraphs

No warnings are issued if any trigraphs are encountered that might change the meaning of the program.

Description
This option determines whether warnings are issued if any trigraphs are encountered that might change the meaning of the program.

IDE Equivalent
None

Alternate Options
None

Wuninitialized
Determines whether a warning is issued if a variable is used before being initialized.

Syntax
Linux OS:
-Wuninitialized
-Wno-uninitialized

Windows OS:
None

Arguments
None

Default
-Wno-uninitialized

No warning is issued if a variable is used before being initialized.

Description
This option determines whether a warning is issued if a variable is used before being initialized. Equivalent to -ww592 and -wd592.

IDE Equivalent
None

Alternate Options
-ww592 and -wd592

Wunknown-pragmas
Determines whether a warning is issued if an unknown #pragma directive is used.
Syntax
Linux OS:
- `Wunknown-pragmas`
- `Wno-unknown-pragmas`

Windows OS:
None

Arguments
None

Default

- `Wunknown-pragmas`  A warning is issued if an unknown `#pragma` directive is used.

Description
This option determines whether a warning is issued if an unknown `#pragma` directive is used.

IDE Equivalent
None

Alternate Options
None

**Wunused-function**

*Determines whether a warning is issued if a declared function is not used.*

Syntax
Linux OS:
- `Wunused-function`
- `Wno-unused-function`

Windows OS:
None

Arguments
None

Default

- `Wno-unused-function`  No warning is issued if a declared function is not used.

Description
This option determines whether a warning is issued if a declared function is not used.

IDE Equivalent
None
Alternate Options
None

**Wunused-variable**

*Determine whether a warning is issued if a local or non-constant static variable is unused after being declared.*

Syntax

**Linux OS:**
- `-Wunused-variable`
- `-Wno-unused-variable`

**Windows OS:**
None

Arguments
None

Default

`-Wno-unused-variable`

No warning is issued if a local or non-constant static variable is unused after being declared.

Description

This option determines whether a warning is issued if a local or non-constant static variable is unused after being declared.

IDE Equivalent

None

Alternate Options
None

**Wwrite-strings**

*Issues a diagnostic message if const char * is converted to (non-const) char*.

Syntax

**Linux OS:**
- `-Wwrite-strings`

**Windows OS:**
None

Arguments
None
Default
OFF

No diagnostic message is issued if const char * is converted to (non-const) char*.

Description
This option issues a diagnostic message if const char* is converted to (non-const) char *.

IDE Equivalent
None

Alternate Options
None

Compatibility Options
This section contains descriptions for compiler options that pertain to language compatibility.

gcc-toolchain
Lets you specify the location of the base toolchain.

Syntax
Linux OS:
--gcc-toolchain=dir
Windows OS:
None

Arguments

dir
Is the location of the base toolchain.

Default
OFF

The compiler uses heuristics to locate the base toolchain.

Description
This option lets you specify the location of the base toolchain.

IDE Equivalent
None

Alternate Options
None

vmv
Enables pointers to members of any inheritance type.

Syntax
Linux OS:
None
Windows OS:

/vmv

Arguments
None

Default
OFF The compiler uses default rules to represent pointers to members.

Description
This option enables pointers to members of any inheritance type. To use this option, you must also specify option /vmg.

IDE Equivalent
None

Alternate Options
None

Linking or Linker Options
This section contains descriptions for compiler options that pertain to linking or to the linker.

Bdynamic
Enables dynamic linking of libraries at run time. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
-Bdynamic

Windows OS:
None

Arguments
None

Default
OFF Limited dynamic linking occurs.

Description
This option enables dynamic linking of libraries at run time. Smaller executables are created than with static linking.

This option is placed in the linker command line corresponding to its location on the user command line. It controls the linking behavior of any library that is passed using the command line.
All libraries on the command line following option -Bdynamic are linked dynamically until the end of the command line or until a -Bstatic option is encountered. The -Bstatic option enables static linking of libraries.

**IDE Equivalent**
None

**Alternate Options**
None

**See Also**
Bstatic compiler option

**Bstatic**

*Enables static linking of a user's library. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**

- Bstatic

**Windows OS:**

None

**Arguments**

None

**Default**

OFF

Default static linking occurs.

**Description**

This option enables static linking of a user's library.

This option is placed in the linker command line corresponding to its location on the user command line. It controls the linking behavior of any library that is passed using the command line.

All libraries on the command line following option -Bstatic are linked statically until the end of the command line or until a -Bdynamic option is encountered. The -Bdynamic option enables dynamic linking of libraries.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

Bdynamic compiler option
**Bsymbolic**

*Binds references to all global symbols in a program to the definitions within a user's shared library. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**

-Bsymbolic

**Windows OS:**

None

**Arguments**

None

**Default**

OFF  

When a program is linked to a shared library, it can override the definition within the shared library.

**Description**

This option binds references to all global symbols in a program to the definitions within a user's shared library.

This option is only meaningful on Executable Linkage Format (ELF) platforms that support shared libraries.

**Caution**

This option can have unintended side-effects of disabling symbol preemption in the shared library.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

*Bsymbolic-functions* compiler option

**Bsymbolic-functions**

*Binds references to all global function symbols in a program to the definitions within a user's shared library. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**

-Bsymbolic-functions
Windows OS:
None

Arguments
None

Default
OFF When a program is linked to a shared library, it can override the definition within the shared library.

Description
This option binds references to all global function symbols in a program to the definitions within a user's shared library.

This option is only meaningful on Executable Linkage Format (ELF) platforms that support shared libraries.

Caution
This option can have unintended side-effects of disabling symbol preemption in the shared library.

IDE Equivalent
None

Alternate Options
None

See Also
Bsymblic compiler option

dynamic-linker
Specifies a dynamic linker other than the default. This content is specific to C++; it does not apply to DPC++.

Syntax
Linux OS:
-dynamic-linker file

Windows OS:
None

Arguments
file Is the name of the dynamic linker to be used.

Default
OFF The default dynamic linker is used.
Description
This option lets you specify a dynamic linker other than the default.

IDE Equivalent
None

Alternate Options
None

F (Windows*)
Specifies the stack reserve amount for the program. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
None

Windows OS:
/Fn

Arguments

n Is the stack reserve amount. It can be specified as a decimal integer or as a hexadecimal constant by using a C-style convention (for example, /F0x1000).

Default
OFF The stack size default is chosen by the operating system.

Description
This option specifies the stack reserve amount for the program. The amount (n) is passed to the linker. Note that the linker property pages have their own option to do this.

IDE Equivalent
None

Alternate Options
None

fixed
Causes the linker to create a program that can be loaded only at its preferred base address. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
None
Windows OS:

/fixe

Arguments
None

Default
OFF The compiler uses default methods to load programs.

Description
This option is passed to the linker, causing it to create a program that can be loaded only at its preferred base address.

IDE Equivalent
None

Alternate Options
None

Fm
Tell the linker to generate a link map file. This is a deprecated option. There is no replacement option. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
None

Windows OS:

/Fm[filename|dir]

Arguments

filename Is the name for the link map file.

dir Is the directory where the link map file should be placed. It can include file.
Alternate Options
None

**fuse-ld**
*Tells the compiler to use a different linker instead of the default linker (ld).*

**Syntax**

**Linux OS:**
- `fuse-ld=keyword`

**Windows OS:**
None

**Arguments**

`keyword`
 Possible values are:

- `bfd`  
  Tells the compiler to use the bfd linker.
- `gold`  
  Tells the compiler to use the gold linker.

**Default**

Id  
The compiler uses the ld linker by default.

**Description**

This option tells the compiler to use a different linker instead of default linker (ld).
This option is provided for compatibility with gcc.

**IDE Equivalent**

None

Alternate Options
None

**l**
*Tells the linker to search for a specified library when linking.*

**Syntax**

**Linux OS:**
- `-lstring`

**Windows OS:**
None

**Arguments**

`string`  
Specifies the library (libstring) that the linker should search.
Default

OFF  The linker searches for standard libraries in standard directories.

Description
This option tells the linker to search for a specified library when linking.
When resolving references, the linker normally searches for libraries in several standard directories, in
directories specified by the \texttt{L} option, then in the library specified by the \texttt{l} option.
The linker searches and processes libraries and object files in the order they are specified. So, you should
specify this option following the last object file it applies to.

IDE Equivalent
None

Alternate Options
None

See Also
\texttt{L}  compiler option

\texttt{L}

\textit{Tells the linker to search for libraries in a specified
directory before searching the standard directories.}

Syntax

Linux OS:
\texttt{-Ldir}

Windows OS:
None

Arguments

dir  

Is the name of the directory to search for libraries.

Default

OFF  The linker searches the standard directories for libraries.

Description
This option tells the linker to search for libraries in a specified directory before searching for them in the
standard directories.

IDE Equivalent
None

Alternate Options
None

See Also
\texttt{l}  compiler option
**LD**
Specifies that a program should be linked as a dynamic-link (DLL) library.

**Syntax**

Linux OS: None

Windows OS: 
/ LD
/ LDd

**Arguments**
None

**Default**
OFF The program is not linked as a dynamic-link (DLL) library.

**Description**
This option specifies that a program should be linked as a dynamic-link (DLL) library instead of an executable (.exe) file. You can also specify /LDd, where d indicates a debug version.

**IDE Equivalent**
None

**Alternate Options**
None

**link**
Passes user-specified options directly to the linker at compile time.

**Syntax**

Linux OS: None

Windows OS: 
/link

**Arguments**
None

**Default**
OFF No user-specified options are passed directly to the linker.

**Description**
This option passes user-specified options directly to the linker at compile time.
All options that appear following `/link` are passed directly to the linker.

**IDE Equivalent**
None

**Alternate Options**
None

**See Also**
Xlinker compiler option

**MD**
*Tells the linker to search for unresolved references in a multithreaded, dynamic-link run-time library.*

**Syntax**

**Linux OS:**
None

**Windows OS:**
/MD
/MDd

**Arguments**
None

**Default**
OFF

**Description**
This option tells the linker to search for unresolved references in a multithreaded, dynamic-link (DLL) run-time library. You can also specify `/MDd`, where `d` indicates a debug version.

This option is processed by the compiler, which adds directives to the compiled object file that are processed by the linker.

**IDE Equivalent**

**Visual Studio**
Visual Studio: **Code Generation > Runtime Library**

**Eclipse**
Eclipse: None

**Alternate Options**
None

**MT**
*Tells the linker to search for unresolved references in a multithreaded, static run-time library. This content is specific to C++; it does not apply to DPC++.**
Syntax

Linux OS:
None

Windows OS:
/MT
/MTd

Arguments
None

Default
/MT

The linker searches for unresolved references in a multithreaded, static run-time library.

Description
This option tells the linker to search for unresolved references in a multithreaded, static run-time library. You can also specify /MTd, where d indicates a debug version.

This option is processed by the compiler, which adds directives to the compiled object file that are processed by the linker.

IDE Equivalent

Visual Studio
Visual Studio: Code Generation > Runtime Library

Eclipse
Eclipse: None

Alternate Options
None

See Also

no-libgcc
Prevents the linking of certain gcc-specific libraries. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
-no-libgcc

Windows OS:
None

Arguments
None
**Default**

OFF

**Description**
This option prevents the linking of certain gcc-specific libraries.
This option is not recommended for general use.

**IDE Equivalent**

None

**Alternate Options**

None

**nodefaultlibs**

*Prevents the compiler from using standard libraries when linking.*

**Syntax**

**Linux OS:**

- `nodefaultlibs`

**Windows OS:**

None

**Arguments**

None

**Default**

OFF  The standard libraries are linked.

**Description**

This option prevents the compiler from using standard libraries when linking. On Linux* systems, it is provided for GNU compatibility.

**IDE Equivalent**

**Windows**

Visual Studio: None

**Linux**

Eclipse: **Libraries > Use no system libraries**

**Alternate Options**

None

**See Also**

nostdlib  compiler option
no-intel-lib, Qno-intel-lib
Disables linking to specified Intel® libraries, or to all Intel® libraries.

Syntax

Linux OS:
-no-intel-lib[=library]

Windows OS:
/Qno-intel-lib[;library]

Arguments

library Indicates which Intel® library should not be linked. Possible values are:

libirc Disables linking to the Intel® C/C++ library.
libimf Disables linking to the Intel® Math library. This value is only available for Linux*.
libsxml Disables linking to the Intel® Short Vector Math library.
libirng Disables linking to the Random Number Generator library.

If you specify more than one library, they must be separated by commas.
If library is omitted, the compiler will not link to any of the Intel® libraries shown above.

Default

OFF If this option is not specified, the compiler uses default heuristics for linking to libraries.

Description

This option disables linking to specified Intel® libraries, or to all Intel® libraries.

NOTE
When offloading is enabled, this option only applies to host compilation.

IDE Equivalent

None

Alternate Options

None

nostartfiles
Prevents the compiler from using standard startup files when linking.
Syntax

Linux OS:
-nostartfiles

Windows OS:
None

Arguments
None

Default
OFF  The compiler uses standard startup files when linking.

Description
This option prevents the compiler from using standard startup files when linking.

IDE Equivalent
None

Alternate Options
None

See Also
nostdlib  compiler option

nostdlib
Prevents the compiler from using standard libraries and startup files when linking.

Syntax

Linux OS:
-nostdlib

Windows OS:
None

Arguments
None

Default
OFF  The compiler uses standard startup files and standard libraries when linking.

Description
This option prevents the compiler from using standard libraries and startup files when linking. On Linux* systems, it is provided for GNU compatibility.

IDE Equivalent
None
Alternate Options
None

See Also
nodefaultlibs compiler option
nostartfiles compiler option

pie
Determines whether the compiler generates position-independent code that will be linked into an executable.

Syntax
Linux OS:
-pie
-no-pie

Windows OS:
None

Arguments
None

Default
varies
On Linux*, the default is -no-pie.

Description
This option determines whether the compiler generates position-independent code that will be linked into an executable. To enable generation of position-independent code that will be linked into an executable, specify -pie.

To disable generation of position-independent code that will be linked into an executable, specify -no-pie.

IDE Equivalent
None

Alternate Options
None

See Also
fpic compiler option

pthread
Tells the compiler to use pthreads library for multithreading support.

Syntax
Linux OS:
-pthread
Windows OS:
None

Arguments
None

Default
OFF  The compiler does not use pthreads library for multithreading support.

Description
Tells the compiler to use pthreads library for multithreading support.

IDE Equivalent
None

Alternate Options
None

shared
Tells the compiler to produce a dynamic shared object instead of an executable.

Syntax
Linux OS:
- shared

Windows OS:
None

Arguments
None

Default
OFF  The compiler produces an executable.

Description
This option tells the compiler to produce a dynamic shared object (DSO) instead of an executable. This includes linking in all libraries dynamically and passing -shared to the linker.
You must specify option fpic for the compilation of each object file you want to include in the shared library.

IDE Equivalent
None

Alternate Options
None

See Also
fpic  compiler option
Xlinker compiler option

shared-intel

Causes Intel-provided libraries to be linked in dynamically. This content is specific to C++; it does not apply to DPC++.

Syntax

**Linux OS:**
- `shared-intel`

**Windows OS:**
- None

**Arguments**

None

**Default**

OFF

Intel® libraries are linked in statically, with the exception of Intel's OpenMP® runtime support library, which is linked in dynamically unless you specify option `-qopenmp-link=static`.

**Description**

This option causes Intel-provided libraries to be linked in dynamically. It is the opposite of `-static-intel`. This option is processed by the icx or icpx command that initiates linking, adding library names explicitly to the link command.

If you specify option `-mcmodel=medium` or `-mcmodel=large`, it sets option `-shared-intel`.

**IDE Equivalent**

**Visual Studio**

Visual Studio: None

**Eclipse**

Eclipse: None

**Alternate Options**

None

**See Also**

- `static-intel` compiler option
- `qopenmp-link` compiler option
- `shared-libgcc` compiler option

**shared-libgcc**

Links the GNU libgcc library dynamically.

**Syntax**

**Linux OS:**
- `shared-libgcc`
Windows OS:
None

Arguments
None

Default
-shared-libgcc  The compiler links the libgcc library dynamically.

Description
This option links the GNU libgcc library dynamically. It is the opposite of option static-libgcc.
This option is processed by the icx or icpx command (C++) or the dpcpp command (DPC++) that initiates linking, adding library names explicitly to the link command.
This option is useful when you want to override the default behavior of the static option, which causes all libraries to be linked statically.

IDE Equivalent
None

Alternate Options
None

See Also
static-libgcc  compiler option

static  
Prevents linking with shared libraries.

Syntax
Linux OS:
-static

Windows OS:
None

Arguments
None

Default
OFF  The compiler links with shared libraries except as otherwise specified by -static-intel or its default.

Description
This option prevents linking with shared libraries. It causes the executable to link all libraries statically.
NOTE
This option does not cause static linking of libraries for which no static version is available. These libraries can only be linked dynamically.

IDE Equivalent

Visual Studio
Visual Studio: None

Eclipse
Eclipse: Libraries > Link with static libraries

Alternate Options
None

See Also
static-intel compiler option

static-intel
Causes Intel-provided libraries to be linked in statically. This content is specific to C++; it does not apply to DPC++.

Syntax

Linux OS:
-static-intel

Windows OS:
None

Arguments
None

Default
ON

Intel libraries are linked in statically, with the exception of Intel's OpenMP* runtime support library, which is linked in dynamically unless you specify option -qopenmp-link=static.

Description

This option causes Intel-provided libraries to be linked in statically with certain exceptions (see the Default above). It is the opposite of -shared-intel.

This option is processed by the icx or icpx command that initiates linking, adding library names explicitly to the link command.

If you specify option -static-intel while option -mcmode=medium or -mcmode=large is set, an error will be displayed.

If you specify option -static-intel and any of the Intel-provided libraries have no static version, a diagnostic will be displayed.
**static-libgcc**

*Links the GNU libgcc library statically.*

**Syntax**

**Linux OS:**

```
-static-libgcc
```

**Windows OS:**

None

**Arguments**

None

**Default**

OFF

The compiler links the GNU libgcc library dynamically.

**Description**

This option links the GNU libgcc library statically. It is the opposite of option `-shared-libgcc`.

This option is processed by the icx or icpx command (C++) or the dpcpp command (DPC++) that initiates linking, adding library names explicitly to the link command.

This option is useful when you want to override the default behavior, which causes the library to be linked dynamically.

**NOTE**

If you want to use traceback, you must also link to the static version of the libgcc library. This library enables printing of backtrace information.

**IDE Equivalent**

None

**Alternate Options**

None
See Also
shared-libgcc compiler option
static-libstdc++ compiler option

static-libstdc++
Links the GNU libstdc++ library statically.

Syntax
Linux OS:
- static-libstdc++

Windows OS:
None

Arguments
None

Default
OFF
The compiler links the GNU libstdc++ library dynamically.

Description
This option links the GNU libstdc++ library statically.
This option is processed by the icx or icpx command (C++) or the dpcpp command (DPC++) that initiates linking, adding library names explicitly to the link command.
This option is useful when you want to override the default behavior, which causes the library to be linked dynamically.

IDE Equivalent
None

Alternate Options
None

See Also
static-libgcc compiler option

T
Tells the linker to read link commands from a file.

Syntax
Linux OS:
- T filename

Windows OS:
None
Arguments

filename Is the name of the file.

Default
OFF The linker does not read link commands from a file.

Description
This option tells the linker to read link commands from a file.

IDE Equivalent
None

Alternate Options
None

u (Linux*)
_Tells the compiler the specified symbol is undefined._

Syntax

Linux OS:
-\texttt{u symbol}

Windows OS:
None

Arguments
None

Default
OFF Standard rules are in effect for variables.

Description
This option tells the compiler the specified symbol is undefined.

IDE Equivalent
None

Alternate Options
None

v
_Specifies that driver tool commands should be displayed and executed._

Syntax

Linux OS:
-\texttt{v [filename]}
**Windows OS:**
None

**Arguments**

`filename`  
Is the name of a source file to be compiled. A space must appear before the file name.

**Default**

OFF  
No tool commands are shown.

**Description**

This option specifies that driver tool commands should be displayed and executed.  
If you use this option without specifying a source file name, the compiler displays only the version of the compiler.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

`dryrun`  
compiler option

**Wa**

*Passes options to the assembler for processing.*

**Syntax**

**Linux OS:**

`-Wa,option1[,option2,...]`

**Windows OS:**

None

**Arguments**

`option`  
Is an assembler option. This option is not processed by the driver and is directly passed to the assembler.

**Default**

OFF  
No options are passed to the assembler.

**Description**

This option passes one or more options to the assembler for processing. If the assembler is not invoked, these options are ignored.

**IDE Equivalent**

None
**Alternate Options**

None

**Wl**

*Passes options to the linker for processing.*

**Syntax**

Linux OS:

\[-Wl,\text{option1[,option2,...]}\]

Windows OS:

None

**Arguments**

\textit{option}  

Is a linker option. This option is not processed by the driver and is directly passed to the linker.

**Default**

OFF  

No options are passed to the linker.

**Description**

This option passes one or more options to the linker for processing. If the linker is not invoked, these options are ignored.

This content is specific to C++; it does not apply to DPC++. This option is equivalent to specifying option \(-Qoption,\text{link},\text{options}\).

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

\textit{Qoption}  

compiler option

**Wp**

*Passes options to the preprocessor.*

**Syntax**

Linux OS:

\[-Wp,\text{option1[,option2,...]}\]

Windows OS:

None
Arguments

option

Is a preprocessor option. This option is not processed by the driver and is directly passed to the preprocessor.

Default

OFF

No options are passed to the preprocessor.

Description

This option passes one or more options to the preprocessor. If the preprocessor is not invoked, these options are ignored.

This content is specific to C++; it does not apply to DPC++. This option is equivalent to specifying option 
-Qoption, cpp, options.

IDE Equivalent

None

Alternate Options

None

See Also

Qoption  compiler option

Xlinker

Passes a linker option directly to the linker.

Syntax

Linux OS:

-Xlinker option

Windows OS:

None

Arguments

option

Is a linker option.

Default

OFF

No options are passed directly to the linker.

Description

This option passes a linker option directly to the linker. If -Xlinker -shared is specified, only -shared is passed to the linker and no special work is done to ensure proper linkage for generating a shared object.

-Xlinker just takes whatever arguments are supplied and passes them directly to the linker.

If you want to pass compound options to the linker, for example "-L $HOME/lib", you must use the following method:

-Xlinker -L -Xlinker $HOME/lib
IDE Equivalent

Windows
Visual Studio: None

Linux
Eclipse: Linker > Miscellaneous > Other Options

Alternate Options
None

See Also
shared compiler option
link compiler option

Zl
*Causes library names to be omitted from the object file.*

Syntax
Linux OS:
None

Windows OS:
/Zl

Arguments
None

Default
OFF
Default or specified library names are included in the object file.

Description
This option causes library names to be omitted from the object file.

IDE Equivalent

Windows
Visual Studio: Advanced > Omit Default Library Names

Linux
Eclipse: None

Alternate Options
None

Miscellaneous Options
This section contains descriptions for compiler options that do not pertain to a specific category.
dryrun
Specifies that driver tool commands should be shown but not executed. This content is specific to C++; it does not apply to DPC++.

Syntax
Linux OS:
-dryrun
Windows OS:
None

Arguments
None

Default
OFF  No tool commands are shown, but they are executed.

Description
This option specifies that driver tool commands should be shown but not executed.

IDE Equivalent
None

Alternate Options
None

See Also
v  compiler option

dumpmachine
Displays the target machine and operating system configuration.

Syntax
Linux OS:
-dumpmachine
Windows OS:
None

Arguments
None

Default
OFF  The compiler does not display target machine or operating system information.

Description
This option displays the target machine and operating system configuration. No compilation is performed.
IDE Equivalent
None

Alternate Options
None

See Also
dumpversion compiler option

dumpversion
Displays the version number of the compiler.

Syntax
Linux OS:
-dumpversion

Windows OS:
None

Arguments
None

Default
OFF  The compiler does not display the compiler version number.

Description
This option displays the version number of the compiler. It does not compile your source files.

IDE Equivalent
None

Alternate Options
None

See Also
dumpmachine compiler option

Gy
Separates functions into COMDATs for the linker. This is a deprecated option. There is no replacement option. This content is specific to C++; it does not apply to DPC++.

Syntax
Linux OS:
None

Windows OS:
/Gy
/Gy-
Arguments
None

Default
ON

Description
The compiler separates functions into COMDATs.

IDE Equivalent
Windows
Visual Studio: **Code Generation > Enable Function-Level Linking**

Linux
Eclipse: None

Alternate Options
None

**help**
Displays a list of supported compiler options in alphabetical order.

Syntax
Linux OS:
- help

Windows OS:
/help

Arguments
None

Default
OFF

Description
No list is displayed unless this compiler option is specified.

Alternate Options
None

**intel-freestanding**
Lets you compile in the absence of a gcc environment.
This content is specific to C++; it does not apply to DPC++.
Syntax

Linux OS:
-intel-freestanding[=ver]

Windows OS:
None

Arguments

ver

Is a three-digit number that is used to determine the gcc version that the compiler should be compatible with for compilation. It also sets the corresponding GNUC macros.

The number will be normalized to reflect the gcc compiler version numbering scheme. For example, if you specify 493, it indicates the compiler should be compatible with gcc version 4.9.3.

Default

OFF The compiler uses default heuristics when choosing the gcc environment.

Description

This option lets you compile in the absence of a gcc environment. It disables any external compiler calls (such as calls to gcc) that the compiler driver normally performs by default.

This option also removes any default search locations for header and library files. So, for successful compilation and linking, you must provide these search locations.

This option does not affect ld, as, or cpp. They will be used for compilation as needed.

NOTE

This option does not imply option -nostdinc -nostdlib. If you want to assure a clean environment for compilation (including removal of Intel-specific header locations and libs), you should specify -nostdinc and/or -nostdlib.

NOTE

This option is supported for any Linux-target compiler, including a Windows-host to Linux-target compiler.

IDE Equivalent

None

Alternate Options

None

See Also

intel-freestanding-target-os compiler option
nostdlib compiler option
nostdinc compiler option, which is an alternate option for option X
**intel-freestanding-target-os**

*Lets you specify the target operating system for compilation. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**

- `intel-freestanding-target-os=os`

**Windows OS:**

None

**Arguments**

`os`  
Is the target operating system for the Linux compiler. Currently, the only possible value is **linux**.

**Default**

**OFF**  
The installed gcc determines the target operating system.

**Description**

This option lets you specify the target operating system for compilation. It sets option `-intel-freestanding`.

**NOTE**

This option is supported for any Linux-target compiler, including a Windows-host to Linux-target compiler.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

`intel-freestanding` compiler option

`multibyte-chars, Qmultibyte-chars`  
*Determines whether multi-byte characters are supported. This content is specific to C++; it does not apply to DPC++.*

**Syntax**

**Linux OS:**

- `multibyte-chars`
- `no-multibyte-chars`
Windows OS:
/Qmultibyte-chars
/Qmultibyte-chars-

Arguments
None

Default
-multibyte-chars Multi-byte characters are supported.  
or /Qmultibyte-chars

Description
This option determines whether multi-byte characters are supported.

IDE Equivalent
Windows
Visual Studio: None

Linux
Eclipse: Language > Support Multibyte Characters in Source

Alternate Options
None

multiple-processes
Creates multiple processes that can be used to compile large numbers of source files at the same time. This content is specific to C++; it does not apply to DPC++.

Syntax
Linux OS:
-multiple-processes=[n]

Windows OS:
None

Arguments
n Is the maximum number of processes that the compiler should create.

Default
OFF A single process is used to compile source files.

Description
This option creates multiple processes that can be used to compile large numbers of source files at the same time. It can improve performance by reducing the time it takes to compile source files on the command line.
This option causes the compiler to create one or more copies of itself, each in a separate process. These copies simultaneously compile the source files.

If \( n \) is not specified for this option, the default value is 2.

This option applies to compilations, but not to linking or link-time code generation.

**IDE Equivalent**

None

**Alternate Options**

None

**nologo**

*Tells the compiler to not display compiler version information.*

**Syntax**

**Linux OS:**

None

**Windows OS:**

\(/\text{nologo}\)

**Arguments**

None

**Default**

OFF

**Description**

Tells the compiler to not display compiler version information.

**IDE Equivalent**

**Windows**

Visual Studio: *General > Suppress Startup Banner*

**Linux**

Eclipse: None

**Alternate Options**

None

**save-temps, Qsave-temps**

*Tells the compiler to save intermediate files created during compilation.*

**Syntax**

**Linux OS:**

\(-\text{save-temps}\)
-no-save-temps

**Windows OS:**
/Qsave-temps (C++ only)
/Qsave-temps- (C++ only)

**Windows OS:**
None (DPC++ only)

**Arguments**
None

**Default**

<table>
<thead>
<tr>
<th>DPC++: Linux* systems: -no-save-temps</th>
<th>C++: Linux* systems: -no-save-temps</th>
<th>Windows* systems: .obj files are saved</th>
</tr>
</thead>
</table>

On Linux systems, the compiler deletes intermediate files after compilation is completed.

<table>
<thead>
<tr>
<th>-no-save-temps</th>
<th>C++: Linux* systems: -no-save-temps</th>
<th>Windows* systems: .obj files are saved</th>
</tr>
</thead>
</table>

On Linux systems, the compiler deletes intermediate files after compilation is completed.

On Windows systems, the compiler saves only intermediate object files after compilation is completed.

**Description**

This option tells the compiler to save intermediate files created during compilation. The names of the files saved are based on the name of the source file; the files are saved in the current working directory.

If option [Q]save-temps (C++) or save-temps (DPC++) is specified, the following occurs:

- The object .o file (Linux) is saved.
- C++: The .obj file (Windows) object .o file is saved.

If -no-save-temps is specified on Linux systems, the following occurs:

- The .o file is put into /tmp and deleted after calling ld.
- The preprocessed file is not saved after it has been used by the compiler.

This content is specific to C++; it does not apply to DPC++.

If /Qsave-temps- is specified on Windows systems, the following occurs:

- The .obj file is not saved after the linker step.
- The preprocessed file is not saved after it has been used by the compiler.

**NOTE**

This option only saves intermediate files that are normally created during compilation.

**IDE Equivalent**
None

**Alternate Options**
None

**showIncludes**

*Tells the compiler to display a list of the include files.*
**Syntax**

**Linux OS:**
None

**Windows OS:**
/showIncludes

**Arguments**
None

**Default**
OFF  The compiler does not display a list of the include files.

**Description**
This option tells the compiler to display a list of the include files. Nested include files (files that are included from the files that you include) are also displayed.

**IDE Equivalent**

**Windows**
Visual Studio: Advanced > Show Includes

**Linux**
Eclipse: None

**Alternate Options**
None

sox, Qsox
*Tell the compiler to save the compilation options in the executable file.*

**Syntax**

**Linux OS:**
-sox

**Windows OS:**
/Qsox

**Arguments**
None

**Default**
OFF  The compiler version number is saved in the object file.

**Description**
This option tells the compiler to save the compilation options in the executable file. The information is embedded as a string in each object file or assembly output.
When you specify this option, the size of the executable on disk is increased slightly. When you link the object files into an executable file, the linker places each of the information strings into the header of the executable. It is then possible to use a tool, such as a strings utility, to determine what options were used to build the executable file.

**IDE Equivalent**

None

**Alternate Options**

None

**sysroot**

Specifies the root directory where headers and libraries are located.

**Syntax**

**Linux OS:**

--sysroot=dir

**Windows OS:**

None

**Arguments**

*dir*  

Specifies the local directory that contains copies of target libraries in the corresponding subdirectories.

**Default**

Off  

The compiler uses default settings to search for headers and libraries.

**Description**

This option specifies the root directory where headers and libraries are located.

For example, if the headers and libraries are normally located in `/usr/include` and `/usr/lib` respectively, `--sysroot=/mydir` will cause the compiler to search in `/mydir/usr/include` and `/mydir/usr/lib` for the headers and libraries.

This option is provided for compatibility with gcc.

**NOTE**

Even though this option is not supported for a Windows-to-Windows native compiler, it is supported for a Windows-host to Linux-target compiler.

**IDE Equivalent**

None

**Alternate Options**

None
**Tc**
*Tells the compiler to process a file as a C source file.*

**Syntax**

**Linux OS:**
None

**Windows OS:**
/Tc filename

**Arguments**

*filename*  
Is the file name to be processed as a C source file.

**Default**

OFF  
The compiler uses default rules for determining whether a file is a C source file.

**Description**

This option tells the compiler to process a file as a C source file.

**IDE Equivalent**

None

**Alternate Options**

None

**See Also**

TC  
compiler option

Tp  
compiler option

**TC**
*Tells the compiler to process all source or unrecognized file types as C source files.*

**Syntax**

**Linux OS:**
None

**Windows OS:**
/TC

**Arguments**

None

**Default**

OFF  
The compiler uses default rules for determining whether a file is a C source file.
**Description**  
This option tells the compiler to process all source or unrecognized file types as C source files.

**IDE Equivalent**  
**Windows**  
Visual Studio: Advanced > Compile As

**Linux**  
Eclipse: None

**Alternate Options**  
None

**See Also**  
TP compiler option  
Tc compiler option

**Tp**  
*Tells the compiler to process a file as a C++ source file.*

**Syntax**  
**Linux OS:**  
None

**Windows OS:**  
/Tp filename

**Arguments**  
filename  
Is the file name to be processed as a C++ source file.

**Default**  
OFF  
The compiler uses default rules for determining whether a file is a C++ source file.

**Description**  
This option tells the compiler to process a file as a C++ source file.

**IDE Equivalent**  
None

**Alternate Options**  
None

**See Also**  
TP compiler option  
Tc compiler option
version
Tells the compiler to display GCC-style version information.

Syntax
Linux OS:
--version

Windows OS:
None

Arguments
None

Default
OFF

Description
Tells the compiler to display GCC-style version information.

IDE Equivalent
None

Alternate Options
None

watch
Tells the compiler to display certain information to the console output window. This content is specific to C++; it does not apply to DPC++.

Syntax
Linux OS:
-watched="keyword[, keyword...]"
-wnowledge

Windows OS:
/watch[="keyword[, keyword...]"
/nowatch

Arguments
keyword Determines what information is displayed. Possible values are:

- none Disables cmd and source.
- [no]cmd Determines whether driver tool commands are displayed and executed.
- [no]source Determines whether the name of the file being compiled is displayed.
Enables cmd and source.

**Default**

nowatch  Pass information and source file names are not displayed to the console output window.

**Description**

Tells the compiler to display processing information (pass information and source file names) to the console output window.

<table>
<thead>
<tr>
<th>Option watch</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Tells the compiler to not display pass information and source file names to the console output window. This is the same as specifying nowatch.</td>
</tr>
<tr>
<td>cmd</td>
<td>Tells the compiler to display and execute driver tool commands.</td>
</tr>
<tr>
<td>source</td>
<td>Tells the compiler to display the name of the file being compiled.</td>
</tr>
<tr>
<td>all</td>
<td>Tells the compiler to display pass information and source file names to the console output window. This is the same as specifying watch with no keyword. For heterogeneous compilation, the tool commands for the host and the offload compilations will be displayed.</td>
</tr>
</tbody>
</table>

**IDE Equivalent**

None

**Alternate Options**

watch cmd  

Linux: -v  
Windows: None

**See Also**

v  compiler option

**Alternate Compiler Options**

This content is specific to C++; it does not apply to DPC++.

This topic lists alternate names for compiler options and show the primary option name. Some of the alternate option names are deprecated and may be removed in future releases.

For more information on compiler options, see the detailed descriptions of the individual, primary options. Some of these options are deprecated. For more information, see Deprecated and Removed Options.

<table>
<thead>
<tr>
<th>Alternate Linux* Options</th>
<th>Primary Option Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Generation:</td>
<td></td>
</tr>
<tr>
<td>-fp</td>
<td>-fomit-frame-pointer</td>
</tr>
<tr>
<td>Advanced Optimizations:</td>
<td></td>
</tr>
<tr>
<td>-funroll-loops</td>
<td>-unroll</td>
</tr>
</tbody>
</table>
### Alternate Linux* Options

<table>
<thead>
<tr>
<th>OpenMP* and Parallel Processing Options:</th>
<th>Primary Option Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fopenmp</td>
<td>-qopenmp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linking or Linker:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-i-dynamic</td>
<td>-shared-intel</td>
</tr>
<tr>
<td>-i-static</td>
<td>-static-intel</td>
</tr>
</tbody>
</table>

### Alternate Windows* Options

<table>
<thead>
<tr>
<th>OpenMP* and Parallel Processing Options:</th>
<th>Primary Option Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>/openmp</td>
<td>/Qopenmp</td>
</tr>
</tbody>
</table>

### Related Options

This section discusses portability options and GCC*-compatible warning options.

### Portability Options

This content is specific to C++; it does not apply to DPC++.

A challenge in porting applications from one compiler to another is making sure there is support for the compiler options you use to build your application. The Intel® oneAPI DPC++/C++ Compiler supports many of the options that are valid on other compilers you may be using.

The following sections list compiler options that are supported by the Intel® oneAPI DPC++/C++ Compiler and the following:

- Microsoft* C++ Compiler
- GCC* Compiler

Options that are unique to either compiler are not listed in these sections.

### Options Equivalent to Microsoft C++ Options (Windows*)

The following table lists compiler options that are supported by both the Intel® oneAPI DPC++/C++ Compiler and the Microsoft C++ Compiler.

For complete details about these options, for example, the possible values for `<n>` when it appears below, see the Microsoft Visual Studio C++ documentation.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/C</td>
<td></td>
</tr>
<tr>
<td>/c</td>
<td></td>
</tr>
<tr>
<td>/D{name}&gt;{</td>
<td>}={</td>
</tr>
<tr>
<td>/E</td>
<td></td>
</tr>
<tr>
<td>/EH{a</td>
<td>s</td>
</tr>
<tr>
<td>/EP</td>
<td></td>
</tr>
<tr>
<td>/F&lt;n&gt;</td>
<td></td>
</tr>
<tr>
<td>/Fa[file]</td>
<td></td>
</tr>
</tbody>
</table>
/FA[(c|s|cs)]
/FC
/Fe<file>
/FI<file>
/Fm[<file>]
/Fo<file>
/fp:<model>
/Fp<file>
/FR[<file>]
/GA
/Gd
/GF
/Gr
/GR[-]
/GS[-]
/Gs[<n>]
/Gy[-]
/Gz
/GZ
/H<n>
/help
/I<dir>
/J
/LD
/LDd
/link
/MD
/MDd
/MT
/MTd
/nologo
/O1
/O2
/Od
/Oi[-]
/ Os
/ Ot
/ Ox
/Oy[-]
/P
/QIfist[-]
/RTC{1|c|s|u}
/showIncludes
/ TC
/TC<source file>
/TP
/Tp<source file>
/u
/U<name>
/vd<n>
/vmg
/vmv
/W<n>
/Wall
/WX
/X
/Y-
/Yc[<file>]
/Yu[<file>]
/Z7
/Zc:<arg1>[, <arg2>]
/Zg
Options Equivalent to GCC* Options (Linux*)

The following table lists compiler options that are supported by both the Intel® oneAPI DPC++/C++ Compiler and the GCC Compiler.

<table>
<thead>
<tr>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ansi</td>
</tr>
<tr>
<td>-B</td>
</tr>
<tr>
<td>-C</td>
</tr>
<tr>
<td>-c</td>
</tr>
<tr>
<td>-D</td>
</tr>
<tr>
<td>-dD</td>
</tr>
<tr>
<td>-dM</td>
</tr>
<tr>
<td>-E</td>
</tr>
<tr>
<td>-fargument-noalias</td>
</tr>
<tr>
<td>-fargument-noalias-global</td>
</tr>
<tr>
<td>-fcf-protection</td>
</tr>
<tr>
<td>-fdata-sections</td>
</tr>
<tr>
<td>-ffunction-sections</td>
</tr>
<tr>
<td>-f[no-]builtin</td>
</tr>
<tr>
<td>-f[no-]common</td>
</tr>
<tr>
<td>-f[no-]freestanding</td>
</tr>
<tr>
<td>-f[no-]gnu-keywords</td>
</tr>
<tr>
<td>-f[no-]inline</td>
</tr>
<tr>
<td>-f[no-]inline-functions</td>
</tr>
<tr>
<td>-f[no-]math-errno</td>
</tr>
<tr>
<td>-f[no-]operator-names</td>
</tr>
<tr>
<td>-f[no-]stack-protector</td>
</tr>
<tr>
<td>-f[no-]unsigned-bitfields</td>
</tr>
</tbody>
</table>
-fpack-struct
-fpermissive
-fPIC
-fpic
-freg-struct-return
-fshort-enums
-fsyntax-only
-funroll-loops
-funsigned-char
-fverbose-asm
-H
-help
-I
-idirafter
-imacros
-iprefix
-iwithprefix
-iwithprefixbefore
-l
-L
-M
-malign-double
-march
-mcpu
-MD
-MF
-MG
-MM
-MMD
-m[no-]ieee-fp
-mp
-MQ
-msse
-msse2
-msse3
-MT
-nodefaultlibs
-nostartfiles
-nostdinc
-nostdinc++
-nostdlib
-0
-O
-O0
-O1
-O2
-O3
-Os
-p
-P
-S
-shared
-static
-std
-trigraphs
-U
-u
-v
-V
-Wall
-Werror
-W[no-]cast-qual
 GCC*-Compatible Warning Options

This content is specific to C++; it does not apply to DPC++.

The Intel® oneAPI DPC++/C++ Compiler recognizes many GCC*-compatible warning options, but we do not document all of them.

In general, if a GCC-compatible option is accepted by the compiler, but not documented, the implementation of the option is the same as described in the GCC documentation.

To find the GCC documentation about GCC warning options, you can do any of the following:

• Check the GCC website (http://gcc.gnu.org/onlinedocs/gcc/)
• On Linux*, enter the command man gcc
• Search the web for "gcc warning options"
Floating-Point Operations

This section contains information about floating-point operations, including IEEE floating-point operations, and it provides guidelines that can help you improve the performance of floating-point applications.

Understanding Floating-Point Operations

Floating-point specific operations provide several compiler options that allow you to tune your applications based on specific objectives. This section will guide you through the purpose and use of the floating-point operations.

Programming Tradeoffs in Floating-Point Applications

In general, the programming objectives for floating-point applications fall into the following categories:

- **Accuracy**: The application produces results that are close to the correct result.
- **Reproducibility and portability**: The application produces consistent results across different runs, different sets of build options, different compilers, different platforms, and different architectures.
- **Performance**: The application produces fast, efficient code.

Based on the goal of an application, you will need to make tradeoffs among these objectives. For example, if you are developing a 3D graphics engine, performance may be the most important factor to consider, with reproducibility and accuracy as secondary concerns.

The compiler provides several options that allow you to tune your applications based on specific objectives. Broadly speaking, there are the floating-point specific options, such as the `-fp-model` (Linux*) or `/fp` (Windows*) option, and the fast-but-low-accuracy options, such as the `[Q]imf-max-error` option. The compiler optimizes and generates code differently when you specify these different compiler options. Select appropriate compiler options by carefully balancing your programming objectives and making tradeoffs among these objectives. Some of these options may influence the choice of math routines that are invoked.

Many routines in the `libirc`, `libm`, and `svml` library are more highly optimized for Intel microprocessors than for non-Intel microprocessors.

Using Floating-Point Options

Take the following code as an example:

```
float t0, t1, t2;
...
t0=t1+t2+4.0f+0.1f;
```

If you specify the `-fp-model extended` (Linux) or `/fp:extended` (Windows) option in favor of accuracy, the compiler generates the following assembly code:

```
fld       DWORD PTR _t1
fadd      DWORD PTR _t2
fadd      DWORD PTR _Cnst4.0
fadd      DWORD PTR _Cnst0.1
fstp      DWORD PTR _t0
```

This code maximizes accuracy because it utilizes the highest mantissa precision available on the target platform. The code performance might suffer when managing the x87 stack, and it might yield results that cannot be reproduced on other platforms that do not have an equivalent extended precision type.
If you specify the -fp-model source (Linux) or /fp:source (Windows) option in favor of reproducibility and portability, the compiler generates the following assembly code:

```
movss xmm0, DWORD PTR _t1
addss xmm0, DWORD PTR _t2
addss xmm0, DWORD PTR _Cnst4.0
addss xmm0, DWORD PTR _Cnst0.1
movss DWORD PTR _t0, xmm0
```

This code maximizes portability by preserving the original order of the computation, and by using the IEEE single-precision type for all computations. It is not as accurate as the previous implementation, because the intermediate rounding error is greater compared to extended precision. It is not the highest performance implementation, because it does not take advantage of the opportunity to pre-compute 4.0f + 0.1f.

If you specify the -fp-model fast (Linux) or /fp:fast (Windows) option in favor of performance, the compiler generates the following assembly code:

```
movss xmm0, DWORD PTR _Cnst4.1
addss xmm0, DWORD PTR _t1
addss xmm0, DWORD PTR _t2
movss DWORD PTR _t0, xmm0
```

This code maximizes performance using Intel® Streaming SIMD Extensions (Intel® SSE) instructions and pre-computing 4.0f + 0.1f. It is not as accurate as the first implementation, due to the greater intermediate rounding error. It does not provide reproducible results like the second implementation, because it must reorder the addition to pre-compute 4.0f + 0.1f. All compilers, on all platforms, at all optimization levels do not reorder the addition in the same way.

For many other applications, the considerations may be more complicated.

**Using Fast-But-Low-Accuracy Options**

The fast-but-low-accuracy options provide an easy way to control the accuracy of mathematical functions and utilize performance/accuracy tradeoffs offered by the Intel® oneAPI Math Kernel Library (oneMKL). You can specify accuracy, via a command line interface, for all math functions or a selected set of math functions at the level more precise than low, medium, or high.

You specify the accuracy requirements as a set of function attributes that the compiler uses for selecting an appropriate function implementation in the math libraries. Examples using the attribute, max-error, are presented here. For example, use the following option to specify the relative error of two ULPs for all single, double, long double, and quad precision functions:

```
-fimf-max-error=2
```

To specify twelve bits of accuracy for a sin function, use:

```
-fimf-accuracy-bits=12:sin
```

To specify relative error of ten ULPs for a sin function, and four ULPs for other math functions called in the source file you are compiling, use:

```
-fimf-max-error=10:sin-fimf-max-error=4
```

On Windows systems, the compiler defines the default value for the max-error attribute depending on the /fp option settings. In /fp:fast mode the compiler sets a max-error=4.0 for the call. Otherwise, it sets a max-error=0.6.
Dispatching of Math Routines

The compiler optimizes calls to routines from the libm and svml libraries into direct CPU-specific calls, when the compilation configuration specifies the target CPU where the code is tuned, and if the set of instructions available for the code compilation is not narrower than the set of instructions available in the tuning target CPU.

The dispatching optimization applies to the exp() routine, and to the other math routines with CPU specific implementations in the libraries. The dispatching optimization can be disabled using the -fimf-force-dynamic-target (or Qimf-force-dynamic-target) option. This option specifies a list of math routines that are improved with a dynamic dispatcher.

See Also
Using -fp-model(/fp) Options
fimf-max-error, Qimf-max-error compiler option

Using the -fp-model (/fp) Option

The -fp-model (Linux*) or /fp (Windows*) option allows you to control the optimizations on floating-point data. You can use this option to tune the performance, level of accuracy, or result consistency for floating-point applications across platforms and optimization levels.

NOTE The -fpmodel (/fp) option is only available for C++; it is not available for DPC++.

You can use keywords to specify the semantics to be used. The keywords specified for this option may influence the choice of math routines that are invoked. Many routines in the libirc, libm, and libsvml libraries are more highly optimized for Intel microprocessors than for non-Intel microprocessors. Possible values of the keywords are as follows:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>precise</td>
<td>Enables value-safe optimizations on floating-point data.</td>
</tr>
<tr>
<td>fast</td>
<td>Enables more aggressive optimizations on floating-point data.</td>
</tr>
<tr>
<td>double</td>
<td>Rounds intermediate results to 53-bit (double) precision and enables value-safe optimizations.</td>
</tr>
</tbody>
</table>

NOTE Using the default option keyword -fp-model fast or /fp:fast, you may get significant differences in your result depending on whether the compiler uses x87 or SSE/AVX instructions to implement floating-point operations. Results are more consistent when the other option keywords are used.

See Also
fp-model, fp compiler option

Denormal Numbers

A normalized number is a number for which both the exponent (including bias) and the most significant bit of the mantissa are non-zero. For such numbers, all the bits of the mantissa contribute to the precision of the representation.
The smallest normalized single-precision floating-point number greater than zero is about 1.1754943\(^{-38}\). Smaller numbers are possible, but those numbers must be represented with a zero exponent and a mantissa whose leading bit(s) are zero, which leads to a loss of precision. These numbers are called denormalized numbers or denormals (newer specifications refer to these as subnormal numbers).

Denormal computations use hardware and/or operating system resources to handle denormals; these can cost hundreds of clock cycles. Denormal computations take much longer to calculate than normal computations.

There are several ways to avoid denormals and increase the performance of your application:

- Scale the values into the normalized range.
- Use a higher precision data type with a larger range.
- Flush denormals to zero.

See Also

Reducing Impact of Denormal Exceptions

Intel® 64 and IA-32 (for C++ only) Architectures Software Developer's Manual, Volume 1: Basic Architecture

Institute of Electrical and Electronics Engineers, Inc*. (IEEE) web site for information about the current floating-point standards and recommendations

Setting the FTZ and DAZ Flags

In Intel® processors, the flush-to-zero (FTZ) and denormals-are-zero (DAZ) flags in the MXCSR register are used to control floating-point calculations. Intel® Streaming SIMD Extensions (Intel® SSE) and Intel® Advanced Vector Extensions (Intel® AVX) instructions, including scalar and vector instructions, benefit from enabling the FTZ and DAZ flags. Floating-point computations using the Intel® SSE and Intel® AVX instructions are accelerated when the FTZ and DAZ flags are enabled. This improves the application's performance.

Manually set the flags with the following macros:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable FTZ</td>
<td>_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON)</td>
</tr>
<tr>
<td>Enable DAZ</td>
<td>_MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON)</td>
</tr>
</tbody>
</table>

The prototypes for these macros are in xmmintrin.h (FTZ) and pmmintrin.h (DAZ).

Overview: Tuning Performance

This section describes several programming guidelines that can help you improve the performance of floating-point applications:

- Avoid exceeding representable ranges during computation; handling these cases can have a performance impact.
- Use a single-precision type (for example, float) unless the extra precision and/or range obtained through double or long double is required. Greater precision types increase memory size and bandwidth requirements. See Using Efficient Data Types section.
- Reduce the impact of denormal exceptions for all supported architectures.
- Avoid mixed data type arithmetic expressions.

See Also

Avoiding Mixed Data Type Arithmetic Expressions
Reducing the Impact of Denormal Exceptions
Using Efficient Data Types

Handling Floating-point Array Operations in a Loop Body

Following the guidelines below will help auto-vectorization of the loop.
Statements within the loop body may contain float or double operations (typically on arrays). The following arithmetic operations are supported: addition, subtraction, multiplication, division, negation, square root, MAX, MIN, and mathematical functions such as SIN and COS.

Writing to a single-precision scalar/array and a double scalar/array within the same loop decreases the chance of auto-vectorization due to the differences in the vector length (that is, the number of elements in the vector register) between float and double types. If auto-vectorization fails, try to avoid using mixed data types.

NOTE
The special __m64, __m128, and __m256 datatypes are not vectorizable. The loop body cannot contain any function calls. Use of the Intel® Streaming SIMD Extensions (Intel® SSE) and Intel® Advanced Vector Extensions (Intel® AVX) intrinsics (for example, mm_add_ps) is not allowed.

See Also
Programming Guidelines for Vectorization

Reducing the Impact of Denormal Exceptions

Denormalized floating-point values are those that are too small to be represented in the normal manner; that is, the mantissa cannot be left-justified. Denormal values require hardware or operating system interventions to handle the computation, so floating-point computations that result in denormal values may have an adverse impact on performance.

There are several ways to handle denormals to increase the performance of your application:

- Scale the values into the normalized range
- Use a higher precision data type with a larger range
- Flush denormals to zero

For example, you can translate them to normalized numbers by multiplying them using a large scalar number, doing the remaining computations in the normal space, then scaling back down to the denormal range. Consider using this method when the small denormal values benefit the program design.

Consider using a higher precision data type with a larger range; for example, by converting variables declared as float to be declared as double. Understand that making the change can potentially slow down your program. Storage requirements will increase, which will increase the amount of time for loading and storing data from memory. Higher precision data types can also decrease the potential throughput of Intel® Streaming SIMD Extensions (Intel® SSE) and Intel® Advanced Vector Extensions (Intel® AVX) operations.

If you change the type declaration of a variable, you might also need to change associated library calls, unless these are generic; for example, cos() instead of cosf(). You should verify that the gain in performance from eliminating denormals is greater than the overhead of using a data type with higher precision and greater dynamic range.

In many cases, denormal numbers can be treated safely as zero without adverse effects on program results. Depending on the target architecture, use flush-to-zero (FTZ) options.

See Also
Setting the FTZ and DAZ Flags

Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture

Avoiding Mixed Data Type Arithmetic Expressions

Avoid mixing integer and floating-point (float, double, or long double) data in the same computation. Expressing all numbers in a floating-point arithmetic expression (assignment statement) as floating-point values eliminates the need to convert data between fixed and floating-point formats. Expressing all numbers in an integer arithmetic expression as integer values also achieves this. This improves run-time performance.
For example, assuming that \( I \) and \( J \) are both int variables, expressing a constant number (2.0) as an integer value (2) eliminates the need to convert the data. The following examples demonstrate inefficient and efficient code.

**Examples**

<table>
<thead>
<tr>
<th>Inefficient Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int I, J;</code></td>
</tr>
<tr>
<td><code>I = J / 2.0;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Efficient Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int I, J;</code></td>
</tr>
<tr>
<td><code>I = J / 2;</code></td>
</tr>
</tbody>
</table>

**Using Efficient Data Types**

In cases where more than one data type can be used for a variable, consider selecting the data types based on the following hierarchy, listed from most to least efficient:

- `char`
- `short`
- `int`
- `long`
- `long long`
- `float`
- `double`
- `long double`

**NOTE**

In an arithmetic expression, you should avoid mixing integer and floating-point data.

You can use integer data types (\( int, int\ long\ ), etc.) in loops to improve floating point performance. Convert the data type to integer data types, process the data, then convert the data to the old type.

**Understanding IEEE Floating-Point Operations**

**Understanding the IEEE Standard for Floating-Point Arithmetic, IEEE 754-2008**

This version of the compiler uses a close approximation to the IEEE Standard for Floating-Point Arithmetic, version IEEE 754-2008, unless otherwise stated. This standard is common to many microcomputer-based systems due to the availability of fast processors that implement the required characteristics.

This section outlines the characteristics of the IEEE 754-2008 standard and its implementation in the compiler. Except as noted, the description refers to both the IEEE 754-2008 standard and the compiler implementation.

**Special Values**

This is a list and a brief description of the special values that the Intel® oneAPI DPC++/C++ Compiler supports.
Signed Zero

The sign of zero is the same as the sign of a nonzero number. Comparisons consider +0 to be equal to -0. A signed zero is useful in certain numerical analysis algorithms, but in most applications the sign of zero is invisible.

Denormalized Numbers

Denormalized numbers (denormals) fill the gap between the smallest positive and the smallest negative normalized number, otherwise only (+/-) 0 occurs in the interval. Denormalized numbers extend the range of computable results by allowing for gradual underflow.

This content is specific to C++; it does not apply to DPC++. Systems based on the IA-32 architecture support a Denormal Operand status flag. When this is set, at least one of the input operands to a floating-point operation is a denormal. The Underflow status flag is set when a number loses precision and becomes a denormal.

Signed Infinity

Infinities are the result of arithmetic in the limiting case of operands with arbitrarily large magnitude. They provide a way to continue when an overflow occurs. The sign of an infinity is simply the sign you obtain for a finite number in the same operation as the finite number approaches an infinite value.

By retrieving the status flags, you can differentiate between an infinity that results from an overflow and one that results from division by zero. The compiler treats infinity as signed by default. The output value of infinity is +Infinity or -Infinity.

Not a Number

Not a Number (NaN) may result from an invalid operation. For example, 0/0 and SQRT(-1) result in NaN. In general, an operation involving a NaN produces another NaN. Because the fraction of a NaN is unspecified, there are many possible NaNs.

The compiler treats all NaNs identically, but there are two classes of NaNs:

- **Signaling NaNs**: Have an initial mantissa bit of 0. They usually raise an invalid exception when used in an operation.
- **Quiet NaNs**: Have an initial mantissa bit of 1.

The floating-point hardware usually converts a signaling NaN into a quiet NaN during computational operations. An invalid exception is raised and the resulting floating-point value is a quiet NaN.

Attributes

Attributes are a way to provide additional information about a declaration to the compiler. The C++11 attribute syntax is consistent with the C2x standard.

Using Attributes

The compiler supports three ways to add attributes to your program:

- **Gnu Syntax**
  
  `__attribute__((attribute_name(arguments)))`

- **Microsoft Syntax**
  
  `__declspec(attribute_name(argument))`
- **C++11 Standardized Attribute Syntax** (part of the C++11 language standard)

  ```
  [[attribute_name(arguments)]]
  [[attribute-namespace :: attribute_name(arguments)]]
  ```

  Some attributes are available for both Intel® microprocessors and non-Intel microprocessors but they may perform additional optimizations for Intel® microprocessors than they perform for non-Intel microprocessors. Refer to the individual attribute name for a detailed description.

### align

*Directs the compiler to align the variable to a specified boundary and a specified offset.*

**Syntax**

**Windows® OS:**

```
__declspec(align(n))
```

**Linux® OS:**

```
__attribute__((aligned(n)))
```

For portability on Linux OS, you should use the syntax form `__attribute__((aligned(n)))`. This form is compatible with the GNU compiler.

**Arguments**

- `n` Specifies the alignment. The compiler will align the variable to an `n`-byte boundary.

**Description**

This keyword directs the compiler to align the variable to an `n`-byte boundary.

**NOTE**

If you require 8-byte alignment, we recommend you specify 16 for `n`, instead of 8. When 8 is used, the compiler interprets the value as a suggestion and you may not get the requested 8-byte alignment, depending on various heuristics.

### align_value

*Provides the ability to add a pointer alignment value to a pointer typedef declaration.*

**Syntax**

**Windows® OS:**

```
__declspec(align_value(alignment))
```

**Linux® OS:**

```
__attribute__((align_value(alignment)))
```

**Arguments**

- `alignment` Specifies the alignment (8, 16, 32, 64, 128, 256,...) for what the pointer points to.
Description
This keyword can be added to a pointer typedef declaration to specify the alignment value of pointers declared for that pointer type.

It tells the compiler that the data referenced by the designated pointer is aligned by the indicated value, and the compiler can generate code based on that assumption. If this attribute is used incorrectly, and the data is not aligned to the designated value, the behavior is undefined.

allow_cpu_features
Provides the ability for a function to use intrinsic functions and architecture specific functionality.

Syntax
Windows OS:
__declspec(allow_cpu_features(featp1[, featp2]))

Linux OS:
__attribute__((allow_cpu_features(featp1[, featp2])))

Arguments
featp1 Specifies features to allow for the function. Values are integral constant expressions that evaluate to the page one bitmask of permissible features from the libirc CPUID information. The evaluated type is an unsigned 64-bit integer which permits use of template-dependent code. Possible values are:

- _FEATURE_GENERIC_IA32
- _FEATURE_FPU
- _FEATURE_CMOV
- _FEATURE_MMX
- _FEATURE_FXSAVE
- _FEATURE_SSE
- _FEATURE_SSE2
- _FEATURE_SSE3
- _FEATURE_SSSE3
- _FEATURE_SSE4_1
- _FEATURE_SSE4_2
- _FEATURE_MOVBE
- _FEATURE_POPCNT
- _FEATURE_PCLMULQDQ
- _FEATURE_AES
- _FEATURE_F16C
- _FEATURE_AVX
- _FEATURE_RDRND
- _FEATURE_FMA
- _FEATURE_BMI
- _FEATURE_LZCNT
- _FEATURE_HLE
- _FEATURE_RTM
- _FEATURE_AVX2
Optional. Specifies features to allow for the function. Values are integral constant expressions that evaluate to the page two bitmask of permissible features from the libirc CPUID information. The evaluated type is an unsigned 64-bit integer which permits use of template-dependent code. If only features from page two are desired, specify 0 for featp1. Possible values are:

- _FEATURE_CLDEMOTE
- _FEATURE_MOVDIRI
- _FEATURE_MOVDIR64B
- _FEATURE_WAITPKG
- _FEATURE_AVX512_Bf16
- _FEATURE_ENQCMD
- _FEATURE_AVX_VNNI
- _FEATURE_AMX_TILE
- _FEATURE_AMX_INT8
- _FEATURE_AMX_BF16
- _FEATURE_KL
- _FEATURE_WIDE_KL
Description
This keyword can be added to a function to specify intrinsic functions and architecture specific functionality that the function is allowed to use. The function is generated as if the specified features are available.

**concurrency_safe**
*Guides the compiler to parallelize more loops and straight-line code.*

Syntax
Windows* OS:
__declspec(concurrency_safe(clause))

Linux* OS:
__attribute__((concurrency_safe(clause)))

Arguments

clause

Is one of the following:

- **cost(cycles)**: Specifies the execution cycles of the annotated function for the compiler to perform parallelization profitability analysis while compiling its enclosing loops or blocks. The value of cycles is a 2-byte unsigned integer (unsigned short); its maximal value is $2^{16}-1$. If the cycle count is greater than $2^{16}-1$, you should use **profitable**.

- **profitable**: Specifies that the loops or blocks that contain calls to the annotated function are profitable to parallelize.

Description
This keyword specifies that there are no incorrect side-effects and no illegal (or improperly synchronized) memory access interferences among multiple invocations of the annotated function or between an invocation of this annotated function and other statements in the program, if they are executed concurrently.

For every function that is marked with this keyword, you must ensure that its side effects (if any) are acceptable (or expected), and the memory access interferences are properly synchronized.

**const**

*Indicates that a function has no effect other than returning a value and that it uses only its arguments to generate that return value.*

Syntax
Windows* OS:
__declspec(const)

Linux* OS:
__attribute__((const))

Arguments

None
Description
This keyword is equivalent to the gcc\* attribute const and applies to function declarations.

cpu_dispatch, cpu_specific

Provides the ability to write one or more versions of a function that execute only on a list of targeted processors (cpu_dispatch). Provides the ability to declare that a version of a function is targeted at particular types of processors (cpu_specific).

Syntax

Windows\* OS:
__declspec(cpu_dispatch(cpuid, cpuid, ...))
__declspec(cpu_specific(cpuid))

Linux\* OS:
__attribute__((cpu_dispatch(cpuid, cpuid, ...)))
__attribute__((cpu_specific(cpuid)))

Arguments

**cpuid**

Possible values are:
- *atom*: Intel\® Atom™ processors with Intel\® Supplemental Streaming SIMD Extensions 3 (Intel\® SSSE3)
- *atom_sse4_2*: Intel\® Atom™ processors with Intel\® Streaming SIMD Extensions 4.2 (Intel\® SSE4.2)
- *atom_sse4_2_movbe*: Intel\® Atom™ processors with Intel\® Streaming SIMD Extensions 4.2 (Intel\® SSE4.2) with MOVBE instructions enabled
- *broadwell*: This is a synonym for core_5th_gen_avx
- *core_2nd_gen_avx*: 2nd generation Intel\® Core™ processor family with support for Intel\® Advanced Vector Extensions (Intel\® AVX)
- *core_3rd_gen_avx*: 3rd generation Intel\® Core™ processor family with support for Intel\® Advanced Vector Extensions (Intel\® AVX) including the RDRND instruction
- *core_4th_gen_avx*: 4th generation Intel\® Core™ processor family with support for Intel\® Advanced Vector Extensions 2 (Intel\® AVX2) including the RDRND instruction
- *core_4th_gen_avx_tsx*: 4th generation Intel\® Core™ processor family with support for Intel\® Advanced Vector Extensions 2 (Intel\® AVX2) including the RDRND instruction, and support for Intel\® Transactional Synchronization Extensions (Intel\® TSX)
- *core_5th_gen_avx*: 5th generation Intel\® Core™ processor family with support for Intel\® Advanced Vector Extensions 2 (Intel\® AVX2) including the RDSEED and Multi-Precision Add-Carry Instruction Extensions (ADX) instructions
core_5th_gen_avx_tsx: 5th generation Intel® Core™ processor family with support for Intel® Advanced Vector Extensions 2 (Intel® AVX2) including the RDSEED and Multi-Precision Add-Carry Instruction Extensions (ADX) instructions, and support for Intel® Transactional Synchronization Extensions (Intel® TSX)

core_aes_pclmulqdq: Intel® Core™ processors with support for Advanced Encryption Standard (AES) instructions and carry-less multiplication instruction

core_i7_sse4_2: Intel® Core™ i7 processors with Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) instructions

generic: Other Intel processors for IA-32 (for C++ only) or Intel® 64 architecture or compatible processors not provided by Intel Corporation

haswell: This is a synonym for core_4th_gen_avx

pentium: Intel® Pentium® processor

pentium_4: Intel® Pentium® 4 processors

pentium_4_sse3: Intel® Pentium® 4 processor with Intel® Streaming SIMD Extensions 3 (Intel® SSE3) instructions, Intel® Core™ Duo processors, Intel® Core™ Solo processors

pentium_ii: Intel® Pentium® II processors

pentium_iii: Intel® Pentium® III processors

pentium_iii_no_xmm_regs: Intel® Pentium® III processors with no XMM registers

pentium_m: Intel® Pentium® M processors

pentium_mmx: Intel® Pentium® processors with MMX™ technology

pentium_pro: Intel® Pentium® Pro processors

**Description**

Use the `cpu_dispatch` keyword to provide a list of targeted processors, along with an empty function body/function stub.

Use the `cpu_specific` keyword to declare each function version targeted at particular type of processor.

These features are available only for Intel processors based on IA-32 (for C++ only) or Intel® 64 architecture. They are not available for non-Intel processors. Applications built using the manual processor dispatch feature may be more highly optimized for Intel processors than for non-Intel processors.

**See Also**

mpx

Directs the compiler to pass Intel® Memory Protection Extensions (Intel® MPX) bounds information along with any pointer-typed parameters.
Syntax
Windows* OS:
__declspec(mpx)
Linux* OS:
__attribute__((mpx))

Arguments
None

Description
When a function declared with this keyword is called, any pointer-typed parameters passed to the function will also have Intel® MPX bounds information passed. If the called function returns a pointer-typed object, the compiler will expect the function to return Intel® MPX bounds information along with the pointer object. Similarly, if this keyword is applied to a function definition, the function will expect the caller to pass Intel® MPX bounds information along with any pointer-type parameters. If the function returns a pointer-typed object, Intel® MPX bounds information will be returned with the object.

NOTE
The usage of this attribute is intended for Windows code that contains hand-written Intel® MPX enhancements based on Intel® MPX inline assembly or calls to Intel® MPX intrinsics, and where the user does not wish to enable automatic Intel® MPX code generation.

Intrinsics

A detailed introduction and information about Intel intrinsics is provided in the Intel® C++ Compiler Classic Developer Guide and Reference. The Intel® Intrinsics Guide provides detailed information and a lookup tool for viewing the available Intel intrinsics.

The following is some general information:

• Intrinsics are assembly-coded functions that let you use C++ function calls and variables in place of assembly instructions.
• Intrinsics can be used only on the host.
• Intrinsics are expanded inline eliminating function call overhead. Providing the same benefit as using inline assembly, intrinsics improve code readability, assist instruction scheduling, and help reduce debugging.
• Intrinsics provide access to instructions that cannot be generated using the standard constructs of the C and C++ languages.

NOTE
To use intrinsic-based code with the Intel® oneAPI DPC++/C++ Compiler, do the following:

• Specify compiler option `march` so that the compiler recognizes the processor-specific or architecture-specific intrinsic.
• Include the immintrin.h header file that comes with the intrinsic declarations.
Availability of Intrinsics on Intel Processors

Not all Intel® processors support all intrinsics. For information on which intrinsics are supported on Intel® processors, visit the Product Specification, Processors page. The Processor Spec Finder tool links directly to all processor documentation and the datasheets list the features, including intrinsics, supported by each processor.

Libraries

The Intel® oneAPI DPC++/C++ Compiler lets you use all the standard run-time libraries that are part of Microsoft* Visual C++*. The options described in this section can help you determine which libraries your application uses.

To create libraries, use the lib.exe tool or xilib.exe tool.

Creating Libraries

Libraries are simply an indexed collection of object files that are included as needed in a linked program. Combining object files into a library makes it easy to distribute your code without disclosing the source. It also reduces the number of command-line entries needed to compile your project.

Static Libraries

Executables generated using static libraries are no different than executables generated from individual source or object files. Static libraries are not required at runtime, so you do not need to include them when you distribute your executable. At compile time, linking to a static library is generally faster than linking to individual source files.

To build a static library on Linux*:

1. Use the c option to generate object files from the source files:

   [invocation] -c my_source1.cpp my_source2.cpp my_source3.cpp

   Where the [invocation] is icpx for C++, or dpcpp for DPC++.

2. Use the GNU* tool ar to create the library file from the object files:

   ar rc my_lib.a my_source1.o my_source2.o my_source3.o

3. Compile and link your project with your new library:

   [invocation] main.cpp my_lib.a

   Where the [invocation] is icpx for C++, or dpcpp for DPC++.

If your library file and source files are in different directories, use the Ldir option to indicate where your library is located:

[invocation] -L/cpp/libs main.cpp my_lib.a

Where the [invocation] is icpx for C++, or dpcpp for DPC++.

If you are using Interprocedural Optimization, see the topic on Creating a Library from IPO Objects using xiar.
**Shared Libraries**

Shared libraries, also referred to as dynamic libraries or Dynamic Shared Objects (DSO), are linked differently than static libraries. At compile time, the linker insures that all the necessary symbols are either linked into the executable, or can be linked at runtime from the shared library. Executables compiled from shared libraries are smaller, but the shared libraries must be included with the executable to function correctly. When multiple programs use the same shared library, only one copy of the library is required in memory.

To build a shared library on Linux:

1. **Use options** `-fPIC` and `-c` to generate object files from the source files:

   ```sh
   [invocation] -fPIC -c my_source1.cpp my_source2.cpp my_source3.cpp
   ```
   
   Where the `[invocation]` is `icpx` for C++, or `dpcpp` for DPC++.

2. **Use the `shared` option to create the library file from the object files**:

   ```sh
   [invocation] -shared -o my_lib.so my_source1.o my_source2.o my_source3.o
   ```
   
   Where the `[invocation]` is `icpx` for C++, or `dpcpp` for DPC++.

3. **Compile and link your project with your new library**:

   ```sh
   [invocation] main.cpp my_lib.so
   ```
   
   Where the `[invocation]` is `icpx` for C++, or `dpcpp` for DPC++.

Use the following options to create libraries on Windows*:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/LD</code>, <code>/Ldd</code></td>
<td>Produces a DLL. <code>d</code> indicates debug version.</td>
</tr>
<tr>
<td><code>/MD</code>, <code>/Mdd</code></td>
<td>Compiles and links with the dynamic, multi-thread C run time library. <code>d</code> indicates debug version.</td>
</tr>
<tr>
<td><code>/MT</code>, <code>/MTd</code></td>
<td>Compiles and links with the static, multi-thread C run time library. <code>d</code> indicates debug version.</td>
</tr>
<tr>
<td><code>/Zl</code></td>
<td>Disables embedding default libraries in object files.</td>
</tr>
</tbody>
</table>

See Also

Using Intel Shared Libraries

See Also

`/LD` compiler option
`/MD` compiler option
`/MT` compiler option

**Using Intel Shared Libraries**

This topic applies to Linux*.

This content is specific to C++; it does not apply to DPC++.

By default, the Intel® oneAPI DPC++/C++ Compiler links Intel® C++ libraries dynamically. The GNU*/Linux* system libraries are also linked dynamically.
Options for Shared Libraries (Linux*)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-shared-intel</td>
<td>Use the shared-intel option to link Intel®-provided libraries dynamically. This has the advantage of reducing the size of the application binary, but it also requires the libraries to be on the application’s target system.</td>
</tr>
<tr>
<td>-shared</td>
<td>The shared option instructs the compiler to build a Dynamic Shared Object (DSO) instead of an executable. For more details, refer to the ld man page documentation.</td>
</tr>
<tr>
<td>-fpic</td>
<td>Use the fpic option when building shared libraries. It is required for the compilation of each object file included in the shared library.</td>
</tr>
</tbody>
</table>

Managing Libraries

Managing Libraries on Linux*

During compilation, the compiler reads the LIBRARY_PATH environment variable for static libraries it needs to link when building the executable. At runtime, the executable will link against dynamic libraries referenced in the LD_LIBRARY_PATH environment variable.

Modifying LIBRARY_PATH

If you want to add a directory, /libs for example, to the LIBRARY_PATH, you can do either of the following:

- Command line: prompt> export LIBRARY_PATH=/libs:$LIBRARY_PATH
- Startup file: export LIBRARY_PATH=/libs:$LIBRARY_PATH

To compile file.cpp and link it with the library mylib.a, enter the following command:

```bash
//on Linux
[invocation] file.cpp mylib.a
```

With [invocation] being icpx or dpcpp.

The compiler passes file names to the linker in the following order:

1. To the object file.
2. To any objects or libraries specified at the command line, in a response file, or in a configuration file.
3. To the Intel® Math Library, libimf.a.

By default, the Intel® oneAPI DPC++/C++ Compiler uses the GNU* implementation of the C++ Standard Library (libstdc++) on OS* X v10.8, and libc++ implementation on OS* X v10.9. You can change the default using the -stdlib option:

```bash
-stdlib=libc++    //to switch to libc++
-stdlib=libstdc++ //to switch to libstdc++
```

Compile with DPC++ and Link Other Compilers

When you use the dpcpp compiler and source its entire environment, then linking works correctly with other compilers if the correct path to the compiler libraries is set. This allows programs to be compiled with DPC++ and then linked with other compilers (example: gcc). If you try to do this without sourcing the compiler environment, the linking fails with undefined references in libsyscl.so and other internal libraries.
To resolve this, add the following paths to `LD_LIBRARY_PATH`:

```bash
<install_dir>/compiler/latest/linux/compiler/lib/intel64
<install_dir>/compiler/latest/linux/lib
<install_dir>/compiler/latest/linux/lib/x64
<install_dir>/tbb/latest/lib/intel64/gcc4.8
```

Managing Libraries on Windows*

The `LIB` environment variable contains a semicolon-separated list of directories in which the Microsoft* linker will search for library (.lib) files. The compiler does not specify library names to the linker, but includes directives in the object file to specify the libraries to be linked with each object.

For more information on adding library names to the response file and the configuration file, see Using Response Files and Using Configuration Files.

To specify a library name on the command line, you must first add the library’s path to the `LIB` environment variable. Then, to compile `file.cpp` and link it with the library `mylib.lib`, enter the following command:

```
[invocation] file.cpp mylib.lib
```

Where `[invocation]` is `icx` for C++ or `dpcpp-cl` for DPC++.

Other Considerations

The Intel Compiler Math Libraries contain performance-optimized implementations for various Intel platforms. By default, the best implementation for the underlying hardware is selected at runtime. The library dispatch of multi-threaded code may lead to apparent data races, which may be detected by certain software analysis tools. However, as long as the threads are running on cores with the same CPUID, these data races are harmless and are not a cause for concern.

Redistributing Libraries When Deploying Applications

When you deploy your application to systems that do not have a compiler installed, you need to redistribute certain Intel® libraries where your application is linked. You can do so in one of the following ways:

- **Statically link your application.**
  
  An application built with statically-linked libraries eliminates the need to distribute runtime libraries with the application executable. By linking the application to the static libraries, you are not dependent on the Intel® Fortran or Intel® C/C++ dynamic shared libraries.

- **Dynamically link your application.**

  If you must build your application with dynamically linked (or shared) compiler libraries, you should address the following concerns:

  - You must build your application with shared or dynamic libraries that are redistributable.
  - Pay careful attention to the directory where the redistributables are installed and how the OS finds them.
  - You should determine which shared or dynamic libraries your application needs.

The information here is only introductory. The redistributable library installation packages are available at the following locations:

- **Intel® oneAPI versions**
- **Older Intel® Parallel Studio XE versions**
Resolving References to Shared Libraries Provided with Intel® oneAPI

If you are relying on shared libraries distributed with Intel® oneAPI tools, you must make sure that your users have these shared libraries on their systems.

If you are building an application that will be deployed to your user community and you are relying on shared libraries (.so shared objects on Linux®, .dll dynamic libraries on Windows®) distributed with Intel® oneAPI tools, you must make sure that your users have these shared libraries on their systems. You can determine what shared libraries you depend on by doing the following for your program and components:

Linux

Use the `ldconfig` command.

Windows

Use the `dumpbin /DEPENDENTS programOrComponentName` command.

Once you have done this, you must choose how your users will receive these libraries.

Shared Library Deployment

Once you have built, run, and debugged your application, you must deploy it to your users. That deployment includes any shared libraries, including libraries that are components of the Intel® oneAPI toolkits.

Deployment Models

You have two options for deploying the shared libraries from the Intel oneAPI toolkit that your application depends on:

Private Model

Copy the shared libraries from the Intel oneAPI toolkit into your application environment, and then package and deploy them with your application. Review the license and third-party files associated with the Intel oneAPI toolkits and/or components you have installed to determine the files that you can redistribute.

The advantage to this model is that you have control over your library and version choice, so you only package and deploy the libraries that you have tested. The disadvantage is that the end users may see multiple libraries installed on their system, if multiple installed applications all use the private model. You are also responsible for updating these libraries whenever updates are required.

Public Model

You direct your users to runtime packages provided by Intel. Your users install these packages on their system when they install your application. The run-time packages install onto a fixed location, so all applications built with Intel oneAPI tools can be used.

The advantage is that one copy of each library is shared by all applications, which results in improved performance. You also can rely on updates to the run-time packages to resolve issues with libraries independently from when you update your application. The disadvantage is that the footprint of the run-time package is larger than a package from the private model. Another disadvantage is that your tested versions of the run-time libraries may not be the same as your end user's versions.

Select the model that best fits your environment, your needs, and the needs of your users.
NOTE Intel ensures that newer compiler-support libraries work with older versions of generated compiler objects, but newer versioned objects require newer versioned compiler-support libraries. If an incompatibility is introduced that causes newer compiler-support libraries not to work with older compilers, you will have sufficient warning and the library will be versioned so that deployed applications continue to work.

Additional Steps
Under either model, you must manually configure certain environment variables that are normally handled by the setvars/vars scripts or modulefiles.

For example, with the Intel® MPI Library, you must set the following environment variables during installation:

Linux

\[
\text{I\_MPI\_ROOT=installPath FI\_PROVIDER\_PATH=installPath/intel64/}
\]

libfabric:/usr/lib64/libfabric

Windows

\[
\text{I\_MPI\_ROOT=installPath}
\]

Compatibility in the Minor Releases of the Intel oneAPI Products

For Intel oneAPI products, each minor version of the product is compatible with the other minor version from the same release (for example, 2021). When there are breaking changes in API or ABI, the major version is increased. For example, if you tested your application with an Intel oneAPI product with a 2021.1 version, it will work with all 2021.x versions. It is not guaranteed that it will work with 2022.x or 19.x versions.

Intel’s Memory Allocator Library

Intel’s libqkmalloc library for fast memory allocation provides a C-level interface for memory allocation that is optimized for performance.

You can link the libqkmalloc library as a shared library only on Linux® platforms for Intel® 64 architecture. This library provides optimized implementation of standard allocation routines malloc, calloc, realloc, and free, and is C99 standard compliant.

NOTE This library is limited to work only on Intel® processors and will redirect to standard C routines at runtime if used on non-Intel® processors.

Using Intel’s Custom Memory Allocator Library

You can use the libqkmalloc library by linking directly to it or by using the LD_PRELOAD environment variable.

To ensure the application will override the standard library allocation routines with libqkmalloc, set the environment variable LD_PRELOAD in the command line before the application execution. This environment variable allows you to set the path of the library that will be loaded before any other library (including the C runtime library), and the application will use symbols from this specified library instead of the symbols from the standard library.

Restrictions

This library does not support threaded code such as OpenMP® and is not thread-safe. It should not be used simultaneously from multiple threads. For the best results this library should be used with large throughput workloads.
Introduction to the SIMD Data Layout Templates

SIMD Data Layout Templates (SDLT) is a C++11 template library providing containers that represent arrays of "Plain Old Data" objects (a struct whose data members do not have any pointers/references and no virtual functions) using layouts that enable generation of efficient SIMD (single instruction multiple data) vector code. SDLT uses standard ISO C++11 code. It does not require a special language or compiler to be functional, but takes advantage of performance features (such as OpenMP* SIMD extensions and `#pragma ivdep`) that may not be available to all compilers. It is designed to promote scalable SIMD vector programming. To use the library, specify SIMD loops and data layouts using explicit vector programming model and SDLT containers, and let the compiler generate efficient SIMD code in an efficient manner.

Many of the library interfaces employ generic programming, in which interfaces are defined by requirements on types and not specific types. The C++ Standard Template Library (STL) is an example of generic programming. Generic programming enables SDLT to be flexible yet efficient. The generic interfaces enable you to customize components to your specific needs.

The net result is that SDLT enables you to specify a preferred SIMD data layout far more conveniently than re-structuring your code completely with a new data structure for effective vectorization, and at the same time can improve performance.

Motivation

C++ programs often represent an algorithm in terms of high level objects. For many algorithms there is a set of data that the algorithm will need to process. It is common for the data set to be represented as array of "plain old data" objects. It is also common for developers to represent that array with a container from the C++ Standard Template Library, like `std::vector`. For example:

```cpp
struct Point3s
{
    float x;
    float y;
    float z;
    // helper methods
};

std::vector<Point3s> inputDataSet(count);
std::vector<Point3s> outputDataSet(count);

for(int i=0; i < count; ++i) {
    Point3s inputElement = inputDataSet[i];
    Point3s result = // transformation of inputElement that is independent of other iterations
        // can keep algorithm high level using object helper methods
    outputDataSet[i] = result;
}
```
When possible a compiler may attempt to vectorize the loop above, however the overhead of loading the "Array of Structures" data set into vector registers may overcome any performance gain of vectorizing. Programs exhibiting the scenario above could be good candidates to use a SDLT container with a SIMD-friendly internal memory layout. SDLT containers provide accessor objects to import and export Primitives between the underlying memory layout and the objects original representation. For example:

```cpp
SDLT_PRIMITIVE(Point3s, x, y, z)

sdlt::soa1d_container<Point3s> inputDataSet(count);
sdlt::soa1d_container<Point3s> outputDataSet(count);

auto inputData = inputDataSet.const_access();
auto outputData = outputDataSet.access();

#pragma forceinline recursive
#pragma omp simd
for(int i=0; i < count; ++i) {
    Point3s inputElement = inputData[i];
    Point3s result = // transformation of inputElement that is independent of other iterations
                     // can keep algorithm high level using object helper methods
    outputData[i] = result;
}
```

When a local variable inside the loop is imported from or exported to using that loop's index, the compiler's vectorizer can now access the underlying SIMD friendly data format and when possible perform unit stride loads. If the compiler can prove nothing outside the loop can access the loop's local object, then it can optimize its private representation of the loop object be "Structure of Arrays" (SOA). In our example, the container's underlying memory layout is also SOA and unit stride loads can be generated. The Container also allocates aligned memory and its accessor objects provide the compiler with the correct alignment information for it to optimize code generation accordingly.

**Version Information**

This documentation is for SDLT version 2, which extends version 1 by introducing support for n-dimensional containers.

**Backwards Compatibility**

Public interfaces of version 2 are fully backward compatible with interfaces of version 1.

The backwards compatibility includes:

- **Existing source code compatibility.**
  - Any source code using the SDLT v1 public API (non-internal interfaces) can be recompiled against SDLT v2 headers with no changes.

- **Binary compatibility.**
  - Because SDLT v2 API's exist in a new name space, sdlt::v2, all ABI linkage should not collide with any existing SDLT v1 ABI's that exist only in sdlt namespace.
  - A binary, dynamically-linked library that uses SDLT v1 internally, can be linked into a program using SDLT v2, and vice versa.
  - Passing SDLT containers or accessors as part of a libraries public API (ABI). When SDLT is used as part of an ABI, that library and the calling code must use the same version of SDLT. They cannot be mixed or matched.

This compatibility doesn't cover internal implementation. Internal implementation for SDLT v1 was updated and unified with parts introduced in v2, so for codes dependent on internal interfaces backwards compatibility is not guaranteed.
**Deprecated**

This content is specific to C++; it does not apply to DPC++.

The interfaces below are deprecated; use the replacements provided in the table.

<table>
<thead>
<tr>
<th>Deprecated Interface</th>
<th>Deprecated in Version</th>
<th>Replaced By</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdl::fixed_offset&lt;&gt;</td>
<td>v2</td>
<td>sdl::fixed&lt;&gt;</td>
</tr>
<tr>
<td>sdl::aligned_offset&lt;&gt;</td>
<td>v2</td>
<td>sdl::aligned&lt;&gt;</td>
</tr>
</tbody>
</table>

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Notice revision #20201201

**Usage Guidelines: Function Calls and Containers**

**Function Calls**

Function calls are a commonly used programming construct. Follow these simple guidelines when using SDLT containers:

- If an SDLT Primitive is passed to a function by value, by pointer, or by reference, be sure to inline them.
- Any Non-inlined functions should be SIMD enabled (for example, denote them with `#pragma omp declare simd`).

If a loop variable is passed to a non-inlined function, the current C++ Application Binary Interface (ABI) requires the memory layout match object's original which could cause additional data transformations or inhibit vectorization. For that reason, the SDLT approach works best when all the methods or functions called are inlined or use `#pragma omp declare simd`. Marking a function "inline" explicitly or implicitly is only a hint. Compilers have several limits and heuristics that could cause a function to not be inlined. To avoid this issue, we recommend utilizing the `#pragma forceinline` which instructs the compiler to ignore its limits and heuristics: causing all functions in the following code block that could be inlined to actually be inlined together with any functions called, and functions they call, and so on. Please also note that this can cause the loop body and/or the function body to become too big to optimize. Under such circumstances, carefully examine and restructure the function call boundaries and consider applying non-inlined, SIMD-enabled function calls.

**1-Dimensional Containers Overview**

What if that `std::vector<typename>` could store data SIMD-friendly format internally while exposing an AOS view to the programmer?

The 1-dimensional containers in SDLT aim to achieve that goal. They can abstract the in-memory data layout of an array of objects to:

1. AOS (Array of Structures)
2. SOA (Structure of Arrays) which is SIMD friendly

**Import/Export Only**

As the memory layout is abstracted and may not match the original structure's layout, containers cannot provide memory references to the underlying data. Only import or export of the object to and from a particular element in the container. In use, an algorithm might require some minor code changes to follow import/export paradigm, however algorithm itself should read/flow the same.
The 1D containers in SDLT are dynamically resizable with an interface similar to `std::vector<T>`. To avoid accidental misuse of copying containers into C++11 lambda functions we chose to delete the container’s copy constructor and instead provide explicit “clone” method instead.

Containers provide SDLT concepts of an accessor and const_accessor for use with SIMD loops, interfaces for `std::vector` compatibility are intended for ease of integration, not high performance.

Just like `std::vector`, the containers own the array data and its scope controls the life of that data.

### n-Dimensional Containers Overview

Multi-dimensional containers generalize ideas from 1-dimensional containers; they separate multi-dimensional access semantics from storage logic in an abstract way. A multi-dimensional SDLT container is a generic container that handles an arbitrary number of dimensions, and at the same time internally represents data as needed. Unlike 1-dimensional containers, multi-dimensional containers are not resizable and don't have interfaces like that of `std::vector`. While 1-dimensional containers are like `std::vectors` with decoupled storage, multi-dimensional containers are more akin to arrays (statically sized or variable length).

Below is an example of an n-dimensional container parameterized by three concerns: the data item (primitive) type, the storage layout in memory, and the observed shape of the container.

~~~
 n_container<PrimitiveT, LayoutT, ExtentsT>

<table>
<thead>
<tr>
<th>Template Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typename PrimitiveT</td>
<td>The type of primitive that will be contained.</td>
</tr>
<tr>
<td>typename LayoutT</td>
<td>The type of data layout.</td>
</tr>
<tr>
<td>typename ExtentsT</td>
<td>Specifies the dimensions of the container</td>
</tr>
</tbody>
</table>

### Product and Performance Information

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

### Constructing an n_container

**Description**

An N-dimensional (multi-dimensional) container must be constructed before it can be used. The data type to be contained must first be declared as a `SDLT_PRIMITIVE`, then a data layout is chosen, and finally the shape of the container is determined describing the extents of each dimension.

### Specifying Data Layout

Rather than defining different containers for different data layouts, the data layout to use is specified as a template parameter to the container.

Available layouts are summarized in table below. Full details can found on the table in the topic n_container.

~~~
<table>
<thead>
<tr>
<th>Layout</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layout::soa&lt;&gt;</td>
<td>Structure of Arrays (SOA). Each data member of the Primitive will have its own N-dimensional array.</td>
</tr>
<tr>
<td>layout::soa_per_row&lt;&gt;</td>
<td>Structure of Arrays Per Row. Each data member of the Primitive will have its own 1-dimensional array per row. Layout repeats for remaining N-1 dimensions.</td>
</tr>
</tbody>
</table>
### Layout

<table>
<thead>
<tr>
<th>Layout</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layout::aos_by_struct</td>
<td>Array of Structures (AOS) Accessed by Struct. Native AOS layout and data access.</td>
</tr>
<tr>
<td>layout::aos_by_stride</td>
<td>Array of Structures Accessed by Stride. Native SOA data access through pointers to the built in types of members using a stride to account for the size of the Primitive.</td>
</tr>
</tbody>
</table>

### Numbers and Constants

In order to define shape, integer values can be provided in three different forms, each successively providing less information to compiler. It is advised to use as precise specification as possible. The compiler may optimize better with more information.

<table>
<thead>
<tr>
<th>Integer Value Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed&lt;int NumberT&gt;</td>
<td>Known at compile time.</td>
</tr>
<tr>
<td></td>
<td>foo(fixed&lt;1080&gt;(), fixed&lt;1920&gt;());</td>
</tr>
<tr>
<td></td>
<td>The suffix _fixed will declare an equivalent literal.</td>
</tr>
<tr>
<td></td>
<td>For example, (1080_fixed is equivalent to fixed&lt;1080&gt;.</td>
</tr>
<tr>
<td></td>
<td>foo(1080_fixed, 1920_fixed); )</td>
</tr>
<tr>
<td>aligned&lt;int AlignmentT&gt;(number)</td>
<td>Programmer guarantees the number is a multiple of the AlignmentT.</td>
</tr>
<tr>
<td></td>
<td>foo(aligned&lt;8&gt;(height), aligned&lt;128&gt;(width));</td>
</tr>
<tr>
<td>“int”</td>
<td>Arbitrary integer value.</td>
</tr>
<tr>
<td></td>
<td>foo(width, height);</td>
</tr>
</tbody>
</table>

### Specifying Container Shape

n_extent_t<..> is a variadic template that accepts any number of arguments defining dimensions. Because construction using this type may look unclear, a generator object, n_extent, is provided to construct extents for all dimensions using a familiar array-definition-like syntax. Extent values may be specified using the most precise representation possible, as described above, to allow the compiler to better prove any potential data alignments.

```
    n_extent[height][width]; // OK
    n_extent[height][aligned<128>(width)]; // Better
    n_extent[1080_fixed][1920_fixed]; // Best
```

### Defining an n_container

Using a previously declared primitive (same as SDLT v1),

```
struct RGBAs { float red, green, blue, alpha; };
SDLT_PRIMITIVE(RGBAs, red, green, blue, alpha)
```

A two-dimensional container of RGBAs with HD image size 1920x0180 can be declared and instantiated as in the below example.

```
typedef n_container<RGBAs, layout::soa, |
    n_extent_t<fixed<1080>, fixed<1920>>> HdImage;
HdImage image1;```
If sizes are not known, a container may be defined with extents unknown to the compiler but known at run-
time when an instance of the container is created.

```cpp
typedef n_container<RGBAs, layout::soa, n_extent_t<int, int>> Image;
Image image2(n_extent[height][width]);
```

Additionally, the templated factory function make_n_container<PrimitiveT, LayoutT> may be used to create
containers.

```cpp
auto image1 = make_n_container<RGBAs,
  layout::soa>(n_extent[1080_fixed][1920_fixed]);
auto image2 = make_n_container<RGBAs,
  layout::soa>(n_extent[height][width]);
```

### Accessing Cells

Containers own data. To get to the data inside, use an "accessor."

```cpp
auto ca = image1.const_access();
auto a = image2.access();
```

Specify the index for each dimension with a series of calls to the array subscript operator [], similar to a
multi-dimensional array in C.

```cpp
RGBAs pixel = ca[y][x];
float greyscale = (pixel.red + pixel.green + pixel.blue)/3;
a[y][x] = RGBAs(greyscale, greyscale, greyscale);
```

### Discovering Extents

Accessors know their extents.

Use template function extent_d<int DimensionT>(object).

```cpp
for (int y = 0; y < extent_d<0>(ca); ++y)
  for (int x = 0; x < extent_d<1>(ca); ++x) {
    RGBAs pixel = ca[y][x];
    // …
  }
```

For convenience, non-template methods are also provided.

```cpp
for (int y = 0; y < ca.extent_d0(); ++y)
  for (int x = 0; x < ca.extent_d1(); ++x) {
    RGBAs pixel = ca[y][x];
    // …
  }
```

### Lowering Dimensions

The result of not specifying all the dimensions required by an accessor is a new accessor with a lower rank
that can then be accessed.

```cpp
auto cay = ca[y];
RGBAs pixel = cay[x];
```

### Bounds

#### Description

`bounds_t<LowerT, UpperT>` holds the lower and upper bounds of a half-open interval. It is templated to
allow the different integer representations for the lower and upper bounds. The intent is to model a valid
iteration space over a single dimension.

Bounds can be used to iterate over an entire extent or to restrict iteration space within an extent.
Creating Bounds

Bounds can be created using full bounds_t type, but this may be tedious.

```cpp
bounds_t<int, int>(start, finish)
bounds_t<int, aligned<16>>(start, aligned<16>(finish))
bounds_t<fixed<0>, fixed<1920>>(())
```

It is simpler and clearer to use factory function bounds to build a bounds_t<>.

```cpp
bounds(start, finish);
bounds(start, aligned<16>(finish));
bounds(0_fixed, 1920_fixed)
```

Discovering Bounds

Accessors know their valid iteration space. Initial bounds for an accessor are set to set the lower bound to be fixed<0> and the upper bound set to the value and type of the dimension's extent as specified during construction of the n_container(fixed<>, aligned<>, or int).

To query bounds for given dimension of the accessor use template function `bounds_d<int DimensionT>(object)`.

```cpp
auto b0 = bounds_d<0>(ca);
auto b1 = bounds_d<1>(ca);
for (int y = b0.lower(); y < b0.upper(); ++y)
  for (int x = b1.lower(); x < b1.upper(); ++x) {
    RGBAs pixel = ca[y][x];
    // …
  }
```

bounds_t can participate in C++11 range-based for loops.

```cpp
for (auto y: bounds_d<0>(ca))
  for (auto x: bounds_d<1>(ca)) {
    RGBAs pixel = ca[y][x];
    // …
  }
```

```cpp
for (auto y: ca.bounds_d0())
  for (auto x: ca.bounds_d1()) {
    RGBAs pixel = ca[y][x];
    // …
  }
```

N-Dimensional Indexes and Bounds

To model index and bounds values over multiple dimensions, respectively the following template classes are provided: `n_index_t<...>` and `n_bounds_t<...>`. These are both variadic templates, accepting any number of arguments.

`n_index` is a generator to simplify creating instances of `n_index_t`.

```cpp
n_index[540][960]
```

`n_bounds` is a generator to simplify creating instances of `n_bounds_t`.

```cpp
n_bounds[bounds(540,1080)][bounds(960,1920)]
```

Alternatively, `n_bounds_t` can be defined in terms of a `n_index_t` and `n_extent_t`.

```cpp
n_bounds(n_index[540][960], n_extent[540][960]);
```

Accessing Subsections
From a container's accessors, a new accessor can be created over a subsection defined by a `n_bounds_t`.

```cpp
auto ca = c.const_access();
auto subsect = ca.section(n_bounds[540, 1080][960, 1920]);
```

The effect is to restrict the results of `bounds_d<int Dimension>` on the subsection accessor.

You can create a new accessor translated to a different index space.

```cpp
auto offsetnewSpace = ca.translated_to(n_index[1000][2000]);
auto zeroSpace = ca.translated_to_zero();
```

Accesses will have a translation applied that maps the `n_index` back to the lower bounds of the accessor that created it. This allows a smaller container to be reused in a larger index space that is being walked over by blocks, or to move a subsection index space back to the origin.

**User-Level Interface**

This section describes the user-level interface for the SIMD Data Layout Templates (SDLT). This API is defined in `sdlt.h` and its associated header files.

**SDLT Primitives (SDLT_PRIMITIVE)**

Primitives represent the data we want to work over in SIMD. They can be more than just data structures. As a C++ object, it can have its own methods that modify its data.

Rules:

- Must be Plain Old Data (POD)
  - Has trivial copy constructor
  - Has trivial move constructor
  - Has trivial destructor
  - No virtual functions or virtual bases
- No reference data members
- No unions
- No bit fields
- No bool types
  - Comparison semantics not efficient in SIMD
  - Use 32-bit integer and compare against known values like 0 or 1 explicitly
- Data members need to be public or declare `SDLT_PRIMITIVE_FRIEND` in the object's definition

Current limitations:

- No pointer data members
- No C++11 strongly typed enums—use integers instead.
- No array based data members.
- copy constructor and assignment operator (=) defined by individual member assignment—strongly encouraged to facilitate better code generation

They may seem like large restrictions, but often code can easily be re-factored to meet this requirement. For example:

```cpp
class Point3d {
    // methods...
protected:
    double v[3];
};
```
can be re-factored to have a public data member for each element in the array and update methods to use the \( x \), \( y \), and \( z \) data members rather than the array \( v \).

```cpp
class Point3d {
public:
    // methods...
    double x;
    double y;
    double z;
};
```

For better code generation, explicitly define a copy constructor and assignment operator (=) by individual member assignment.

**SDLT_PRIMITIVE** Macro

Once an object meets the criteria above, we can consider it a Primitive type in SDLT. In order for Container's to import and export the Primitive, it has to understand its data layout. Unfortunately C++11 lacks compile time reflection, so the user must provide SDLT with a description of your structure's data layout. This is easily done with the SDLT_PRIMITIVE helper macro that accepts a struct type followed by a comma separated list of its data members.

```
SDLT_PRIMITIVE(STRUCT_NAME, DATA_MEMBER_1, ...)
```

**Example Usage:**

```cpp
struct UserObject
{
    float x;
    float y;
    double acceleration;
    int behavior;
};

SDLT_PRIMITIVE(UserObject, x, y, acceleration, behavior)
```

An object must be declared as a Primitive before it can be used in a Container. However, built-in types like float, double, int, etc. do not need to be declared as a Primitive before use with a Container. Built-in's are automatically considered Primitives by SDLT.

Nested Primitives are supported, but the nested Primitive must be declared before the outer Primitive is.

**Example:** Axis Aligned Bounding Box made up of two 3d points

```cpp
struct Point3s
{
    float x;
    float y;
    float z;
};

struct AABB
{
    Point3s topLeft;
    Point3s bottomRight;
};

SDLT_PRIMITIVE(Point3s, x, y, z)
SDLT_PRIMITIVE(AABB, topLeft, bottomRight)
```

Notice the `struct` definitions themselves do not derive from SDLT or use any of its nomenclature. This independence allows classes to be used in code not using SDLT and only code that does use SDLT Containers needs to see the Primitive declarations.
soa1d_container
Template class for "Structure of Arrays" memory layout of a one-dimensional container of Primitives.
#include <sdlt/soa1d_container.h>

Syntax

```
template<typename PrimitiveT,
    int AlignD1OnIndexT = 0,
    class AllocatorT = allocator::default_alloc>
class soa1d_container;
```

Arguments

- **typename PrimitiveT**
  The type that each element in the array will store

- **int AlignD1OnIndexT = 0**
  [Optional] The index on which the data access will be aligned (useful for stencils)

- **class AllocatorT = allocator::default_alloc**
  [Optional] Specify type of allocator to be used. allocator::default_alloc is currently the only allocator supported.

Description

Dynamically sized container of Primitive elements with memory layout as a Structure of Arrays internally providing:

- Dynamic resizing with interface similar to std::vector
- Accessor objects suitable for efficient data access inside SIMD loops

### Member

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef size_t size_type;</td>
<td>Type to use when specifying sizes to methods of the container.</td>
</tr>
<tr>
<td>template&lt;typename OffsetT = no_offset&gt; using accessor;</td>
<td>Template alias to an accessor for this container</td>
</tr>
<tr>
<td>template&lt;typename OffsetT = no_offset &gt; using const_accessor;</td>
<td>Template alias to an const_accessor for this container</td>
</tr>
</tbody>
</table>

### Member Type

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| soa1d_container(
    size_type size_d1 = 0u,
    buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0),
    const allocator_type & an_allocator = allocator_type()); | Constructs an uninitialized container of size_d1 elements, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing. |
| soa1d_container(
    size_type size_d1,
    const PrimitiveT &a_value,
    buffer_offset_in_cachelines buffer_offset | Constructs a container of size_d1 elements initializing each with a_value, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing. |
<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| = buffer_offset_in_cachelines(0),
 const allocator_type & an_allocator = allocator_type(); | Constructs a container with a copy of each of the elements in other, in the same order, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing. |
| template<typename StlAllocatorT>
soa1d_container(
    const std::vector<PrimitiveT>,
    StlAllocatorT & other,
    buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0),
    const allocator_type & an_allocator = allocator_type(); | |
| soa1d_container(
    const PrimitiveT * other_array,
    size_type number_of_elements,
    buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0),
    const allocator_type & an_allocator = allocator_type(); | Constructs a container with a copy of number_of_elements elements from the array other_array, in the same order, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing. |
| template< typename IteratorT >
soa1d_container(
    IteratorT a_begin,
    IteratorT an_end,
    buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0),
    const allocator_type & an_allocator = allocator_type(); | Constructs a container with as many elements as the range [a_begin - an_end), each with a copy of the value from its corresponding element in that range, in the same order, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing. |
| soa1d_container clone() const; | Returns: a new soa1d_container instance with its own copy of the elements |
| void resize(size_type new_size_d1); | Resize the container so that it contains new_size_d1 elements. If the new size is greater than the current container size, the new elements are unitialized. |
| accessor<> access(); | Returns: accessor with no embedded index offset. |
| accessor<int> access(int offset); | Returns: accessor with an integer based embedded index offset. |
| accessor<aligned_offset<IndexAlignmentT> > access(aligned_offset<IndexAlignmentT>); | Returns: accessor with an aligned_offset<IndexAlignmentT> based embedded index offset. |
| accessor<fixed_offset<OffsetT> > access(fixed_offset<OffsetT>); | Returns: accessor with a fixed_offset<OffsetT> based embedded index offset. |
| const_accessor<> const_access() const; | Returns: const_accessor with no embedded index offset. |
### Member Type

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const_accessor&lt;int&gt;</td>
<td>Returns: const_accessor with an integer based embedded index offset.</td>
</tr>
<tr>
<td>const_access(int offset) const;</td>
<td></td>
</tr>
<tr>
<td>const_accessor&lt;aligned_offset&lt;IndexAlignmentT&gt;&gt;</td>
<td>Returns: const_accessor with an aligned_offset&lt;IndexAlignmentT&gt; based</td>
</tr>
<tr>
<td>const_access(aligned_offset&lt;IndexAlignmentT&gt; offset) const;</td>
<td>embedded index offset.</td>
</tr>
<tr>
<td>template&lt;int OffsetT&gt;</td>
<td>Returns: const_accessor with a fixed_offset&lt;OffsetT&gt; based embedded</td>
</tr>
<tr>
<td>const_accessor&lt;fixed_offset&lt;OffsetT&gt; &gt; const;</td>
<td>index offset.</td>
</tr>
</tbody>
</table>

### STL Compatibility

In addition to the performance oriented interface explained in the table above, `soa1d_container` implements a subset of the `std::vector` interface that is intended for ease of integration, not high performance. Due to the import/export only requirement we can’t return a reference to the object, instead `iterators` and `operator[]` return a Proxy object while other "const" methods return a "value_type const". Furthermore, iterators do not support the `->` operator. Despite that limitation the iterators can be passed to any STL algorithm. Also for performance reasons, resize does not initialize new elements. The following `std::vector` interface methods are implemented:

- size, max_size, capacity, empty, reserve, shrink_to_fit
- assign, push_back, pop_back, clear, insert, emplace, erase
- cbegin, cend, begin, end, begin, end, crbegin, crend, rbegin, rend, rbeg
- operator[], front() const, back() const, at() const
- swap, ==, !=
- swap, soa1d_container(soa1d_container&& donor), soa1d_container & operator=(soa1d_container&& donor)

### `aos1d_container`

*Template class for "Array of Structures" memory layout of a one-dimensional container of Primitives.*

```
#include <sdl/t/aos1d_container.h>
```

### Syntax

```
template<
    typename PrimitiveT,
    AccessBy AccessByT,
    class AllocatorT = allocator::default_alloc
>
class aos1d_container;
```

### Arguments

- `typename PrimitiveT`  
The type that each element in the array will store
- `access_by AccessByT`  
  Enum to control how the memory layout will be accessed. Recommend `access_by_struct` unless you are having issues vectorizing. See the documentation of `access_by` for more details
class AllocatorT = allocator::default_alloc

[Optional] Specify the type of allocator to be used. allocator::default_alloc is currently the only allocator supported.

Description
Provide compatible interface with soald_container while keeping the memory layout as an Array of Structures internally. User can easily switch between data layouts by changing the type of container they use. The rest of the code written against accessors and proxy elements and members can stay the same.

- Dynamic resizing with interface similar to std::vector
- Accessor objects suitable for efficient data access inside SIMD loops

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef size_t size_type;</td>
<td>Type to use when specifying sizes to methods of the container.</td>
</tr>
<tr>
<td>template&lt;typename OffsetT = no_offset&gt; using accessor;</td>
<td>Template alias to an accessor for this container</td>
</tr>
<tr>
<td>template&lt;typename OffsetT = no_offset&gt; using const_accessor;</td>
<td>Template alias to a const_accessor for this container</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aos1d_container(size_type size_d1 = 0u, buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0), const allocator_type &amp; an_allocator = allocator_type());</td>
<td>Constructs an uninitialized container of size_d1 elements, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing.</td>
</tr>
<tr>
<td>aos1d_container(size_type size_d1, const PrimitiveT &amp; a_value, buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0), const allocator_type &amp; an_allocator = allocator_type());</td>
<td>Constructs a container of size_d1 elements initializing each with a_value, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing.</td>
</tr>
<tr>
<td>template&lt;typename StlAllocatorT&gt; aos1d_container(const std::vector&lt;PrimitiveT, StlAllocatorT&gt; &amp; other, buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0), const allocator_type &amp; an_allocator = allocator_type());</td>
<td>Constructs a container with a copy of each of the elements in other, in the same order, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing.</td>
</tr>
<tr>
<td>aos1d_container(const PrimitiveT * other_array, size_type number_of_elements, buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0), const allocator_type &amp; an_allocator = allocator_type());</td>
<td>Constructs a container with a copy of number_of_elements elements from the array other_array, in the same order, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing.</td>
</tr>
</tbody>
</table>
specifies the number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing.

Constructs a container with as many elements as the range `[a_begin-an_end)`, each with a copy of the value from its corresponding element in that range, in the same order, using optionally specified allocator instance, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing.

Returns: a new `aos1d_container` instance with its own copy of the elements.

Resizes the container so that it contains `new_size_d1` elements. If the new size is greater than the current container size, the new elements are uninitialized.

Returns: `accessor` with no embedded index offset.

Returns: `accessor` with an integer based embedded index offset.

Returns: `accessor` with a fixed offset based embedded index offset.

Returns: `const_accessor` with no embedded index offset.

Returns: `const_accessor` with an integer based embedded index offset.

Returns: `const_accessor` with an aligned offset based embedded index offset.

Returns: `const_accessor` with a fixed offset based embedded index offset.

STL Compatibility

In addition to the performance oriented interface explained in the table above, `aos1d_container` implements a subset of the `std::vector` interface that is intended for ease of integration, not high performance. Due to the import/export only requirement we can’t return a reference to the object, instead
Iterators and operator[] return a Proxy object while other "const' methods return a "value_type const". Furthermore, iterators do not support the -> operator. Despite that limitation the iterators can be passed to any STL algorithm. Also for performance reasons, resize does not initialize new elements. The following std::vector interface methods are implemented:

- size, max_size, capacity, empty, reserve, shrink_to_fit
- assign, push_back, pop_back, clear, insert, emplace, erase
- cbegin, cend, begin, end, crbegin, crend, rbegin, rend, rbegin, rend
- operator[], front() const, back() const, at() const
- swap, ==, !=
- swap, aos1d_container(aos1d_container&& donor), aos1d_container & operator=(aos1d_container&& donor)

**access_by**

Enum to control how the memory layout will be accessed. `#include <sdlt/access_by.h>`

**Syntax**

```cpp
enum access_by
{
    access_by_struct,
    access_by_stride
};
```

**Description**

The `access_by_struct` causes data access via structure member access. Nested structures will drill down through the structure members in a nested manner. For example an Axis Aligned Bounding Box (AABB) containing two Point3d objects (with x,y,z data members) will logically expand to something like:

```cpp
AABB local;
local = accessor.mData[i];
```

`access_by_stride` will cause data access through pointers to built in types with a stride to account for the size of the primitive. For an Axis Aligned Bounding Box (AABB) containing two Point3d objects (with x,y,z data members) will logically expand to something like:

```cpp
AABB local;
local.topLeft.x = *(accessor.mData + offsetof(AABB,topLeft) + offset(Point3d,x) + (sizeof(AABB)*i));
local.topLeft.y = *(accessor.mData + offsetof(AABB,topLeft) + offset(Point3d,y) + (sizeof(AABB)*i));
local.topLeft.z = *(accessor.mData + offsetof(AABB,topLeft) + offset(Point3d,z) + (sizeof(AABB)*i));
local.topRight.x = *(accessor.mData + offsetof(AABB,topRight) + offset(Point3d,x) + (sizeof(AABB)*i));
local.topRight.y = *(accessor.mData + offsetof(AABB,topRight) + offset(Point3d,y) + (sizeof(AABB)*i));
local.topRight.z = *(accessor.mData + offsetof(AABB,topRight) + offset(Point3d,z) + (sizeof(AABB)*i));
```

When vectorizing, `access_by_struct` can sometimes generate better code as the compiler could perform wide loads and use shuffle/insert instructions to move data into SIMD registers. However, depending on the complexity of the primitive, it can also fail to vectorize, especially when the primitive contains nested structures.

On the other hand `access_by_stride` has always vectorized successfully, because the data access is simplified to an array pointer with a stride. The compiler is able to handle any complexity of primitive, because it never sees the complexity and instead just sees the simple array pointer with strided access.
**access_by_struct** is probably the best choice as it offers a chance of better code generation especially when used outside of a SIMD loop. However if you run into issues when vectorizing, try **access_by_stride** to see if that alleviates the problem.

We leave this choice up to the developer and require they explicitly make a choice, so this is not hidden behavior.

**n_container**

*Template class for N-dimensional container. The contained primitive type, exact memory layout and container shape are defined via template arguments.*

**Syntax**

```cpp
template <typename PrimitiveT,
          typename LayoutT,
          typename ExtentsT,
          typename AllocatorT >
class n_container;
```

**Description**

N-dimensional container of PrimitiveT elements with predefined memory layout and shape. Provides accessor interface suitable for flexible and efficient data access inside SIMD loops.

The following table provides information on the template arguments for **n_container**

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typename PrimitiveT</td>
<td>The type that each cell in the multi-dimensional container will store. Requirements: PrimitiveT must be previously declared with the SDLT_PRIMITIVE macro.</td>
</tr>
<tr>
<td>typename LayoutT</td>
<td>The in-memory data layout of cells in the container. Requirements: LayoutT must be a class from layout namespace.</td>
</tr>
<tr>
<td>typename ExtentsT</td>
<td>The shape of the container. Requirements: ExtentsT must be a concrete type of n_extent_t variadic template.</td>
</tr>
<tr>
<td>class AllocatorT = allocator::default_alloc</td>
<td>[Optional] Specify type of allocator to be used. allocator::default_alloc is currently the only allocator supported.</td>
</tr>
</tbody>
</table>

The following table provides information on the types defined as members of **n_container**

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef PrimitiveT primitive_type;</td>
<td>Type inside each cell of the container.</td>
</tr>
<tr>
<td>typedef PrimitiveT allocator_type;</td>
<td>Type of allocator used by the container.</td>
</tr>
<tr>
<td>typedef implementation-defined accessor</td>
<td>Type of an accessor that can write or read cells to and from this container.</td>
</tr>
</tbody>
</table>
typedef implementation-defined const_accessor;

Type of a const_accessor that can read cells from this container.

The following table provides information on the methods of n_container.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_container (const ExtentsT &amp;a_extents, buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0), const AllocatorT &amp;an_allocator=AllocatorT())</td>
<td>Constructs an uninitialized container of the shape defined as a_extents, using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing, using optionally specified allocator instance. ExtentsT must be default constructible. Only true when ExtentsT is made up entirely of fixed&lt;NumberT&gt; types.</td>
</tr>
<tr>
<td>n_container (buffer_offset_in_cachelines buffer_offset = buffer_offset_in_cachelines(0), const AllocatorT &amp;an_allocator=AllocatorT())</td>
<td>Constructs an uninitialized container of the shape, defined via template parameter ExtentsT using optionally specified number of cache lines to offset the start of the buffer in memory to allow management of 4k cache aliasing, using optionally specified allocator instance.</td>
</tr>
<tr>
<td>n_container(n_container&amp;&amp; donor)</td>
<td>Transfers ownership of the donor’s currently owned buffers and organization, if any. Any outstanding accessors on the donor are no longer valid.</td>
</tr>
<tr>
<td>n_container &amp; operator = (n_container&amp;&amp; donor)</td>
<td>Frees any existing buffers, then transfers ownership of the donor’s currently owned buffers and organization, if any. Any outstanding accessors on the donor are no longer valid.</td>
</tr>
<tr>
<td>const ExtentsT&amp; n_extent () const</td>
<td>Provides the shape of the container. Alternatively, the free template function extent_d&lt;DimensionT&gt;(const n_container &amp;) could be used.</td>
</tr>
<tr>
<td>const_accessor const_access();</td>
<td>Constructs a const_accessor with knowledge of the underlying data organization to read cells inside the container.</td>
</tr>
<tr>
<td>accessor access();</td>
<td>Constructs an accessor with knowledge of the underlying data organization to write or read cells inside the container.</td>
</tr>
</tbody>
</table>

The following table provides information about the friend functions of n_container.
Friend Function

\[
\text{std::ostream}& \\
\text{operator } \ll (\text{std::ostream}& \\
\text{output\_stream, const} \\
\text{n\_container & a\_container})
\]

Append string representation of a_container's extents values to a_output_stream.

**Returns:** Reference to a_output_stream for chained calls.

## Layouts

### sdl::layout namespace

Rather than having different container types for different data layouts, the library uses the types from the layout namespace as a template parameter to the n_container.

Available layouts are defined in the namespace layout and summarized in table below.

<table>
<thead>
<tr>
<th>Layout</th>
<th>Description</th>
</tr>
</thead>
</table>
| template &lt;typename AlignOnColumnIndexT=0&gt; \\
layout::soa | Structure of Arrays: Each data member of the Primitive will have its own N-dimensional array. The arrays are placed back-to-back inside a contiguous buffer. Template parameter AlignOnColumnIndexT identifies which column of the row dimension should be cache line aligned. The AlignOnColumnIndexT of each row is cache line aligned. Structure of Arrays Per Row: Each data member of the Primitive will have its own 1-dimensional array for the row dimension (Soa1d) placed back to back. The AlignOnColumnIndexT of each row is cache line aligned. Multiple of these Soa1d's are laid out sequentially to model the remaining dimensions, effectively becoming an Array of Structures of Arrays where the SOA where the size of the array is the row's extent. This can be particularly efficient when the extent of the row can be fixed<NumberT>. **Note:** If the size of the row isn't known at compile time, consider adding an additional dimension that is fixed<Number> and dividing the row up by that fixed<NumberT>. |
| template &lt;typename AlignOnColumnIndexT&gt; \\
layout::soa_per_row | |
| layout::aos_by_struct | Array of Structures Accessed by Struct: Primitives are laid out in native format back to back in memory and access happens via structure or member access. Nested structures will drill down through the structure members in a nested manner. |
| layout::aos_by_stride | Array of Structures Accessed by Stride: Primitives are laid out in native format back to back in memory and accessed through pointers to built in types with a stride to account for the size of the Primitive. Can be useful if aos_by_struct doesn't vectorize. |

**Description**

The classes are empty and only for specialization of containers for denoted layouts.
Shape

Variadic template class `n_extent_t` describes the shape of the n_dimensional container. Specifically, the number of dimensions the size of each.

Syntax

```cpp
template<typename... TypeListT>
class n_extent_t
```

Description

`n_extent_t` represents the shape of a container as a sequence of sizes for each dimension. The size of each dimension can be represented by different types. This flexibility allows the same interface to be used to declare `n_extents_t` whose dimensions are fully known at compile time with `fixed<int NumberT>`, or to be only known at runtime with `int`, or only known at runtime but with a guarantee will be a multiple of an alignment with `aligned<int Alignment>`. For details, see the Number representation section.

The following table provides information on the template arguments for `n_extent_t`.

<table>
<thead>
<tr>
<th>Template Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typename... TypeListT</code></td>
</tr>
</tbody>
</table>

Comma separated list of types, where the number of types provided controls how many dimensions there are. Each type in the list identifies how the size of the corresponding dimension is to be represented. The order of the dimensions is the same order as C++ subscripts declaring a multi-dimensional array, from leftmost to rightmost.

Type must be `int`, `fixed<NumberT>`, or `aligned<AlignmentT>` for each value describing corresponding dimensions size (extent) in regular order of C++ subscripts - from outer to inner.

The following table provides information on the members of `n_extent_t`.

<table>
<thead>
<tr>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static constexpr int rank;</code></td>
</tr>
</tbody>
</table>

Number of dimensions.

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static constexpr int row_dimension = rank-1;</code></td>
</tr>
</tbody>
</table>

Index of last dimension, row.

| Requirements: Every type in TypeListT is default constructible. |
| Effects: Construct `n_extent_t`, uses default values of each type in TypeListT for the dimension sizes. In general, only correctly initialized when every type is a `fixed<NumberT>` |

<table>
<thead>
<tr>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>n_extent_t()</code></td>
</tr>
</tbody>
</table>

N/A

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>n_extent_t(const n_extent_t &amp;a_other)</code></td>
</tr>
</tbody>
</table>

Effects: Construct `n_extent_t`, copying size of each dimension from `a_other`. |
explicit n_extent_t(const TypeListT & ... a_values)

**Effects:** Construct `n_extent_t`, initializing each dimension with the corresponding value from the list of `a_values` passed as an argument. In use, `a_values` is a comma separate list of values whose length and types are defined by `TypeListT`.

**Requirements:** `DimensionT` >=0 and `DimensionT` <= rank.

**Effects:** Determine the extent of `DimensionT`.

**Returns:** In the type declared by the `DimensionT` position of 0-based `TypeListT`, the extent of the specified `DimensionT`.

**Requirements:** `DimensionT` >=0 and `DimensionT` <= rank.

**Effects:** Construct a `n_extent_t` with a lower rank by copying the rightmost `DimensionT` values from this instance.

**Returns:** `n_extent[get<rank - DimensionT>()]` [get<rank + 1 - DimensionT>()] [get<...>()] [get<row_dimension>()]

**Requirements:** rank of `a_other` is the same as this instance’s.

**Effects:** Compare size of each dimension for equality. Only compares numeric values, not the types of each dimension.

**Returns:** `true` if all dimensions are numerically equal, `false` otherwise.

**Requirements:** rank of `a_other` is the same as this instance’s.

**Effects:** Compare size of each dimension for inequality. Only compares numeric values, not the types of each dimension.

**Returns:** `true` if any dimensions are numerically different, `false` otherwise.

**Returns:** Number of elements specified by extent

**Effects:** Calculates the number of cells represented by the current extent values of each dimension by multiplying them all together.

**Returns:** `get<0>()*get<1>()*get<...>()[...]*get<rank-1>()`

The following table provides information on the friend functions of `n_extent_t`.  

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit n_extent_t(const TypeListT &amp; ... a_values)</td>
<td><strong>Effects:</strong> Construct <code>n_extent_t</code>, initializing each dimension with the corresponding value from the list of <code>a_values</code> passed as an argument. In use, <code>a_values</code> is a comma separate list of values whose length and types are defined by <code>TypeListT</code>. <strong>Requirements:</strong> <code>DimensionT</code> &gt;=0 and <code>DimensionT</code> &lt;= rank. <strong>Effects:</strong> Determine the extent of <code>DimensionT</code>. <strong>Returns:</strong> In the type declared by the <code>DimensionT</code> position of 0-based <code>TypeListT</code>, the extent of the specified <code>DimensionT</code>. <strong>Requirements:</strong> <code>DimensionT</code> &gt;=0 and <code>DimensionT</code> &lt;= rank. <strong>Effects:</strong> Construct a <code>n_extent_t</code> with a lower rank by copying the rightmost <code>DimensionT</code> values from this instance. <strong>Returns:</strong> <code>n_extent[get&lt;rank - DimensionT&gt;()]</code> [get&lt;rank + 1 - DimensionT&gt;()] [get&lt;...&gt;()] [get&lt;row_dimension&gt;()] <strong>Requirements:</strong> rank of <code>a_other</code> is the same as this instance’s. <strong>Effects:</strong> Compare size of each dimension for equality. Only compares numeric values, not the types of each dimension. <strong>Returns:</strong> <code>true</code> if all dimensions are numerically equal, <code>false</code> otherwise. <strong>Requirements:</strong> rank of <code>a_other</code> is the same as this instance’s. <strong>Effects:</strong> Compare size of each dimension for inequality. Only compares numeric values, not the types of each dimension. <strong>Returns:</strong> <code>true</code> if any dimensions are numerically different, <code>false</code> otherwise. <strong>Returns:</strong> Number of elements specified by extent</td>
</tr>
</tbody>
</table>
**Friend function**

```cpp
std::ostream& operator << (std::ostream& output_stream, const n_extent_t & a_extents)
```

**Description**

**Effects:** Append string representation of a_extents’ values to a_output_stream

**Returns:** Reference to a_output_stream for chained calls.

---

**n_extent_generator**

*To facilitate simpler and clearer creation of n_extent_t objects.*

**Syntax**

```cpp
template<typename... TypeListT>
class n_extent_generator;
```

```cpp
namespace {
    // Instance of generator object
    n_extent_generator<> n_extent;
}
```

**Description**

The generator object provides recursively constructing operators [] for fixed<> aligned<> and integer values allowing building of an n_extent_t ... instance, one dimension at a time. The main purpose is to allow a usage syntax that is similar to C multi-dimensional array definition:

Compare the following examples, instantiating three n_extent_t instances. and using the generator object to instantiate equivalent instances.

```cpp
n_extent_t<int, int> ext1(height, width);
n_extent_t<int, aligned<128>> ext2(height, width);
n_extent_t<fixed<1080>, fixed<1920>> ext3(1080_fixed, 1920_fixed);
```

```cpp
auto ext1 = n_extent[height][width];
auto ext2 = n_extent[height][aligned<128>(width)];
auto ext3 = n_extent[1080_fixed][1920_fixed];
```

**Class Hierarchy**

It is expected that n_extent_generator <> not be directly used as a data member or parameter, instead only n_extent_t <> from which it is derived. The generator object n_extent can be automatically downcast any place expecting an n_extent_t<...>.

The following table provides the template arguments for n_extent_generator

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typename... TypeListT</td>
<td>Comma separated list of types, where the number of types provided controls how many dimensions the generator currently represent. Each type in the list identifies how the size of the corresponding dimension is to be represented. The order of the dimensions is the same order as C++ subscripts declaring a multi-dimensional array – from leftmost to rightmost. <strong>Requirements:</strong> Type is int, fixed&lt;NumberT&gt;, or aligned&lt;AlignmentT&gt;.</td>
</tr>
</tbody>
</table>

The following table provides information on the types defined as members of n_extent_generator in addition to those inherited from n_extent_t.
typedef n_extent_t<TypeListT...> value_type

Type value that the any chained [] operator calls have produced.

The following table provides information on the members of n_extent_generator in addition to those inherited from n_extent_t

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef n_extent_t&lt;TypeListT...&gt; value_type</td>
<td>Type value that the any chained [] operator calls have produced.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_extent_generator ()</td>
<td><strong>Requirements:</strong> TypeListT is empty</td>
</tr>
<tr>
<td></td>
<td><strong>Effects:</strong> Construct generator with no extents specified</td>
</tr>
<tr>
<td>n_extent_generator (const n_extent_generator &amp;a_other)</td>
<td><strong>Effects:</strong> Construct generator copying any extent values from a_other</td>
</tr>
<tr>
<td>n_extent_generator&lt;TypeListT..., int&gt; operator () (int a_size) const</td>
<td><strong>Requirements:</strong> a_size &gt;= 0</td>
</tr>
<tr>
<td></td>
<td><strong>Returns:</strong> n_extent_generator&lt;...&gt; with additional rightmost integer based extent.</td>
</tr>
<tr>
<td>n_extent_generator&lt;TypeListT..., fixed&lt;NumberT&gt;&gt; operator () (fixed&lt;NumberT&gt; a_size) const</td>
<td><strong>Requirements:</strong> a_size &gt;= 0</td>
</tr>
<tr>
<td></td>
<td><strong>Returns:</strong> n_extent_generator&lt;...&gt; with additional rightmost fixed&lt;NumberT&gt; extent.</td>
</tr>
<tr>
<td>n_extent_generator&lt;TypeListT..., aligned&lt;AlignmentT&gt;&gt; operator () (aligned&lt;AlignmentT&gt; a_size)</td>
<td><strong>Requirements:</strong> a_size &gt;= 0</td>
</tr>
<tr>
<td></td>
<td><strong>Returns:</strong> n_extent_generator&lt;...&gt; with additional rightmost aligned&lt;AlignmentT&gt; based extent.</td>
</tr>
<tr>
<td>value_type value () const</td>
<td><strong>Returns:</strong> n_extent_t&lt;...&gt; with the correct types and values of the multi-dimensional extents aggregated by the generator.</td>
</tr>
</tbody>
</table>

**make_n_container template function**

Factory function to construct an instance of a properly-typed n_container<...> based on n_extent_t passed to it.

**Syntax**

```cpp
template<
    typename PrimitiveT,
    typename LayoutT,
    typename AllocatorT = allocator::default_alloc,
    typename ExtentsT
>
auto make_n_container(const ExtentsT &_extents) -> n_container<PrimitiveT, LayoutT, ExtentsT, AllocatorT>
```

**Description**

Use make_n_container to more easily create an n-dimensional container using template argument deduction, and avoid specifying the type of extents.

An example of the instantiation of a High Definition image object is below.

```cpp
typedef n_container<RGBAs, layout::soa,
    n_extent_t<int, int>> HdImage;
HdImage image1(n_extent[1080][1920]);
```
Alternatively, it is possible to use factory function with the C++11 keyword `auto`, as shown below.

```cpp
auto image1 = make_n_container<RGBAs,
    layout::soa>(n_extent[1080][1920]);
```

### extent_d template function

**Syntax**

```cpp
template<int DimensionT, typename ObjT>
auto extent_d(const ObjT &a_obj)
```

**Description**

The template function offers a consistent way to determine the extent of a dimension for a multi-dimensional object. It can avoid extracting an entire `n_extent_t<...>` when only the extent of a single dimension is needed.

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int DimensionT</code></td>
<td>0 based index starting at the leftmost dimension indicating which n-dimensions to query the extent of.</td>
</tr>
<tr>
<td><code>typename ObjT</code></td>
<td>The type of n-dimensional object from which to retrieve the extent.</td>
</tr>
</tbody>
</table>

**Requirements:** `DimensionT` >=0 and `DimensionT` < `ObjT::rank`

**Returns**

The correctly typed extent corresponding to the requested `DimensionT` of `a_obj`.

**Example**

```cpp
template <typename VolumeT>
void foo(const VolumeT & a_volume)
{
    int extent_z = extent_d<0>(volume);
    int extent_y = extent_d<1>(volume);
    int extent_x = extent_d<2>(volume);
    // …
}
```

### Bounds

This section provides information related to bounds for the SIMD Data Layout Templates (SDLT).

**bounds_t**

*Class represents a half-open interval with lower and upper bounds.* #include `<sdlt/bounds.h>`

**Syntax**

```cpp
template<typename LowerT = int, typename UpperT = int>
struct bounds_t
```
**Description**

`bounds_t` holds the lower and upper bounds of a half open interval. It is templated to allow the different representations for the lower and upper bounds. Supported types include fixed<NumberT>, aligned<AlignmentT> and integer values. `bounds_t` models a valid iteration space over a single dimension.

`bounds_t` can be used to represent an iteration space over the entire extent of a dimension or to restrict iteration space within the extent. `n_bounds_t` aggregates a number of `bounds_t` objects to allow construction of multi-dimensional subsections restricting multiple extents.

The class interface is compatible with C++ range-based loops to simplify iteration.

<table>
<thead>
<tr>
<th><strong>Template Argument</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typename LowerT = int</code></td>
<td>Type of lower bound. Requirements: type is int, or fixed&lt;NumberT&gt;, or aligned&lt;AlignmentT&gt;</td>
</tr>
<tr>
<td><code>typename UpperT = int</code></td>
<td>Type of upper bound. Requirements: type is int, or fixed&lt;NumberT&gt;, or aligned&lt;AlignmentT&gt;</td>
</tr>
</tbody>
</table>

**Member Types**

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedef LowerT lower_type</code></td>
</tr>
<tr>
<td><code>typedef UpperT upper_type</code></td>
</tr>
<tr>
<td><code>typedef implementation-defined iterator</code></td>
</tr>
</tbody>
</table>

**Member**

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bounds_t()</code> Effects: Constructs <code>bounds_t</code> with uninitialized lower and upper bounds.</td>
</tr>
<tr>
<td><code>bounds_t(lower_type l, upper_type u)</code> Effects: Constructs <code>bounds_t</code> representing the half-open interval [l, u)</td>
</tr>
<tr>
<td><code>bounds_t(const bounds_t &amp; a_other)</code> Effects: Constructs <code>bounds_t</code> with lower and upper bounds initialized from those of <code>a_other</code>. Requirements: <code>OtherLowerT</code> and <code>OtherUpperT</code> can legally be converted to <code>lower_type</code> and <code>upper_type</code>. For example it would be illegal to convert an int to fixed&lt;8&gt;().</td>
</tr>
<tr>
<td><code>template&lt;typename OtherLowerT, typename OtherUpperT&gt; bounds_t(const bounds_t&lt;OtherLowerT, OtherUpperT&gt; &amp; a_other)</code> Effects: Constructs <code>bounds_t</code> with lower and upper bounds initialized from those of <code>a_other</code>.</td>
</tr>
<tr>
<td><code>void set(lower_type l, upper_type u)</code> Effects: Set index of the inclusive lower bound and the index of the exclusive upper bound.</td>
</tr>
<tr>
<td><code>void set_lower(lower_type a_lower)</code> Effects: Set index of the inclusive lower bound</td>
</tr>
<tr>
<td><code>void set_upper(upper_type a_upper)</code> Effects: Set index of the exclusive upper bound</td>
</tr>
<tr>
<td>Member</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>lower_type lower() const</td>
</tr>
<tr>
<td>upper_type upper() const</td>
</tr>
<tr>
<td>iterator begin() const</td>
</tr>
<tr>
<td>iterator end() const</td>
</tr>
<tr>
<td>auto width() const</td>
</tr>
<tr>
<td>template&lt;typename OtherLowerT, typename OtherUpperT&gt; bool contains(const bounds_t&lt;OtherLowerT, OtherUpperT&gt; &amp;a_other) const</td>
</tr>
<tr>
<td>template&lt;typename T&gt; auto operator + (const T &amp;offset) const</td>
</tr>
<tr>
<td>template&lt;typename T&gt; auto operator - (const T &amp; offset) const</td>
</tr>
<tr>
<td>bool operator == (const bounds_t &amp;a_other) const</td>
</tr>
<tr>
<td>template&lt;typename OtherLowerT, typename OtherUpperT&gt; bool operator == (const bounds_t&lt;OtherLowerT, OtherUpperT&gt; &amp;a_other) const</td>
</tr>
</tbody>
</table>
### Member

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool operator != (const bounds_t &amp;a) const</code></td>
<td>Effects: Inequality comparison with same-typed bounds_t object</td>
</tr>
<tr>
<td><code>template&lt;typename OtherLowerT, typename OtherUpperT&gt;</code></td>
<td>Returns: `(lower() != a_other.lower()</td>
</tr>
</tbody>
</table>

### Friend Function

<table>
<thead>
<tr>
<th>Friend Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>std::ostream&amp; operator &lt;&lt; (std::ostream&amp; a_output_stream, const bounds_t &amp;a_bounds)</code></td>
<td>Effects: append string representation of bounds_t lower and upper values to a_output_stream</td>
</tr>
<tr>
<td><code>template&lt;typename OtherLowerT, typename OtherUpperT&gt;</code></td>
<td>Returns: reference to a_output_stream for chained calls</td>
</tr>
</tbody>
</table>

### Range-based loops support

The `bounds_t` provides `begin()` and `end()` methods returning iterators to enable C++11 range-based loops. The may save quite some typing and improve code clarity when iterating over bounds of a multidimensional container.

Compare:

```cpp
auto ca = image_container.const_access();
auto b0 = bounds_d<0>(ca);
auto b1 = bounds_d<1>(ca);
for (auto y = b0.lower(); y < b0.upper(); ++y)
    for (auto x = b1.lower(); x < b1.upper(); ++x) {
        RGBAs pixel = ca[y][x];
        // …
    }
```

and

```cpp
auto ca = image_container.const_access();
for (auto y: bounds_d<0>(ca))
    for (auto x: bounds_d<1>(ca)) {
        RGBAs pixel = ca[y][x];
        // …
    }
```

Note that iterator only gives an index value within the bounds, not an object value. It is expected to be used to index into accessors like in example above.

### `sdlt::bounds Template Function`

Factory function provided for creation of `bounds_t` objects. #include `<sdlt/bounds.h>`

### Syntax

```cpp
template<typename LowerT, typename UpperT>
auto bounds(LowerT a_lower, UpperT a_upper)
```
Description
In order to make creation of objects of bounds_t cleaner the factory function bounds is provided. It basically enables LowerT and UpperT to be deduced from the arguments passed into it.

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typename LowerT = int</td>
<td>Type of lower bound.</td>
</tr>
<tr>
<td></td>
<td>Requirements: type is int, or fixed&lt;NumberT&gt;, or aligned&lt;AlignmentT&gt;</td>
</tr>
<tr>
<td>typename UpperT = int</td>
<td>Type of upper bound.</td>
</tr>
<tr>
<td></td>
<td>Requirements: type is int, or fixed&lt;NumberT&gt;, or aligned&lt;AlignmentT&gt;</td>
</tr>
</tbody>
</table>

Returns:
The correctly typed bounds_t<LowerT, UpperT> corresponding to types of a_lower and a_upper passed to the factory function.

Example:
Compare two ways of instantiating a bounds:

```cpp
bounds_t<fixed<0>, aligned<16>> my_bounds1(0_fixed, aligned<16>(upper))
auto my_bounds2 = bounds_t<fixed<0>, aligned<16>>(0_fixed, aligned<16>(upper))
```

With the factory function:

```cpp
auto my_bounds = bounds(0_fixed, aligned<16>(upper))
```

n_bounds_t
Variadic template class to describe the valid iteration space over an N-dimensional container. #include <sdlt/n_bounds.h>

Syntax

template<typename... TypeListT>
class n_bounds_t

Description

n_bound_t represents the valid iteration space over a n_container or its accessor as as a sequence of bounds_t for each dimension. The bounds_t of each dimension can be represented by different types. This flexibility allows the same interface to be used to declare n_bounds_t whose dimensions are fully known at compile time with fixed<int NumberT>, or to be only known at runtime with int, or only known at runtime but with a guarantee will be a multiple of an alignment with aligned<int Alignment>. For details see the Number Representation section).

When an n_container is created, its n_bounds_t always start at fixed<0> for the inclusive lower bounds of each dimension, and exclusive upper bounds match the extent of the dimension. Accessors can be translated to different index spaces as well as restrict their iteration space to subsections, which will change the n_bounds_t those accessors provide.

The following table provides information on the template arguments for n_bounds_t.
### Template Argument

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comma separated list of types, where the number of types provided controls how many dimensions there are. Each type in the list identifies how the bounds of the corresponding dimension is to be represented. The order of the dimensions is the same order as C++ subscripts declaring a multi-dimensional array – from leftmost to rightmost.</td>
</tr>
</tbody>
</table>

Requirements: types in the list be `bounds_t<LowerT, UpperT>`

### The following table provides information on the member types of `n_bounds_t`

<table>
<thead>
<tr>
<th>Member Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef implementation-defined <code>lower_type</code></td>
<td>Type of <code>n_index_t&lt;...&gt;</code> returned by method <code>lower()</code></td>
</tr>
<tr>
<td>typedef implementation-defined <code>upper_type</code></td>
<td>Type of <code>n_index_t&lt;...&gt;</code> returned by method <code>upper()</code></td>
</tr>
</tbody>
</table>

### The following table provides information on the members of `n_bounds_t`.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static constexpr int <code>rank</code>;</td>
<td>Number of dimensions</td>
</tr>
<tr>
<td>static constexpr int <code>row_dimension</code> = rank-1;</td>
<td>Index of last dimension considered to be the row</td>
</tr>
</tbody>
</table>

Requirements: Every `bounds_t` in `TypeListT` is default constructible.

Effects: Construct `n_bounds_t`, uses default values of each `bounds_t` in `TypeListT` for the dimension sizes. In general only correctly initialized when every `bounds_t` has an `LowerT` and `UpperT` that is a fixed<NumberT>.

Effects: Construct `n_bounds_t`, copying bounds of each dimension from `a_other`.

Requirements: DimensionT >=0 and DimensionT < rank.

Effects: Determine the bounds of DimensionT.

Returns: In the type declared by the DimensionT position of 0-based `TypeListT`, the `bounds_t` of the specified DimensionT

Effects: build `n_index<...>` representing the inclusive lower bounds for all dimensions

Returns: `n_index[get<0>().lower()] [get<1>().lower()] [get<...>().lower()] [get<row_dimension>().lower()]`
<table>
<thead>
<tr>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>upper_type upper()</strong></td>
</tr>
</tbody>
</table>

**Description**

Effects: build n_index<...> representing the exclusive upper bounds for all dimensions.

Returns: \( \text{n_index}[\text{get<0>()}.\text{upper()}] \)

\( \text{[get<1>()}.\text{upper ()}] \)

\( \text{[get<...>()}.\text{upper ()}] \)

\( \text{[get<row_dimension>()}.\text{upper()}] \)

<table>
<thead>
<tr>
<th>Template</th>
</tr>
</thead>
</table>
| **template<typename... OtherTypeListT>**

**bool contains(n_bounds_t<OtherTypeListT...> &a_other) const**

**Description**

Requirements: rank of a_other is the same as this instance's.

Effects: Determine whether each dimension of the passed n_bounds_t is fully contained within bounds of each dimension of this object.

Returns: \( \text{get<0>()}.\text{contains(a_other.get<0>() ) && get<1>()}.\text{contains(a_other.get<1>() ) && get<...>()}.\text{contains(a_other.get<...>() ) && get<row_dimension>()}.\text{contains(a_other.get<row_dimension>() )} \)

<table>
<thead>
<tr>
<th>Template</th>
</tr>
</thead>
</table>
| **template<class... OtherTypeListT>**

**bool operator == (const n_bounds_t<OtherTypeListT...> a_other) const**

**Description**

Requirements: rank of a_other is the same as this instance's.

Effects: Compare bounds each of dimension for equality. Only compares numeric values, not the types of each dimension.

Returns: true if all dimensions are numerically equal, false otherwise.

<table>
<thead>
<tr>
<th>Template</th>
</tr>
</thead>
</table>
| **template<class... OtherTypeListT>**

**bool operator != (const n_bounds_t<OtherTypeListT...> a_other) const**

**Description**

Requirements: rank of a_other is the same as this instance's.

Effects: Compare bounds of each dimension for inequality. Only compares numeric values, not the types of each dimension.

Returns: true if any dimensions are numerically different, false otherwise.

<table>
<thead>
<tr>
<th>Template</th>
</tr>
</thead>
</table>
| **template<class ... OtherTypeListT>**

**auto operator+ (const n_index_t<OtherTypeListT...> a_offset) const**

**Description**

Requirements: rank of a_other is the same as this instance's.

Effects: construct a n_bound_t whose types and bounds value for each dimension are determined by taking the bounds for each dimension and adding the an offset for that dimension from a_offset.

Returns: \( \text{n_bounds}[\text{get<0>() + a_offset.get<0>()} \]

\( \text{[get<1>() + a_offset.get<1>()]} \)

\( \text{[get<...>() + a_offset.get<...>()]} \)

\( \text{[get<row_dimension>() + a_offset.get<row_dimension>()]} \)


template<int DimensionT>
auto rightmost_dimensions() const

Requirements: DimensionT >=0 and DimensionT <= rank.
Effects: Construct a n_bounds_t with a lower rank by copying the rightmost DimensionT values from this instance.
Returns: n_bounds[get<rank – DimensionT>()]
[get<rank + 1 – DimensionT>()]
[get<...>()]
[get<row_dimension>()]

template<class... OtherTypeListT>
auto overlay_rightmost(const n_bounds_t<OtherTypeListT...> & a_other) const

Requirements: rank of a_other is <= rank
Effects: Construct copy of n_bounds_t where the rightmost dimensions' values are copied from a_other, effectively overlaying a_other on top of rightmost dimensions of this instance.
Returns:
 n_bounds[get<0>()]
[get<1>()]
[get<...>()]
[get<rank-a_other::rank>()]
[a_other.get<0>()]
[a_other.get<...>()]
[a_other.get<a_other::row_dimension>()]

The following table provides information on the friend functions of n_bounds_t.

<table>
<thead>
<tr>
<th>Friend Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| std::ostream& operator << (std::ostream& output_stream, const n_bounds_t & a_bounds_list) | Effects: append string representation of a_bounds_list values to output_stream
Returns: reference to output_stream for chained calls. |

**n_bounds_generator**
Facilitates simple creation of n_bounds_t objects.

```
#include <sdlt/n_bounds.h>
```

**Syntax**

```cpp
template<typename... TypeListT>
class n_bounds_generator;

namespace {
    // Instance of generator object
    n_bounds_generator<> n_bounds;
}
```
The generator object provides recursively constructing operators [] for bounds_t<LowerT, UpperT> values allowing building of a n_bounds_t<...> instance one dimension at a time. Its main purpose is to allow a usage syntax that is similar to C multi-dimensional array definition:

Compare creating two n_bounds_t instances:

```cpp
n_bounds_t<bounds_t<fixed<540>, fixed<1080>>,
    bounds_t<fixed<960>, fixed<1920>>> bounds1(bounds_t<540_fixed,1080_fixed>(),
    bounds_t<960_fixed, 1920_fixed>);

n_bounds_t<bounds_t<int, int>,
    bounds_t<int, int>> bounds2(bounds_t<int, int>(540,960),
    bounds_t<int, int>(960, 1920));
```

and the equivalent instances using the generator objects and factory functions

```cpp
auto bounds1 = n_bounds[bounds(540_fixed, 1080_fixed)]
    [bounds(960_fixed, 1920_fixed)];
auto bounds2 = n_bounds[bounds(540, 1080)]
    [bounds(960, 1920)];
```

or alternatively using the operator() with n_index_t and n_extent_t generator objects

```cpp
auto bounds1 = n_bounds(n_index[540_fixed][960_fixed],
    n_extent[540_fixed][960_fixed]);
auto bounds2 = n_bounds(n_index[540][960],
    n_extent[540][960]);
```

**Class Hierarchy**

It is expected that n_bounds_generator<...> not be directly used as a data member or parameter, instead only n_bounds_t<...> from which it is derived. The generator object n_bounds can be automatically downcast any place expecting a n_bounds_t<...>.

The following table provides information on the template arguments for n_bounds_generator

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typename... TypeListT</td>
<td>Comma separated list of types, where the number of types provided controls how many dimensions there are. Each type in the list identifies how the bounds of the corresponding dimension is to be represented. The order of the dimensions is the same order as C++ subscripts declaring a multi-dimensional array – from leftmost to rightmost. Requirements: types in the list be bounds_t&lt;LowerT, UpperT&gt;</td>
</tr>
</tbody>
</table>

The following table provides information on the types defined as members of n_bounds_generator in addition to those inherited from n_bounds_t

<table>
<thead>
<tr>
<th>Member Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef n_bounds_t&lt;TypeListT...&gt; value_type</td>
<td>Type value that the any chained [] operator calls have produced.</td>
</tr>
</tbody>
</table>

The following table provides information on the members of n_bounds_generator in addition to those inherited from n_bounds_t
### n_bounds_generator()

**Requirements:** TypeListT is empty

**Effects:** Construct generator with no bounds specified

### n_bounds_generator(const n_bounds_generator &a_other)

**Effects:** Construct generator copying any bounds values from a_other

### template<typename LowerT, typename UpperT>

auto operator [] (const bounds_t<LowerT, UpperT> &a_bounds) const

**Effects:** build a n_bounds_generator<...> with additional rightmost bounds_t<LowerT, UpperT> based dimension.

**Returns:** n_bounds_generator<TypeListT..., bounds_t< LowerT, UpperT >>

### template<class... IndexTypeListT, class... ExtentTypeListT>

auto operator () (const n_index_t<IndexTypeListT...> &a_indices, const n_extent_t<ExtentTypeListT...> &a_extents) const

**Requirements:** rank of a_indices is same as rank of a_extents and TypeListT be empty

**Effects:** build a n_bounds_generator<...> where n-lower bounds are specified by a_indices, and n-upper bounds are calculated by adding a_extents to a_indices

**Returns:** n_bounds[bounds(a_indices.get<0>(), a_indices.get<0>() + a_extents.get<0>())]

[bounds(a_indices.get<1>(), a_indices.get<1>() + a_extents.get<1>())]

[bounds(a_indices.get<...>(), a_indices.get<...>() + a_extents.get<...>())]

[bounds( a_indices.get<row_dimension>(), a_indices.get< row_dimension >() + a_extents.get< row_dimension >())]

**Returns:** n_bounds_t<...> with the correct types and values of the multi-dimensional bounds aggregated by the generator.

---

### bounds_d Template Function

*Provides a consistent way to determine the bounds of a dimension for a multi-dimensional object.*

```cpp
#include <sdlt/n_extent.h>
```

### Syntax

```cpp
template<int DimensionT, typename ObjT>
auto bounds_d(const ObjT &a_obj)
```

### Description

Consistent way to determine the bounds of a dimension for a multi-dimensional object. Can avoid extracting an entire n_bounds_t<...> when only the extent of a single dimension is needed.
<table>
<thead>
<tr>
<th><strong>Template Argument</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int DimensionT</code></td>
<td>0 based index starting at the leftmost dimension indicating which n-dimensions to query the bounds of. Requirements: DimensionT &gt;=0 and DimensionT &lt; ObjT::rank</td>
</tr>
<tr>
<td><code>typename ObjT</code></td>
<td>The type of n-dimensional object from which to retrieve the extent. Requirements: ObtT is one of: n_container&lt;...&gt; n_bounds_t&lt;...&gt; n_bounds_generator&lt;...&gt; n_container&lt;...&gt;::accessor n_container&lt;...&gt;::const_accessor or any sectioned or translated accessor.</td>
</tr>
</tbody>
</table>

**Returns:**
The correctly typed `bounds_t<LowerT, UpperT>` corresponding to the requested DimensionT of `a_obj`.

**Example:**

```cpp
template <typename VolumeT>
void foo(const VolumeT & a_volume)
{
    auto bounds_z = bounds_d<0>(volume);
    auto bounds_y = bounds_d<1>(volume);
    auto bounds_x = bounds_d<2>(volume);
    for(auto z : bounds_z)
        for(auto y : bounds_y)
            for(auto x : bounds_x) {
                // ...
            }
}
```

**Accessors**

This section provides information related to accessors for the SIMD Data Layout Templates (SDLT).

`soa1d_container::accessor` and `aos1d_container::accessor`

*Lightweight object provides efficient array subscript [] access to the read or write elements from inside a soa1d_container or aos1d_container.*

*include `<sdlt/soa1d_container.h>` and `include `<sdlt/aos1d_container.h>`*

**Syntax**

```cpp
template <typename OffsetT> soa1d_container::accessor;
template <typename OffsetT> aos1d_container::accessor;
```
**Arguments**

typename OffsetT

The type offset that will be applied to each operator[] call determined by the type of offset passed into
soa1d_container::access(offset)/ aos1d_container::access(offset) which constructs an accessor.

**Description**

accessor provides [] operator that returns a proxy object representing an Element inside the Container that can export or import the Primitive's data. Can re-access with an offset to create a new accessor that when accessed at [0] will really be accessing at index corresponding to the embedded offset. Lightweight and meant to be passed by value into functions or lambda closures. Use accessors in place of pointers to access the logical array data.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accessor();</td>
<td>Default Constructible</td>
</tr>
<tr>
<td>accessor(const accessor &amp;);</td>
<td>Copy Constructible</td>
</tr>
<tr>
<td>accessor &amp; operator = (const accessor &amp;);</td>
<td>Copy Assignable</td>
</tr>
<tr>
<td>const int &amp; get_size_d1() const;</td>
<td>Returns: Number of elements in the container.</td>
</tr>
<tr>
<td>auto operator [](int index_d1) const</td>
<td>Returns: proxy Element representing element at index_d1 in the container.</td>
</tr>
<tr>
<td>template&lt;typename IndexT_D1&gt; auto operator [](const IndexT_D1 index_d1);</td>
<td>When: IndexT_D1 is one of the SDLT defined or generated Index types, Returns: proxy Element representing element at index_d1 in the container.</td>
</tr>
<tr>
<td>auto reaccess(const int offset) const;</td>
<td>Returns: accessor with an integer-based embedded index offset.</td>
</tr>
<tr>
<td>template&lt;int IndexAlignmentT&gt; auto reaccess(aligned_offset&lt;IndexAlignmentT&gt; offset) const;</td>
<td>Returns: accessor with an aligned_offset&lt;IndexAlignmentT&gt; based embedded index offset.</td>
</tr>
<tr>
<td>template&lt;int fixed_offsetT&gt; auto reaccess(fixed_offset&lt;fixed_offsetT&gt;) const;</td>
<td>Returns: accessor with a fixed_offset&lt;OffsetT&gt; based embedded index offset.</td>
</tr>
</tbody>
</table>

**soa1d_container::const_accessor and aos1d_container::const_accessor**

Lightweight object provides efficient array subscript [] access to the read elements from inside a soa1d_container or aos1d_container. #include <sdlt/soa1d_container.h> and #include <sdlt/aos1d_container.h>

**Syntax**

template <typename OffsetT> soa1d_container::const_accessor;
template <typename OffsetT> aos1d_container::const_accessor;
Arguments

typename OffsetT

The type offset that embedded offset that will be applied to each operator[] call

Description

const_accessor provides [] operator that returns a proxy object representing a const Element inside the Container that can export the Primitive's data. Can re-access with an offset to create a new const_accessor that when accessed at [0] will really be accessing at index corresponding to the embedded offset. Lightweight and meant to be passed by value into functions or lambda closures. Use const_accessors in place of const pointers to access the logical array data.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const_accessor();</td>
<td>Default Constructible</td>
</tr>
<tr>
<td>const_accessor(const const_accessor &amp;);</td>
<td>Copy Constructible</td>
</tr>
<tr>
<td>const_accessor &amp; operator = (const const_accessor &amp;);</td>
<td>Copy Assignable</td>
</tr>
<tr>
<td>const int &amp; get_size_d1() const;</td>
<td>Returns: Number of elements in the container.</td>
</tr>
<tr>
<td>auto operator [] (int index_d1) const</td>
<td>Returns: proxy ConstElement representing element at index_d1 in the container.</td>
</tr>
<tr>
<td>template&lt;typename IndexT_D1&gt; auto operator [] (const IndexT_D1 index_d1);</td>
<td>When: IndexT_D1 is one of the SDLT defined or generated Index types. Returns: proxy ConstElement representing element at index_d1 in the container.</td>
</tr>
<tr>
<td>auto reaccess(const int offset) const;</td>
<td>Returns: const_accessor with an integer-based embedded index offset.</td>
</tr>
<tr>
<td>template&lt;int IndexAlignmentT&gt; auto reaccess(aligned_offset&lt;IndexAlignmentT&gt; offset) const;</td>
<td>Returns: const_accessor with an aligned_offset&lt;IndexAlignmentT&gt; based embedded index offset.</td>
</tr>
<tr>
<td>template&lt;int fixed_offsetT&gt; auto reaccess(fixed_offset&lt;fixed_offsetT&gt;) const;</td>
<td>Returns: const_accessor with a fixed_offset&lt;OffsetT&gt; based embedded index offset.</td>
</tr>
</tbody>
</table>

Accessor Concept

Accessor and const_accessor objects obtained via n_container::access() and n_container::const_access() provide access to read from or write to cells inside an n_container.
Syntax
The following methods return objects meeting the requirements of the accessor concept.

```cpp
auto n_container::access();
auto n_container::const_access();
auto accessor_concept::section(n_bounds_t<…>);
auto accessor_concept::translated_to(n_index_t<…>);
auto accessor_concept::translated_to_zero();
```

Description
Accessor objects provide read/write access to individual cells of an n-dimensional container. Index values passed to a sequence of array subscript operator calls will produce a proxy concept that can import to or export the primitive data the corresponding cell inside the container.

```cpp
auto image = make_n_container<MyStruct, layout::soa>(n_extent[128][256]);
auto acc = image.access();
MyStruct in_value(100.0f, 200.0f, 300.0f);
acc[64][128] = in_value;
MyStruct out_value = acc[64][128];
assert(out_value == in_value);
```

Accessors also know their valid iteration space, which can queried using the template function bound_d<int DimensionT>(accessor).

```cpp
assert(bounds_d<0>(acc) == bounds(0_fixed,128));
assert(bounds_d<1>(acc) == bounds(0_fixed,256));
```

An accessor may have a non-zero index space if it has a translation embedded into it, bounds_d will reflect any such translation.

```cpp
auto shifted_acc = acc.translated_to(n_index[1000][2000]);
assert(bounds_d<0>(shifted_acc) == bounds(1000,1128));
assert(bounds_d<1>(shifted_acc) == bounds(2000,2256));
```

This is useful to have a smaller sized container participate in a calculation over a portion of a larger index space, simplifying programming as the same index variable can be used, and the accessor takes care of applying the necessary translation. An accessor may represent a subsection over the original extents, bounds_d will identify the valid iteration space for that accessor.

```cpp
auto subsection_acc = a.section(n_bounds[bounds(64,96)][bounds(128,160)]);
assert(bounds_d<0>(subsection_acc) == bounds(64, 96));
assert(bounds_d<1>(subsection_acc) == bounds(128, 160));
```

It can also be useful to have subsections be translated back to start their iteration space at 0. For efficiency, the translated_to_zero() method is provided to create an accessor shifted back to zero.

```cpp
auto zb_sub_acc = a.section( n_bounds[bounds(64, 96)][bounds(128, 160)] ).translated_to_zero();
assert(bounds_d<0>(zb_sub_acc) == bounds(0, 32));
assert(bounds_d<1>(zb_sub_acc) == bounds(0, 32));
```

If fewer array subscript calls applied to an accessor than its rank, the result is another accessor of a lower rank. This can be useful to obtain accessors suitable to pass to code expecting lower rank accessors. Such as a obtaining a 3d accessor from a 4d container by specifying only a single index via array subscript. This has
the effect of embedding the index value of the dimension inside accessor. When the final dimension is sliced, the result is a proxy object to the cell inside the container corresponding to the embedded index values inside the sliced accessors

```cpp
auto image4d = make_n_container<MyStruct, layout::soa>(n_extent[10][20][128][256]);
MyStruct in_value(100.0f, 200.0f, 300.0f);
auto acc4d = image4d.access();
auto acc3d = acc4d[5];
auto acc2d = acc3d[10];
auto acc1d = acc2d[64];
acc1d[128] = in_value;
MyStruct out_value = acc4d[5][10][64][128];
assert(out_value == in_value);
```

The following table provides information on the requirements of the accessor concept.

<table>
<thead>
<tr>
<th>Pseudo-Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef PrimitiveT primitive_type;</td>
<td>Data type inside the cells of the container.</td>
</tr>
<tr>
<td>static constexpr int rank;</td>
<td>Number of free dimensions of accessor</td>
</tr>
<tr>
<td>accessor_concept(const accessor_concept &amp;a_other)</td>
<td>Effects: constructs a copy of another accessor of the exact same type</td>
</tr>
<tr>
<td>Requirements: rank == 1 and IndexT is one of: int, aligned&lt;AlignmentT&gt;, fixed&lt;NumberT&gt;, linear_index, or simd_index&lt;LaneCountT&gt;</td>
<td></td>
</tr>
<tr>
<td>Effects: When only 1 free dimension is left, the operator[] will construct an element_concept which is the proxy to the cell inside the container. If this accessor was obtained with const_access(), then the proxy will provide read only interface to the cell’s data.</td>
<td></td>
</tr>
<tr>
<td>Returns: The proxy object to cell inside the container corresponding to the position identified by the a_index along with any embedded index values for other dimensions</td>
<td></td>
</tr>
<tr>
<td>Requirements: rank &gt; 1 and IndexT is one of: int, aligned&lt;AlignmentT&gt;, fixed&lt;NumberT&gt;, linear_index, or simd_index&lt;LaneCountT&gt;</td>
<td></td>
</tr>
<tr>
<td>Effects: When 2 or more free dimensions are left, the operator[] will construct another accessor_concept of lower rank embeding a_index inside of it, effectively fixing that dimension’s index value for any accesses made through the returned accessor_concept.</td>
<td></td>
</tr>
<tr>
<td>Returns: The accessor_concept of lower rank (one less free dimension).</td>
<td></td>
</tr>
<tr>
<td>Requirements: DimensionT &gt;=0 and DimensionT &lt; rank</td>
<td></td>
</tr>
<tr>
<td>Pseudo-Signature</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| `auto bounds_dXX() const`                                                       | **Effects:** Determine the bounds of a free dimension using DimensionT as a 0 based index starting at the leftmost dimension.  
|                                                                                 | **Returns:** bounds_t of the DimensionT                                                        |
|                                                                                 | **Requirements:** XX >=0 and XX < rank and XX < 20                                             |
|                                                                                 | **Effects:** Non templated methods to determine the bounds of a free dimension using XX as a 0 based index starting at the leftmost dimension.  
|                                                                                 | **Returns:** bounds_t of the XX dimension                                                     |
| template<int DimensionT>                                                        | **Requirements:** DimensionT >=0 and DimensionT < rank                                           |
| `auto extent_d() const`                                                          | **Effects:** Determine the extent of a free dimension using DimensionT as a 0 based index starting at the leftmost dimension.  
|                                                                                 | **Returns:** extent of the DimensionT                                                         |
|                                                                                 | **Requirements:** XX >=0 and XX < rank and XX < 20                                             |
| auto extent_dXX() const                                                          | **Effects:** Non templated methods to determine the extent of a free dimension using XX as a 0 based index starting at the leftmost dimension.  
|                                                                                 | **Returns:** extent of the XX dimension                                                      |
| template<typename ...IndexListT>                                                | **Requirements:** a_n_index has same rank as the accessor                                       |
| accessor_concept translated_to(                                                 | **Effects:** construct an accessor_concept with an embedded translation such that accessing a_n_index will corresponds back to the current lower bounds. Easy way to think of it is that current iteration space is translated to a_n_index space.  
|     n_index_t<IndexListT...> a_n_index) const                                    | **Returns:** accessor_concept whose bounds have the same extents, but whose lower bounds start at the supplied a_n_index |
| template<typename ...IndexListT>                                                | **Effects:** construct an accessor_concept with an embedded translation such that accessing [0] index for all dimensions will corresponds back to the current lower bounds. Easy way to think of it is that current iteration space is translated to [0] for all free dimensions.  
| accessor_concept translated_to_zero() const                                     | **Returns:** accessor_concept whose bounds have the same extents, but whose lower bounds start at [0]... [0] |
**Proxy Objects**

accessors can't return a reference to the Primitive because its memory layout is abstracted. Instead a Proxy object is returned. That Proxy supports importing or exporting data to and from the Container. The actual type of Proxy objects is an implementation detail, but they all support the same public interface which we will document.

Each accessor [index] operator returns a Proxy object.

Each const_accessor [index] operator returns a ConstProxy object.

The Proxy objects provide a Data Member Interface where for each data member of value_type they are representing, a member access method is defined which returns a new Proxy or ConstProxy representing just that data member. Users can drill down through a complex data structure to get a Proxy representing the exact data member they need versus importing and exporting the entire Primitive value.

Proxy objects also overload the following operators if the underlying value_type supports the operator:

```c++
==, !=, <, >, <=, =>, +, -, *, /, %, &&, ||, &|, ^, ~, *=, /=, %=, >>, <<, &|=, |=, ^=, ++, --
```

**Proxy**

Proxy object provides access to a specific Primitive, Primitive data member, or nested data member within a Primitive for an element in a container.

**Description**

accessor [index] or a Proxy object's Data Member Interfaces return Proxy objects. That Proxy object represents the Primitive, Primitive data member, or nested data member within a Primitive for an element in a container. The Proxy object has the following features:

- A value_type can be exported or imported from the Proxy.
  - Conversion operator is used to export the value_type
  - Alternatively the Proxy can be passed to the function unproxy to export a value_type
  - Assignment operator = is used to import value_type into the Proxy
  - Overloads the following operators if the underlying value_type supports the operator
    ```c++
    ==, !=, <, >, <=, =>, +, -, *, /, %, &&, ||, &|, ^, ~, *=, /=, %=, >>, <<, &|=, |=, ^=, ++, --
    ```
  - When an operator is called the following occurs:
• value_type is exported
  • The operator applied to the exported value
  • If the operator was an assignment, the result is imported back into the Member and returns the proxy
  • Otherwise a result is returned.

• Data Member Interface.
  • For each data member of value_type
    • A member access method is defined which returns a Member proxy representing just that member.

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef implementation-defined value_type</td>
<td>The type of the data the Proxy is representing</td>
</tr>
<tr>
<td>Member</td>
<td>Description</td>
</tr>
<tr>
<td>operator value_type const () const;</td>
<td>Returns: exports a copy of the Proxy’s value.</td>
</tr>
<tr>
<td></td>
<td>NOTE: constant return value prevents rvalue assignment for structs offering some protection against code that expected a modifiable reference.</td>
</tr>
<tr>
<td>const value_type &amp; operator = (const value_type &amp;a_value);</td>
<td>Imports a_value into container at the position the Proxy is representing.</td>
</tr>
<tr>
<td></td>
<td>Returns: the same constant value_type it was passed.</td>
</tr>
<tr>
<td></td>
<td>NOTE: This behavior is different from traditional assignment operators that return *this. Choice was to enable efficient chaining of assignment operators versus returning a Proxy which would have to export the value it had just imported.</td>
</tr>
<tr>
<td>Proxy &amp; operator = (const Proxy &amp;other);</td>
<td>Exports value from the other Proxy and imports it.</td>
</tr>
<tr>
<td></td>
<td>Returns: A reference to this Proxy object.</td>
</tr>
<tr>
<td>auto name_of_values_data_member_1()const;</td>
<td>Returns: Proxy instance representing the 1st data member of the value_type</td>
</tr>
<tr>
<td></td>
<td>NOTE: actual method name is the name of the value_type’s 1st data member</td>
</tr>
<tr>
<td>auto name_of_values_data_member_2()const;</td>
<td>Returns: Proxy instance representing the 2nd data member of the value_type.</td>
</tr>
<tr>
<td></td>
<td>NOTE: actual method name is the name of the value_type’s 2nd data member.</td>
</tr>
<tr>
<td>auto name_of_values_data_member_...()const;</td>
<td>Returns: Proxy instance representing the ...th data member of the value_type.</td>
</tr>
<tr>
<td></td>
<td>NOTE: actual method name is the name of the value_type’s ...th data member.</td>
</tr>
<tr>
<td>auto name_of_values_data_member_N()const;</td>
<td>Returns: Proxy instance representing the Nth data member of the value_type.</td>
</tr>
<tr>
<td></td>
<td>NOTE: actual method name is the name of the value_type’s Nth data member.</td>
</tr>
</tbody>
</table>
**ConstProxy**

*ConstProxy object provides access to a specific constant primitive, primitive data member, or nested data member within a primitive for an element in a container.*

**Description**

`const_accessor [index]` or a ConstProxy object’s Data Member Interfaces return ConstProxy objects. That ConstProxy object represents the constant primitive, primitive data member, or nested data member within a primitive for an element in a container. The ConstProxy object has the following features:

- A value_type can be exported or imported from the ConstProxy.
  - Conversion operator is used to export the value_type
  - Alternatively the ConstProxy can be passed to the function `unproxy` to export a value_type
- Overloads the following operators if the underlying value_type supports the operator
  - `==, !=, <, <=, >=, +, -, *, /, %, &&, ||, &, |, ^, ~, *, +, -, !`
  - When an operator is called the following occurs:
    - value_type is exported
    - The operator applied to the exported value
    - returns the result.

- Data Member Interface.
  - For each data member of value_type
    - A member access method is defined which returns a Member ConstProxy representing just that member.

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>typedef implementation-defined value_type</code></td>
<td>The type of the data the ConstProxy is representing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
</table>
| `operator value_type const () const;` | Returns: exports a copy of the ConstProxy’s value.
NOTE: constant return value prevents rvalue assignment for structs offering some protection against code that expected a modifiable reference. |
| `auto name_of_values_data_member_1()const;` | Returns: ConstProxy instance representing the 1st data member of the value_type
NOTE: actual method name is the name of the value_type’s 1st data member |
| `auto name_of_values_data_member_2()const;` | Returns: ConstProxy instance representing the 2nd data member of the value_type.
NOTE: actual method name is the name of the value_type’s 2nd data member. |
| `auto name_of_values_data_member_...()const;` | Returns: ConstProxy instance representing the ...th data member of the value_type.
NOTE: actual method name is the name of the value_type’s ...th data member. |
Member Description

auto name_of_values_data_member_N()const; Returns: ConstProxy instance representing the Nth data member of the value_type. 
NOTE: actual method name is the name of the value_type’s Nth data member

Number Representation

When specifying extents, positions inside of, or bounds of a container, numeric values can be represented three different ways: fixed, aligned, and int. Fixed is most precise and int is least precise. It is advised to use as precise specification as possible. The compiler may optimize better with more information.

Fixed

Represent a numerical constant whose value specified at compile time.

```
template <int NumberT> class fixed;
```

If offsets applied to index values inside a SIMD loop are known at compile time, then the compiler can use that information. For example, to maintain aligned access, if boundary is fixed and known to be aligned when accessing underlying data layout. When multiple accesses are happening near each other, the compiler will have the opportunity to detect which accesses occur in the same cache lines and potentially avoid prefetching the same cache line repeatedly. Additionally, if the start of an iteration space is known at compile time, if it’s a multiple of the SIMD lane count, the compiler could skip generating a peel loop. Whenever possible, fixed values should be used over aligned or arbitrary integer values.

Although std::integral_constant<int> provides the same functionality, the library defines own type to provide overloaded operators and avoid collisions with any other code’s interactions with std::integral_constant<int>.

The following table provides information about the template arguments for fixed.

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int Number T</td>
<td>The numerical value the fixed will represent.</td>
</tr>
</tbody>
</table>

The following table provides information about the members of fixed.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static constexpr int value = NumberT</td>
<td>The numerical value known at compile-time.</td>
</tr>
<tr>
<td>constexpr operator value_type() const</td>
<td>Returns: The numerical value</td>
</tr>
<tr>
<td>constexpr value_type operator() const;</td>
<td>Returns: The numerical value</td>
</tr>
</tbody>
</table>

Constant expression arithmetic operators +,- (both unary and binary), * and / are defined for type sdlt::fixed<> and will be evaluated at compile-time.

The suffix _fixed is a C++11 user-defined equivalent literal. For example, 1080_fixed is equivalent to fixed<1080>. Consider the readability of the two samples below.

```
foo3d(fixed<1080>(), fixed<1920>());
```

versus

```
foo3d(1080_fixed, 1920_fixed);
```
NOTE This content is specific to C++; it does not apply to DPC++. The sdlt::fixed<NumberT> type supersedes the deprecated sdlt::fixed_offset<OffsetT> type found in SDLT v1. It is strongly advised to use sdlt::fixed<NumberT>. However, in this release, a template alias is provided mapping sdlt::fixed_offset<OffsetT> onto sdlt::fixed<NumberT>.

Aligned

Represent integer value known at compile time to be a multiple of an IndexAlignment.

```cpp
template <int IndexAlignmentT> class aligned;
```

If you can tell the compiler that you know that an integer will be a multiple of known value, then, when combined with a loop index inside a SIMD loop, the compiler can use that information to maintain aligned access when accessing underlying data layout.

Internally, the integer value is converted to a block count, where:

```cpp
block_count = value/IndexAlignmentT;
```

Overloaded math operations can then use that aligned block count as needed. The value() is represented by AlignmentT*block_count allowing the compiler to easily prove that the value() is a multiple of AlignmentT, which can utilize alignment optimizations.

The following table provides information about the template arguments for aligned.

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int IndexAlignmentT</td>
<td>The alignment the user is stating that the number is a multiple of. IndexAlignmentT must be a power of two.</td>
</tr>
</tbody>
</table>

The following table provides information about the types defined as members of aligned.

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef int value_type</td>
<td>The type of the numerical value.</td>
</tr>
<tr>
<td>typedef int block_type</td>
<td>The type of the block_count.</td>
</tr>
</tbody>
</table>

The following table provides information about the members of aligned.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int index_alignment</td>
<td>The IndexAlignmentT value.</td>
</tr>
<tr>
<td>aligned()</td>
<td>Constructs empty (uninitialized) object</td>
</tr>
<tr>
<td>explicit aligned(value_type)</td>
<td>Constructs computing block_count=a_value/IndexAlignmentT.</td>
</tr>
<tr>
<td>aligned(const aligned&amp; a_other)</td>
<td>Constructs copying block_count from a_other. a_other must have same IndexAlignmentT.</td>
</tr>
<tr>
<td>template&lt;int OtherAlignment&gt; explicit aligned(const aligned&amp; other)</td>
<td>Constructs computing block_count optimized by avoiding computing other.value(). Must have IndexAlignmentT of a_other &lt; IndexAlignmentT and other.value() be multiple of IndexAlignmentT.</td>
</tr>
</tbody>
</table>
The following operations are supported for the \texttt{aligned} type.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{operator *(int), commutative}</td>
<td>Scale value. \textbf{Returns:} \texttt{aligned&lt;IndexAlignmentT&gt; * \text{int}}</td>
</tr>
<tr>
<td>\texttt{operator *(fixed&lt;V&gt;), commutative}</td>
<td>Scales \texttt{IndexAlignment} by \texttt{2^M} and value by \texttt{K}. Must have \texttt{V=2^M<em>K} (\texttt{V} is a multiple of a power of \texttt{2}). \textbf{Returns:} \texttt{aligned&lt;IndexAlignmentT</em>(2^M)&gt;}</td>
</tr>
<tr>
<td>\texttt{operator *(aligned&lt;OtherAl&gt;)}</td>
<td>Scales \texttt{IndexAlignment} by \texttt{OtherAl} and \texttt{block_count} by argument. \textbf{Returns:} \texttt{aligned&lt;IndexAlignmentT*OtherAl&gt;}</td>
</tr>
<tr>
<td>\texttt{int operator/(fixed&lt;IndexAlignmentT&gt;)}</td>
<td>\textbf{Returns:} \texttt{aligned_block_count()}}</td>
</tr>
<tr>
<td>\texttt{int operator/(fixed&lt;-IndexAlignmentT&gt;)}</td>
<td>\textbf{Returns:} \texttt{-aligned_block_count();}}</td>
</tr>
<tr>
<td>\texttt{int operator/(fixed&lt;V&gt;)}</td>
<td>Must have \texttt{abs(V)&gt;IndexAlignmentT &amp;&amp; IndexAlignmentT%V==0.}} \textbf{Returns:} \texttt{aligned_block_count()/(V/ IndexAlignmentT)}}</td>
</tr>
<tr>
<td>\texttt{int operator/(fixed&lt;V&gt;)}</td>
<td>Must have \texttt{abs(V) &lt; IndexAlignmentT &amp;&amp; V %IndexAlignmentT==0}} \textbf{Returns:} \texttt{aligned_block_count()*(IndexAlignmentT/V)}}</td>
</tr>
<tr>
<td>\texttt{aligned operator -()}</td>
<td>\textbf{Returns:} Same type \texttt{aligned} for negated value.</td>
</tr>
<tr>
<td>\texttt{aligned operator -(const aligned &amp;)}</td>
<td>\textbf{Returns:} Same type \texttt{aligned} for value of difference.</td>
</tr>
<tr>
<td>Operation</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| template<int OtherAl> aligned<?> operator -(const aligned<OtherAl>&) const | Difference with other alignment. Behavior and returned alignment type depend on relation between alignments of operands.  
**Returns:** Value for difference as lower of incoming alignments |
| template<int V> aligned<?> operator -(const fixed<V> &) const | Difference with fixed value. Behavior and returned alignment type depend on relation between alignments of aligned<> operand and the value of V.  
**Returns:** Adjusted aligned value of a difference |
| aligned operator +(const aligned &)const | Sum with other alignment. Behavior and returned alignment type depend on relation between alignments of operands.  
**Returns:** Value for sum as lower of incoming alignments |
| template<int V> aligned<?> operator +(const fixed<V> &) const | Sum with fixed value. Behavior and returned alignment type depend on relation between alignments of aligned<> operand and the value of V.  
**Returns:** Adjusted aligned value of a sum. |
| template<int OtherAl> aligned operator +=(const aligned<OtherAl> &) const | Increments value for the aligned object if IndexAlignmentT is compatible with OtherAl  
**Returns:** Aligned with incremented value. |
| template<int OtherAl> aligned operator -= (const aligned<OtherAl> &) const | Decrements value for the aligned object if IndexAlignmentT is compatible with OtherAl  
**Returns:** Same type aligned with decremented value. |
| template<int OtherAl> aligned operator *= (const aligned<OtherAl> &) const | Multiplies value for the aligned object if IndexAlignmentT is compatible with OtherAl.  
**Returns:** Same type aligned with multiplied value. |
| template<int OtherAl> aligned operator /= (const aligned<OtherAl> &) const | Divides value for the aligned object if IndexAlignmentT is compatible with OtherAl.  
**Returns:** Same type aligned with divided value. |

**NOTE** This content is specific to C++; it does not apply to DPC++. The `sdlt::aligned<>` type supersedes the deprecated `sdlt::aligned_offset<>` type found in SDLT v1. It is strongly advised to use `sdlt::aligned<>`, however in this release a template alias is provided mapping `sdlt::aligned_offset<>` onto `sdlt::aligned<>`. 

418
int
Represents an arbitrary integer value. In interfaces where fixed<> and aligned<> values supported you may also use plain old integer value. It provides least information among these three and so least facilitates compiler optimizations.

aligned_offset
Represent an integer based offset whose value is a multiple of an IndexAlignment specified at compile time. #include <sdlt/aligned_offset.h>

Syntax

    template<int IndexAlignmentT>
    class aligned_offset;

Arguments

int IndexAlignmentT
    The index alignment the user is stating that the offset have.

Description

aligned_offset is a deprecated feature.

If we can tell the compiler that we know an offset will be a multiple of known value, then when combined with a loop index inside a SIMD loop, the compiler can use that information to maintain aligned access when accessing underlying data layout.

Internally, the offset value is converted to a block count.

    Block Count = offsetValue/IndexAlignmentT;

Indices can then use that aligned block count as needed.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static const int IndexAlignment =</td>
<td></td>
</tr>
<tr>
<td>IndexAlignmentT;</td>
<td>The alignment the offset is a multiple of</td>
</tr>
<tr>
<td>explicit aligned_offset(const int offset)</td>
<td>Construct instance based on offset</td>
</tr>
<tr>
<td>static aligned_offset from_block_count(int</td>
<td></td>
</tr>
<tr>
<td>aligned_block_count);</td>
<td>Returns: Instance based on aligned_block_count,</td>
</tr>
<tr>
<td></td>
<td>where the offset value =</td>
</tr>
<tr>
<td></td>
<td>IndexAlignment*aligned_block_count</td>
</tr>
<tr>
<td>int aligned_block_count() const;</td>
<td>Returns: number of blocks of IndexAlignment it</td>
</tr>
<tr>
<td></td>
<td>takes to represent the offset value.</td>
</tr>
<tr>
<td>int value() const;</td>
<td>Returns: offset value</td>
</tr>
</tbody>
</table>

fixed_offset
Represent an integer based offset whose value specified at compile time. #include <sdlt/ fixed_offset.h>

Syntax

    template <int OffsetT> fixed_offset;
Arguments

int OffsetT

The value the fixed_offset will represent

Description

defined Offset is a deprecated feature.

If we can tell the compiler that we know an offset at compile time, then when combined with a loop index inside a SIMD loop, the compiler can use that information to maintain aligned access (should the offset be aligned) when accessing underlying data layout. When multiple accesses are happening near each other, the compiler will have the opportunity to detect which accesses occur in the same cache lines and potentially avoid prefetching the same cache line repeatedly. Whenever possible, a fixed_offset should be used over an aligned_offset or integer based offset.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static constexpr int value = OffsetT</td>
<td>The offset value known at compile</td>
</tr>
</tbody>
</table>

Indexes

soa1d_container's and aos1d_container's accessors [] operator can accept an integer based loop index. However if any modifications were applied to that loop index, the fact that it's a loop index may be lost by the compiler as it is handled before being passed to the [] operator.

To avoid this situation, SDLT provides classes to wrap loop indexes that capture multiple additions or subtractions of offsets (see the Offsets section). The resulting index can be passed to [] and preserve the original loop index and track any arithmetic with Offsets to be applied to underlying data layout.

It is common for stencil based algorithms to need to apply offsets during data access.

For a regular linear loop, use linear_index to wrap your loop index.

linear_index

Wraps an integer-based loop index that is iterating linearly through an iteration space. #include <sdlt/linear_index.h>

Syntax

    class linear_index;

Description

Inside of a linear loop, wrap the loop index with a linear_index to allow addition or subtraction of offsets.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit linear_index(int an_index);</td>
<td>Construct instance from a loop index</td>
</tr>
<tr>
<td>int value() const;</td>
<td>Returns the original loop index</td>
</tr>
</tbody>
</table>

n_index_t (needs new content)

Variadic template class n_index_t describes a position inside of the N-dimensional container. Specifically, the number of dimensions and the of index value of each.

Syntax

    template<typename... TypeListT>
    class n_index_t
Description

\texttt{n\_index\_t} represents a position inside an \textit{n}-dimensional space as a sequence of index value for each dimension. The index of each dimension can be represented by different types. This flexibility allows the same interface to be used to declare \texttt{n\_index\_t} with indices that are fully known at compile time with \texttt{fixed<int NumberT>}, or to be only known at runtime with \texttt{int}, or only known at runtime but with a guarantee will be a multiple of an alignment with \texttt{aligned<int Alignment>}. For more details, see the Number representation section.

Objects of this class may be used to identify a cell in a container, describe the inclusive lower bounds for \texttt{n\_bounds()}, \textit{n}-dimensional position for accessor's \textit{translated\_to()}.

The following table provides information about the template arguments for \texttt{n\_index\_t}.

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{typename... TypeListT}</td>
<td>Comma separated list of types, where the number of types provided controls how many dimensions there are. Each type in the list identifies how the index of the corresponding dimension is to be represented. The order of the dimensions is the same order as C++ subscripts declaring a multi-dimensional array, from leftmost to rightmost.</td>
</tr>
<tr>
<td>Requirements: Type must be \texttt{int}, or \texttt{fixed&lt;NumberT&gt;}, or \texttt{aligned&lt;AlignmentT&gt;}.</td>
<td></td>
</tr>
</tbody>
</table>

The following table provides information about the members of \texttt{n\_index\_t}.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{static constexpr int rank;}</td>
<td>Number of dimensions.</td>
</tr>
<tr>
<td>\texttt{static constexpr int row_dimension = rank-1;}</td>
<td>Index of last dimension, \textit{row}.</td>
</tr>
<tr>
<td>\texttt{n_index_t()}</td>
<td>Default constructor. Uses default values for extent types.</td>
</tr>
<tr>
<td>\texttt{Requirements: Every type in TypeListT is default constructible.}</td>
<td></td>
</tr>
<tr>
<td>\texttt{Effects: Construct n_index_t, uses default values of each type in TypeListT for the dimensions sizes. In general only correctly initialized when every type is a fixed&lt;NumberT&gt;.}</td>
<td></td>
</tr>
<tr>
<td>\texttt{n_index_t(const n_extent_t &amp;a_other)}</td>
<td>Copy constructor.</td>
</tr>
<tr>
<td>\texttt{Effects: Construct n_index_t, copying index value of each dimension from a_other.}</td>
<td></td>
</tr>
<tr>
<td>\texttt{explicit n_index_t(const TypeListT &amp; ... a_values)}</td>
<td>Returns: The last extent in its native type</td>
</tr>
<tr>
<td>\texttt{Effects: Construct n_index_t, initializing each dimension with the corresponding value from the list of a_values passed as an argument. In use, a_values is a comma separate list of values whose length and types are defined by TypeListT.}</td>
<td></td>
</tr>
</tbody>
</table>
### Member Description

**template<int DimensionT>
auto get() const**

**Requirements:** DimensionT >= 0 and DimensionT < rank.

**Effects:** Determine the index value of DimensionT.

**Returns:** In the type declared by the DimensionT position of 0-based TypeListT, the index value of the specified DimensionT

**n_index_t operator +() const**

**Effects:** Determine the positive unary value of each dimension's index, effectively no operation is performed

**Returns:** Copy of the current instance.

**auto operator -() const**

**Effects:** Determine the negative unary value of each dimension's index

**template<class... OtherTypeListT>
auto operator +(const n_index_t<OtherTypeListT...> & a_other) const**

**Requirements:** Rank of a_other is the same as this instance's.

**Effects:** Build n_index_t whose values are the result of adding the index value for each dimension with those of a_other

**Returns:** n_index[get<0>() + a_other.get<0>()]
[get<1>() + a_other.get<1>()]
[get<...>() + a_other.get<...>()]
[get<row_dimension>() + a_other.get<row_dimension>()]

**template<class... OtherTypeListT>
auto operator -(const n_index_t<OtherTypeListT...> & a_other) const**

**Requirements:** Rank of a_other is the same as this instance's.

**Effects:** Build n_index_t whose values are the result of subtracting the index value for each dimension of a_other with this instance's.

**Returns:** n_index[get<0>() - a_other.get<0>()]
[get<1>() - a_other.get<1>()]
[get<...>() - a_other.get<...>()]
[get<row_dimension>() - a_other.get<row_dimension>()]

**template<class... OtherTypeListT>
bool operator ==(const n_index_t<OtherTypeListT...> a_other) const**

**Requirements:** Rank of a_other is the same as this instance's.
**template<class... OtherTypeListT>**

**bool operator != (const n_index_t<OtherTypeListT...> a_other) const**

**Requirements:** Rank of a_other is the same as this instance's.

**Effects:** Compare index of each dimension for inequality. Only compares numeric values, not the types of each dimension.

**Returns:** true if any dimensions are numerically different, false otherwise.

**template<int DimensionT>**

**auto rightmost_dimensions() const**

**Requirements:** DimensionT >= 0 and DimensionT <= rank.

**Effects:** Construct a n_index_t with a lower rank by copying the rightmost DimensionT values from this instance.

**Returns:** n_index[get<rank - DimensionT>()] [get<rank + 1 - DimensionT>()] [get<...>()] [get<row_dimension>()]

**template<class... OtherTypeListT>**

**auto overlay_rightmost(const n_index_t<OtherTypeListT...> & a_other) const**

**Requirements:** rank of a_other is <= rank

**Effects:** Construct copy of n_index_t where the rightmost dimensions' values are copied from a_other, effectively overlaying a_other ontop of rightmost dimensions of this instance.

**Returns:** n_index[get<0>()] [get<1>()] [get<...>()] [get<rank-a_other::rank>()] [a_other.get<0>()] [a_other.get<...>()] [a_other.get<a_other::row_dimension>()]

The following table provides information about the friend functions of n_index_t

<table>
<thead>
<tr>
<th>Friend Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| std::ostream& operator << (std::ostream& output_stream, const n_index_t & a_indices) | **Effects:** Append string representation of a_indices' values to a_output_stream.  
**Returns:** Reference to a_output_stream for chained calls. |
**n_index_generator**

To facilitate simpler creation of \( n\_index\_t \) objects, the generator object \( n\_index \) is provided.

**Syntax**

```cpp
template<typename... TypeListT>
class n_index_generator;
namespace {
    // Instance of generator object
    n_index_generator<> n_index;
}
```

**Description**

The generator object provides recursively constructing operators \([\])\ for \( \text{fixed<>>}, \text{aligned<>>}, \) and integer values allowing building of a \( n\_index\_t<...> \) instance one dimension at a time. Its main purpose is to allow a usage syntax that is similar to C multi-dimensional array definition.

Compare the following examples, instantiating three \( n\_index\_t \) instances, and using the generator object to instantiate equivalent instances.

```cpp
n_index_t<int, int> idx1(row, col);
n_index_t<int, aligned<16>> idx2(row, aligned<16>(col));
n_index_t<fixed<540>, fixed<960>> idx3(540_fixed, 960_fixed);
```

```cpp
auto idx1 = n_index[row][col];
auto idx2 = n_index[row][aligned<16>(col)];
auto idx3 = n_index[540_fixed][960_fixed];
```

**Class Hierarchy**

It is expected that \( n\_index\_generator<...> \) not be directly used as a data member or parameter, instead only \( n\_index\_t<...> \) from which it is derived. The generator object \( n\_index \) can be automatically downcast any place expecting an \( n\_index\_t<...> \).

The following table provides the template arguments for \( n\_index\_generator \)

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typename... TypeListT</td>
<td>Comma separated list of types, where the number of types provided controls how many dimensions the generator currently represents. Each type in the list identifies how the size of the corresponding dimension is to be represented. The order of the dimensions is the same order as C++ subscripts declaring a multi-dimensional array – from leftmost to rightmost. <strong>Requirements</strong>: Type is ( \text{int, fixed&lt;NumberT&gt;}, ) or ( \text{aligned&lt;AlignmentT&gt;} ).</td>
</tr>
</tbody>
</table>

The following table provides information on the types defined as members of \( n\_index\_generator \) in addition to those inherited from \( n\_index\_t \).

<table>
<thead>
<tr>
<th>Member Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef ( n_index_t&lt;\text{TypeListT}...&gt; ) value_type</td>
<td>Type value that the any chained ([])\ operator calls have produced.</td>
</tr>
</tbody>
</table>
The following table provides information on the members of `n_index_generator` in addition to those inherited from `n_index_t`

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>n_index_generator()</code></td>
<td>Requirements: TypeListT is empty. Effects: Construct generator with no indices specified.</td>
</tr>
<tr>
<td><code>n_index_generator(const n_index_generator &amp;a_other)</code></td>
<td>Effects: Construct generator copying any index values from <code>a_other</code></td>
</tr>
<tr>
<td><code>n_index_generator&lt;TypeListT..., int&gt; operator[](int a_index) const</code></td>
<td>Requirements: <code>a_size</code> &gt;= 0. Returns: <code>n_index_generator&lt;...&gt;</code> with additional rightmost integer based index.</td>
</tr>
<tr>
<td><code>n_index_generator&lt;TypeListT..., fixed&lt;NumberT&gt;&gt; operator[](fixed&lt;NumberT&gt; a_index) const</code></td>
<td>Requirements: <code>a_size</code> &gt;= 0. Returns: <code>n_index_generator&lt;...&gt;</code> with additional rightmost fixed&lt;NumberT&gt; index.</td>
</tr>
<tr>
<td><code>n_index_generator&lt;TypeListT..., aligned&lt;AlignmentT&gt;&gt; operator[](aligned&lt;AlignmentT&gt; a_index)</code></td>
<td>Requirements: <code>a_size</code> &gt;= 0 Returns: <code>n_index_generator&lt;...&gt;</code> with additional rightmost aligned&lt;AlignmentT&gt; based index.</td>
</tr>
<tr>
<td><code>value_type value() const</code></td>
<td>Returns: <code>n_extent_t&lt;...&gt;</code> with the correct types and values of the multi-dimensional extents aggregated by the generator.</td>
</tr>
</tbody>
</table>

**index_d template function**

**Syntax**

```
template<int DimensionT, typename ObjT>
auto index_d(const ObjT &a_obj)
```

**Description**

The template function offers a consistent way to determine the index of a dimension for a multi-dimensional object. It can avoid extracting an entire `n_index_t<...>` when only the extent of a single dimension is needed.

<table>
<thead>
<tr>
<th>Template Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int DimensionT</code></td>
<td>0 based index starting at the leftmost dimension indicating which n-dimensions to query the index of. Requirements: <code>DimensionT</code> &gt;= 0 and <code>DimensionT</code> &lt; <code>ObjT::rank</code></td>
</tr>
<tr>
<td><code>typename ObjT</code></td>
<td>The type of n-dimensional object from which to retrieve the extent. Requirements: <code>ObjT</code> is one of: <code>n_index_t&lt;...&gt;</code> <code>n_index_generator&lt;...&gt;</code></td>
</tr>
</tbody>
</table>

**Returns**
The correctly typed index corresponding to the requested DimensionT of a_obj.

**Example**

```cpp
template <typename IndicesT>
void foo(const IndicesT & a_pos)
{
    int z = index_d<0>(a_pos);
    int y = index_d<1>(a_pos);
    int x = index_d<2>(a_pos);
    //...
}
```

**Convenience and Correctness**

Users can include a single header file `sdlt.h` that includes all the supported public features, or users can include the individual headers of features they will be using (which might build faster). In other words,

```cpp
#include <sdlt/sdlt.h>
```

instead of

```cpp
#include <sdlt/primitive.h>
#include <sdlt/soa1d_container.h>
```

For convenience, SDLT provides a macro to encapsulate `#pragma forceinline recursive`.

```cpp
SDLT_INLINE_BLOCK
```

SDLT reduces overhead by trusting the programmer to pass it valid values for template and function parameters. Adding conditional checks inside of a SIMD loop can cause unnecessary code generation and inhibit vectorization by creating multiple exit points in a loop. To assist in verifying that a program is indeed passing valid values to SDLT, the programmer can add a compilation flag to their build to define `SDLT_DEBUG=1`.

```cpp
-DSDLT_DEBUG=1
```

If `__DEBUG` is defined and `SDLT_DEBUG` has not been defined to 0 or 1, then `SDLT_DEBUG` is automatically set to 1. When set to 1, every operator[] is bounds checked and all addresses are validated for correct alignment. It is very useful for tracking down any usage bugs.

The macro `__SDLT_VERSION` is predefined to be 2001. Programs could use it for conditional compilation if incompatibilities arise in future updates.

C++ implementations of `std::min` and `std::max` sometimes have a negative impact on performance. SDLT defines `min_val` and `max_val` that help avoid such performance penalties.

**max_val**

*Return the right value if the right value is greater than left, otherwise returns the left value.*  
`#include <sdlt/min_max_val.h>`

**Syntax**

```cpp
template<typename T>
T max_val(const T left, const T right);
```

**Arguments**

- `typename T`: The type of the left and right values
Description

C++ implementations of std::min and std::max create a conditional control flow that returns references to its parameters, which may cause inefficient vector code generation. max_val is a really simple template that returns by value instead of reference, allowing more efficient vector code to be generated. For most cases the algorithm didn’t need a reference to the inputs and a copy by value should suffice. It should inline, adding no overhead. Inside of SIMD loops, we suggest using sdlt::max_val in place of std::max.

Requires < operator be defined for the type T.

min_val
Return the left value if the right value is greater than left, otherwise returns the right value. #include <sdlt/min_max_val.h>

Syntax

    template<typename T>
    T min_val(const T left, const T right);

Arguments

typename T
The type of the left and right values

Description

C++ implementations of std::min and std::max create a conditional control flow that returns references to its parameters, which may cause inefficient vector code generation. min_val is a really simple template that returns by value instead of reference, allowing more efficient vector code to be generated. For most cases the algorithm didn’t need a reference to the inputs and a copy by value should suffice. It should inline, adding no overhead. Inside of SIMD loops, we suggest using sdlt::min_val in place of std::min.

Requires < operator be defined for the type T.

Examples

The example programs in this section demonstrate the following:

- The efficiency of using SDLT and its Structure of Arrays approach rather than a typical Array of Structures
- Construction of more complex SDLT primitives
- Performance improvement in case of a forward-dependency
- Use of offsets and calling methods on the SDLT primitive
- RGB to YUV conversion

Example 1

Example 1 demonstrates the efficiency of using a Structure of Arrays (SoA) approach by comparing the assembly generated from a simple SIMD loop using an Array of Structures (AoS) approach with the assembly generated using the SoA approach of SDLT.

Array of Structures: Non-unit stride access version

Source:

    #include <stdio.h>
    #define N 1024
    typedef struct RGBs {
        float r;
        float g;
    }
float b;
) RGBTy;

void main()
{
    RGBTy a[N];
    #pragma omp simd
    for (int k = 0; k<N; ++k) {
        a[k].r = k*1.5;  // non-unit stride access
        a[k].g = k*2.5;  // non-unit stride access
        a[k].b = k*3.5;  // non-unit stride access
    }
    std::cout << "k =" << 10 << ", a[k].r =" << a[10].r << ", a[k].g =" << a[10].g << ", a[k].b =" << a[10].b << std::endl;
}

AVX2 assembly generated (69 instructions):

..TOP_OF_LOOP:
    vcvtdq2ps $ymm7, $ymm1
    lea       (%rax), %rcx
    vcvtdq2ps $ymm5, %ymm2
    vpadd    %ymm3, %ymm7, %ymm7
    vpadd    %ymm3, %ymm5, %ymm5
    vmulps    %ymm1, %ymm4, %ymm8
    vmulps    %ymm1, %ymm6, %ymm12
    vmulps    %ymm2, %ymm6, %ymm14
    vmulps    %ymm1, %ymm0, %ymm1
    vmulps    %ymm2, %ymm4, %ymm10
    addl      $16, %edx
    vextractf128 $1, %ymm8, %xmm9
    vmovss    %xmm8, (%rsp,%rcx)
    vmovss    %xmm9, 48(%rsp,%rcx)
    vextractps $1, %xmm8, 12(%rsp,%rcx)
    vextractps $2, %xmm8, 24(%rsp,%rcx)
    vextractps $3, %xmm8, 36(%rsp,%rcx)
    vmulps    %ymm2, %ymm0, %ymm8
    vextractps $1, %ymm9, 60(%rsp,%rcx)
    vextractps $2, %ymm9, 72(%rsp,%rcx)
    vextractps $3, %ymm9, 84(%rsp,%rcx)
    vextractf128 $1, %ymm12, %xmm13
    vextractf128 $1, %ymm12, %xmm13
    vextractf128 $1, %ymm12, %xmm13
    vextractf128 $1, %ymm12, %xmm13
    vextractf128 $1, %ymm12, %xmm13
    vmovss    %xmm12, 4(%rsp,%rax)
    vmovss    %xmm13, 52(%rsp,%rax)
    vextractps $1, %xmm12, 16(%rsp,%rax)
    vextractps $2, %xmm12, 28(%rsp,%rax)
    vextractps $3, %xmm12, 40(%rsp,%rax)
    vextractps $1, %xmm13, 64(%rsp,%rax)
    vextractps $2, %xmm13, 76(%rsp,%rax)
    vextractps $3, %xmm13, 88(%rsp,%rax)
    vmovss    %ymm14, 100(%rsp,%rax)
    vextractps $1, %xmm14, 112(%rsp,%rax)
    vextractps $2, %xmm14, 124(%rsp,%rax)
    vextractps $3, %xmm14, 136(%rsp,%rax)
vmovss  %xmm15, 148(%rsp,%rax)
vecextractps $1, %xmm15, 160(%rsp,%rax)
vecextractps $2, %xmm15, 172(%rsp,%rax)
vecextractps $3, %xmm15, 184(%rsp,%rax)
vmovss  %xmm1,  8(%rsp,%rax)
vecextractps $1, %xmm1,  20(%rsp,%rax)
vecextractps $2, %xmm1,  32(%rsp,%rax)
vecextractps $3, %xmm1,  44(%rsp,%rax)
vmovss  %xmm2,  56(%rsp,%rax)
vecextractps $1, %xmm2,  68(%rsp,%rax)
vecextractps $2, %xmm2,  80(%rsp,%rax)
vecextractps $3, %xmm2,  92(%rsp,%rax)
vmovss  %xmm8, 104(%rsp,%rax)
vecextractps $1, %xmm8, 116(%rsp,%rax)
vecextractps $2, %xmm8, 128(%rsp,%rax)
vecextractps $3, %xmm8, 140(%rsp,%rax)
vmovss  %xmm9, 152(%rsp,%rax)
vecextractps $1, %xmm9, 164(%rsp,%rax)
vecextractps $2, %xmm9, 176(%rsp,%rax)
vecextractps $3, %xmm9, 188(%rsp,%rax)
addq      $192, %rax
vecextractf128 $1, %ymm10, %xmm11
vmovss  %ymm10, 96(%rsp,%rcx)
vmovss  %ymm11, 144(%rsp,%rcx)
vecextractps $1, %ymm10, 108(%rsp,%rcx)
vecextractps $2, %ymm10, 120(%rsp,%rcx)
vecextractps $3, %ymm10, 132(%rsp,%rcx)
vecextractps $1, %ymm11, 156(%rsp,%rcx)
vecextractps $2, %ymm11, 168(%rsp,%rcx)
vecextractps $3, %ymm11, 180(%rsp,%rcx)
cmpl      $1024, %edx
jb        ..TOP_OF_LOOP

Structure of Arrays: Using SDLT for unit stride access

To introduce the use of SDLT, the code below will:

- declares a primitive,
- use an `soa1d_container` instead of an array,
- use an accessor inside a SIMD loop to generate efficient code,
- and use a proxy object’s data member interface to access individual data members of an element inside the container.

Source:

```c
#include <stdio.h>
#include <sdlt/sdlt.h>

#define N 1024

typedef struct RGBs {
    float r;
    float g;
    float b;
} RGBTy;

SDLT_PRIMITIVE(RGBTy, r, g, b)

void main()
```
// Use SDLT to get SOA data layout
sdlt::soal_cont<RGBTy> aContainer(N);
auto &a = aContainer.access();

// use SDLT Data Member Interface to access struct members r, g, and b.
// achieve unit-stride access after vectorization
#pragma omp simd
for (int k = 0; k < N; ++k) {
    a[k].r() = k * 1.5;
    a[k].g() = k * 2.5;
    a[k].b() = k * 3.5;
}
std::cout << "k = " << 10 << ", a[k].r = " << a[10].r() << ", a[k].g = " << a[10].g() << ", a[k].b = " << a[10].b() << std::endl;

AVX2 assembly generated (19 instructions):

AVX2 assembly generated (19 instructions):

Both versions appear to have unfolded the loop twice. When examining the assembly generated for AVX2
instruction set, we can see a measurable reduction in the number of instructions (19 vs. 69) when we are
able to perform unit stride access using SDLT. Also, at runtime, the soal_cont aligned its data
allocation and will gain any of the architectural advantages that come with using aligned instead of unaligned
SIMD stores.

Example 2

Example 2 demonstrates use of nested primitives and the use of an accessor inside a SIMD loop to generate
efficient code.

#include <stdio.h>
#include <sdlt/sdlt.h>

#define N 1024

typedef struct XYZs {
    float x;
}
float y;
float z;
) XYZTy;

SDLT_PRIMITIVE(XYZTy, x, y, z)

typedef struct RGBs {
  float r;
  float g;
  float b;
  XYZTy w;
} RGBTy;

SDLT_PRIMITIVE(RGBs, r, g, b, w)

void main()
{
  sdlt::soa1d_container<RGBTy> aContainer(N);
  auto a = aContainer.access();

  #pragma omp simd
  for (int k = 0; k<N; k++) {
    RGBTy c;
    c.r = k*1.5f;
    c.g = k*2.5f;
    c.b = k*3.5f;
    c.w.x = k*4.5f;
    c.w.y = k*5.5f;
    c.w.z = k*6.5f;
    a[k] = c;
  }
  const RGBTy c = a[10];
  printf("k = %d, a[k].r = %f, a[k].g = %f, a[k].b = %f \n",
         10, c.r, c.g, c.b);
  printf("k = %d, a[k].w.x = %f, a[k].w.y = %f, a[k].w.z = %f \n",
         10, c.w.x, c.w.y, c.w.z);
}

Example 3
Example 3 demonstrates the declaration of a Structure of Arrays (SoA) interacting with a forward dependency.

#include <stdio.h>
#include <sdlt/primitive.h>
#include <sdlt/soa1d_container.h>

#define N 1024

typedef struct RGBs {
  float r;
  float g;
  float b;
) RGBTy;

SDLT_PRIMITIVE(RGBTy, r, g, b)

void main()
{
// RGBTy a[N]; // AOS data layout
sdlt::soa1d_container<RGBTy> aContainer(N);
auto a = aContainer.access();    // SOA data layout

// use SDLT access method to access struct members r, g, and b.
// with unit-stride access after vectorization
#pragma omp simd
for (int k = 0; k<N; k++) {
a[k].r() = k*1.5;
a[k].g() = k*2.5;
a[k].b() = k*3.5;
}

// Test forward-dependency on SOA memory access
#pragma omp simd
for (int i = 0; i<N - 1; i++) {
    sdlt::linear_index k(i);
    a[k].r() = a[k + 1].r() + k*1.5;
    a[k].g() = a[k + 1].g() + k*2.5;
    a[k].b() = a[k + 1].b() + k*3.5;
}
std::cout << "k =" << 10 << "r =" << a[10].r() << "g =" << a[10].g() << "b =" << a[10].b() << std::endl;

Example 4
Example 4 demonstrates a linearized 2d stencil using embedded offsets and calling methods on the primitive.

#include <sdlt/sdlt.h>

// Typical C++ object to represent a pixel in an image
struct RGBs
{
  float red;
  float green;
  float blue;

  RGBs() {} 
  RGBs(const RGBs &iOther)
  : red(iOther.red)
  , green(iOther.green)
  , blue(iOther.blue)
  {
  }

  RGBs & operator =(const RGBs &iOther)
  {
    red = iOther.red;
    green = iOther.green;
    blue = iOther.blue;
    return *this;
  }

  RGBs operator + (const RGBs &iOther) const
  {
  }
RGBs sum;
    sum.red = red + iOther.red;
    sum.green = green + iOther.green;
    sum.blue = blue + iOther.blue;
    return sum;
}

RGBs operator * (float iScalar) const
{
    RGBs scaledColor;
    scaledColor.red = red * iScalar;
    scaledColor.green = green * iScalar;
    scaledColor.blue = blue * iScalar;
    return scaledColor;
};

SDLT_PRIMITIVE(RGBs, red, green, blue)

const int StencilHaloSize = 1;
const int width = 1920;
const int height = 1080;

template<typename AccessorT> void loadImageStub(AccessorT) {}
template<typename AccessorT> void saveImageStub(AccessorT) {}

// performs average color filtering with neighbors left,right,above,below
void main(void)
{
    // We are padding +1 so we can avoid boundary conditions
    const int paddedWidth = width + 2 * StencilHaloSize;
    const int paddedHeight = height + 2 * StencilHaloSize;
    int elementCount = paddedWidth*paddedHeight;
    sdlt::soa1d_container<RGBs> inputImage(elementCount);
    sdlt::soa1d_container<RGBs> outputImage(elementCount);

    loadImageStub(inputImage.access());

    SDLT_INLINE_BLOCK
    {
        const int endOfY = StencilHaloSize + height;
        const int endOfX = StencilHaloSize + width;
        for (int y = StencilHaloSize; y < endOfY; ++y)
        {
            // Embed offsets into Accessors to get the to correct row
            auto prevRow = inputImage.const_access((y - 1)*paddedWidth);
            auto curRow = inputImage.const_access(y*paddedWidth);
            auto nextRow = inputImage.const_access((y + 1)*paddedWidth);

            auto outputRow = outputImage.access(y*paddedWidth);

            #pragma omp simd
            for (int ix = StencilHaloSize; ix < endOfX; ++ix)
            {
                sdl::linear_index x(ix);

                const RGBs color1 = curRow[x - 1];
const RGBs color2 = curRow[x];
const RGBs color3 = curRow[x + 1];
const RGBs color4 = prevRow[x];
const RGBs color5 = nextRow[x];
// Despite looking like AOS code, compiler is able to create
// privatized instances and call inlinable methods on the objects
// keeping the algorithm at very high level
const RGBs sumOfColors = color1 + color2 + color3 + color4 + color5;
const RGBs averageColor = sumOfColors*(1.0f / 5.0f);
outputRow[x] = averageColor;
}
}
}
}
saveImageStub(outputImage.access());

Example 5
Example 5 converts a 2D image from the RGB format to the YUV format. It demonstrates how storing both images in 2D SoA n_containers can improve performance.

#include <iostream>
#include <sdlt/sdlt.h>
using namespace sdl;
#define WIDTH 1024
#define HEIGHT 1024

struct RGBs {
    float r;
    float g;
    float b;
};

struct YUVs {
    float y;
    float u;
    float v;
}

YUVs& operator=(const RGBs &tmp){
    y = 0.229f * tmp.r + 0.587f * tmp.g + 0.114f * tmp.b;
    u = -0.147f * tmp.r - 0.289f * tmp.g + 0.436f * tmp.b;
    v = 0.615 * tmp.r - 0.515f * tmp.g - 0.100 * tmp.b;
    return *this;
}

YUVs(const RGBs &tmp){
    y = 0.229f * tmp.r + 0.587f * tmp.g + 0.114f * tmp.b;
    u = -0.147f * tmp.r - 0.289f * tmp.g + 0.436f * tmp.b;
    v = 0.615 * tmp.r - 0.515f * tmp.g - 0.100 * tmp.b;
}

SDLT_PRIMITIVE(RGBs, r, g, b)
SDLT_PRIMITIVE(YUVs, y, u, v)

int main(){
    typedef layout::soa<> LayoutT;
    n_extent_t<int, int> extents(HEIGHT, WIDTH);
/* Creating a typedef for SoA N-dimensional container.
RGBTy and YUVTy are user defined structures whose collection needs to be stored in SoA
format in memory.
Layout in memory specified as layout::soa.
In the below case N-dimensional SoA container is used in 2-D context
*/
typedef sdlt::n_container< RGBs, LayoutT, decltype(extents) > ContainerRGB;
typedef sdlt::n_container< YUVs, LayoutT, decltype(extents) > ContainerYUV;

//Instantiate Input and Output Containers
ContainerRGB inputRGB(extents);
ContainerYUV outputYUV(extents);

auto input = inputRGB.const_access();  //Get Constant Accessor object for inputRGB
auto output = outputYUV.access();       //Get Accessor object for outputYUV

//Select the iteration range in each dimension
const auto iRGB1 = bounds_d<1>(input);  //bound_d<1>(input);
const auto iRGB0 = bounds_d<0>(input);  //bound_d<0>(input);

for(int y = iRGB0.lower(); y < iRGB0.upper(); y++)
{
    #pragma simd
    for (int x = iRGB1.lower(); x < iRGB1.upper(); x++)
    {
        const RGBs temp1 = input[y][x];
        YUVs temp2 = temp1;
        output[y][x] = temp2;
    }
}
return 0;

---

**Intel® C++ Class Libraries**

The Intel® C++ Class Libraries enable Single-Instruction, Multiple-Data (SIMD) operations. The principle of SIMD operations is to exploit microprocessor architecture through parallel processing. The effect of parallel processing is increased data throughput using fewer clock cycles. The objective is to improve application performance of complex and computation-intensive audio, video, and graphical data bit streams.

**Hardware and Software Requirements**

The Intel® C++ Class Libraries are functions abstracted from the instruction extensions available on Intel® processors.


**Details About the Libraries**

The Intel® C++ Class Libraries for SIMD Operations provide a convenient interface to access the underlying instructions for processors as specified above. These processor-instruction extensions enable parallel processing using the single instruction-multiple data (SIMD) technique as illustrated in the following figure.
SIMD Data Flow

Performing four operations with a single instruction improves efficiency by a factor of four for that particular instruction.

These new processor instructions can be implemented using assembly inlining, intrinsics, or the C++ SIMD classes. Compare the coding required to add four 32-bit floating-point values, using each of the available interfaces:

Comparison Between Inlining, Intrinsics and Class Libraries

<table>
<thead>
<tr>
<th>Assembly Inlining</th>
<th>Intrinsics</th>
<th>SIMD Class Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>... __m128 a,b,c;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__asm{ movaps xmm0,b</td>
<td>__m128 a,b,c; a =</td>
<td>F32vec4 a,b,c; a = b</td>
</tr>
<tr>
<td>movaps xmm1,c addps</td>
<td>__mm_add_ps(b,c); ...</td>
<td>+c; ...</td>
</tr>
<tr>
<td>xmm0,xmm1 movaps a,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xmm0 } ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This table shows an addition of four single-precision floating-point values using assembly inlining, intrinsics, and the libraries. You can see how much easier it is to code with the Intel C++ SIMD Class Libraries. Besides using fewer keystrokes and fewer lines of code, the notation is like the standard notation in C++, making it much easier to implement over other methods.

C++ Classes and SIMD Operations

Use of C++ classes for SIMD operations allows for operating on arrays or vectors of data in a single operation. Consider the addition of two vectors, A and B, where each vector contains four elements. Using an integer vector class, the elements A[i] and B[i] from each array are summed as shown in the following example.

Typical Method of Adding Elements Using a Loop

```
int a[4], b[4], c[4];
for (i=0; i<4; i++) /* needs four iterations */
c[i] = a[i] + b[i]; /* computes c[0], c[1], c[2], c[3] */
```

The following example shows the same results using one operation with an integer class.

SIMD Method of Adding Elements Using Ivec Classes

```
Is16vec4 ivecA, ivecB, ivec C; /*needs one iteration*/
ivecC = ivecA + ivecB; /*computes ivecC0, ivecC1, ivecC2, ivecC3 */
```

Available Classes

The Intel® C++ SIMD classes provide parallelism, which is not easily implemented using typical mechanisms of C++. The following table shows how the Intel® C++ classes use the SIMD classes and libraries.

SIMD Vector Classes
<table>
<thead>
<tr>
<th>Instruction Set</th>
<th>Class</th>
<th>Signedness</th>
<th>Data Type</th>
<th>Size</th>
<th>Elements</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMX™ technology</td>
<td>I64vec1</td>
<td>unspecified</td>
<td>__m64</td>
<td>64</td>
<td>1</td>
<td>ivec.h</td>
</tr>
<tr>
<td></td>
<td>I32vec2</td>
<td>unspecified</td>
<td>int</td>
<td>32</td>
<td>2</td>
<td>ivec.h</td>
</tr>
<tr>
<td></td>
<td>Is32vec2</td>
<td>signed</td>
<td>int</td>
<td>32</td>
<td>2</td>
<td>ivec.h</td>
</tr>
<tr>
<td></td>
<td>IU32vec2</td>
<td>unsigned</td>
<td>int</td>
<td>32</td>
<td>2</td>
<td>ivec.h</td>
</tr>
<tr>
<td></td>
<td>I16vec4</td>
<td>unspecified</td>
<td>short</td>
<td>16</td>
<td>4</td>
<td>ivec.h</td>
</tr>
<tr>
<td></td>
<td>Is16vec4</td>
<td>signed</td>
<td>short</td>
<td>16</td>
<td>4</td>
<td>ivec.h</td>
</tr>
<tr>
<td></td>
<td>IU16vec4</td>
<td>unsigned</td>
<td>short</td>
<td>16</td>
<td>4</td>
<td>ivec.h</td>
</tr>
<tr>
<td></td>
<td>I8vec8</td>
<td>unspecified</td>
<td>char</td>
<td>8</td>
<td>8</td>
<td>ivec.h</td>
</tr>
<tr>
<td></td>
<td>Is8vec8</td>
<td>signed</td>
<td>char</td>
<td>8</td>
<td>8</td>
<td>ivec.h</td>
</tr>
<tr>
<td></td>
<td>IU8vec8</td>
<td>unsigned</td>
<td>char</td>
<td>8</td>
<td>8</td>
<td>ivec.h</td>
</tr>
<tr>
<td>Intel® SSE</td>
<td>F32vec4</td>
<td>unspecified</td>
<td>float</td>
<td>32</td>
<td>4</td>
<td>fvec.h</td>
</tr>
<tr>
<td></td>
<td>F32vec1</td>
<td>unspecified</td>
<td>float</td>
<td>32</td>
<td>1</td>
<td>fvec.h</td>
</tr>
<tr>
<td>Intel® SSE2</td>
<td>F64vec2</td>
<td>unspecified</td>
<td>double</td>
<td>64</td>
<td>2</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>I128vec1</td>
<td>unspecified</td>
<td>__m128i</td>
<td>128</td>
<td>1</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>I64vec2</td>
<td>unspecified</td>
<td>long int</td>
<td>64</td>
<td>2</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>I32vec4</td>
<td>unspecified</td>
<td>int</td>
<td>32</td>
<td>4</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>Is32vec4</td>
<td>signed</td>
<td>int</td>
<td>32</td>
<td>4</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>IU32vec4</td>
<td>unsigned</td>
<td>int</td>
<td>32</td>
<td>4</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>I16vec8</td>
<td>unspecified</td>
<td>int</td>
<td>16</td>
<td>8</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>Is16vec8</td>
<td>signed</td>
<td>int</td>
<td>16</td>
<td>8</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>IU16vec8</td>
<td>unsigned</td>
<td>int</td>
<td>16</td>
<td>8</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>I8vec16</td>
<td>unspecified</td>
<td>char</td>
<td>8</td>
<td>16</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>Is8vec16</td>
<td>signed</td>
<td>char</td>
<td>8</td>
<td>16</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>IU8vec16</td>
<td>unsigned</td>
<td>char</td>
<td>8</td>
<td>16</td>
<td>dvec.h</td>
</tr>
<tr>
<td>Intel® AVX</td>
<td>F32vec8</td>
<td>unspecified</td>
<td>float</td>
<td>32</td>
<td>8</td>
<td>dvec.h</td>
</tr>
<tr>
<td>Intel® AVX-512</td>
<td>F64vec4</td>
<td>unspecified</td>
<td>double</td>
<td>64</td>
<td>4</td>
<td>dvec.h</td>
</tr>
<tr>
<td>Foundation</td>
<td>F32vec16</td>
<td>unspecified</td>
<td>float</td>
<td>32</td>
<td>16</td>
<td>dvec.h</td>
</tr>
<tr>
<td></td>
<td>F64vec8</td>
<td>unspecified</td>
<td>double</td>
<td>64</td>
<td>8</td>
<td>dvec.h</td>
</tr>
<tr>
<td>Instruction Set Extension</td>
<td>Include Directive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMX™ Technology</td>
<td>#include &lt;ivec.h&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Most classes contain similar functionality for all data types and are represented by all available intrinsics. However, some capabilities do not translate from one data type to another without suffering from poor performance, and are therefore excluded from individual classes.

**NOTE**
Intrinsics that take immediate values and cannot be expressed easily in classes are not implemented. For example:

- `__mm_shuffle_ps`
- `__mm_shuffle_pi16`
- `__mm_shuffle_ps`
- `__mm_extract_pi16`
- `__mm_insert_pi16`
### Instruction Set Extension

<table>
<thead>
<tr>
<th>Instruction Set Extension</th>
<th>Include Directive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® SSE</td>
<td>#include &lt;fvec.h&gt;</td>
</tr>
<tr>
<td>Intel® SSE 2</td>
<td>#include &lt;dvec.h&gt;</td>
</tr>
<tr>
<td>Intel® SSE 3</td>
<td>#include &lt;dvec.h&gt;</td>
</tr>
<tr>
<td>Intel® SSE 4</td>
<td>#include &lt;dvec.h&gt;</td>
</tr>
<tr>
<td>Intel® AVX</td>
<td>#include &lt;dvec.h&gt;</td>
</tr>
</tbody>
</table>

Each succeeding file from the top down includes the preceding class. You only need to include `fvec.h` if you want to use both the `Ivec` and `Fvec` classes. Similarly, to use all the classes including those for Intel® Streaming SIMD Extensions 2, you only need to include the `dvec.h` file.

### Usage Precautions

When using the C++ classes, you should follow some general guidelines. More detailed usage rules for each class are listed in Integer Vector Classes, and Floating-point Vector Classes.

### Clear MMX Registers

If you use both the `Ivec` and `Fvec` classes at the same time, your program could mix Intel® MMX™ instructions, called by `Ivec` classes, with Intel® architecture floating-point instructions, called by `Fvec` classes. x87 floating-point instructions exist in the following `Fvec` functions:

- fvec constructors
- debug functions (`cout` and element access)
- rsqrt_nr

#### NOTE

Intel® MMX™ technology registers are aliased on the floating-point registers, so you should clear the MMX state with the EMMS instruction intrinsic before issuing an x87 floating-point instruction, as in the following example.

```cpp
ivecA = ivecA & ivecB;  // Ivec logical operation that uses MMX instructions
empty ();               // clear state
cout << f32vec4a;       // F32vec4 operation that uses x87 floating-point instructions
```

#### Caution

Failure to clear the Intel® MMX™ technology registers can result in incorrect execution or poor performance due to an incorrect register state.

### Capabilities of C++ SIMD Classes

The fundamental capabilities of each C++ SIMD class include:

- computation
- horizontal data support
- branch compression/elimination
• caching hints
Understanding each of these capabilities and how they interact is crucial to achieving desired results.

Computation
The SIMD C++ classes contain vertical operator support for most arithmetic operations, including shifting and saturation.

Computation operations include: +, −, *, /, reciprocal (rcp and rcp_nr), square root (sqrt), and reciprocal square root (rsqrt and rsqrt_nr).

Operations rcp and rsqrt are approximating instructions with very short latencies that produce results with at least 12 bits of accuracy. You may get a different answer if used on non-Intel processors. Operations rcp_nr and rsqrt_nr use software refining techniques to enhance the accuracy of the approximations, with a minimal impact on performance. (The "nr" stands for Newton-Raphson, a mathematical technique for improving performance using an approximate result.)

Horizontal Data Support
The C++ SIMD classes provide horizontal support for some arithmetic operations. The term "horizontal" indicates computation across the elements of one vector, as opposed to the vertical, element-by-element operations on two different vectors.

The add_horizontal, unpack_low and pack_sat functions are examples of horizontal data support. This support enables certain algorithms that cannot exploit the full potential of SIMD instructions.

Shuffle intrinsics are another example of horizontal data flow. Shuffle intrinsics are not expressed in the C++ classes due to their immediate arguments. However, the C++ class implementation enables you to mix shuffle intrinsics with the other C++ functions. For example:

```cpp
F32vec4 fveca, fvecb, fvecd;
fveca += fvecb;
fvectd = _mm_shuffle_ps(fveca,fvecb,0);
```

Branch Compression/Elimination
Branching in SIMD architectures can be complicated and expensive. The SIMD C++ classes provide functions to eliminate branches, using logical operations, max and min functions, conditional selects, and compares.
Consider the following example:

```cpp
short a[4], b[4], c[4];
for (i=0; i<4; i++)
c[i] = a[i] > b[i] ? a[i] : b[i];
```

This operation is independent of the value of i. For each i, the result could be either A or B depending on the actual values. A simple way of removing the branch altogether is to use the select_gt function, as follows:

```cpp
Is16vec4 a, b, c
C = select_gt(a, b, c)
```

Caching Hints
Intel® Streaming SIMD Extensions provide prefetching and streaming hints. Prefetching data can minimize the effects of memory latency. Streaming hints allow you to indicate that certain data should not be cached.

Integer Vector Classes
The Ivec classes provide an interface to single instruction, multiple data (SIMD) processing using integer vectors of various sizes. The class hierarchy is represented in the following figure.
Ivec Class Hierarchy

The M64 and M128 classes define the __m64 and __m128i data types from which the rest of the Ivec classes are derived. The first generation of child classes (the intermediate classes) are derived on element sizes of 128, 64, 32, 16, and 8 bits:

I128vec1, I64vec1, I64vec2, I32vec2, I32vec4, I16vec4, I16vec8, I8vec8, I8vec16

The second generation specify the signedness:

Is64vec2, Iu64vec2, Is32vec2, Iu32vec2, Is32vec4, Iu32vec4, Is16vec4, Iu16vec4, Is16vec8, Iu16vec8, Is8vec8, Iu8vec8, Is8vec16, Iu8vec16

Caution

Intermixing the M64 and M128 data types will result in unexpected behavior.

Terms, Conventions, and Syntax Defined

The following are special terms and syntax used in this chapter to describe functionality of the classes with respect to their associated operations.

Ivec Class Syntax Conventions

The name of each class denotes the data type, signedness, bit size, and number of elements using the following generic format:

```plaintext
<type><signedness><bits>vec<elements>
```

where

- `type` Indicates floating point (F) or integer (I).
- `signedness` Indicates signed (s) or unsigned (u). For the Ivec class, leaving this field blank indicates an intermediate class. For the Fvec classes, this field is blank because there are no unsigned Fvec classes.
- `bits` Specifies the number of bits per element.
- `elements` Specifies the number of elements.
Special Terms and Conventions

The following terms are used to define the functionality and characteristics of the classes and operations defined in this manual.

- **Nearest Common Ancestor**: This is the intermediate or parent class of two classes of the same size. For example, the nearest common ancestor of `Iu8vec8` and `Is8vec8` is `I8vec8`, and the nearest common ancestor between `Iu8vec8` and `I16vec4` is `M64`.

- **Casting**: Changes the data type from one class to another. When an operation uses different data types as operands, the return value of the operation must be assigned to a single data type, and one or more of the data types must be converted to a required data type. This conversion is known as a typecast. While typecasting is occasionally automatic, in cases where it is not automatic you must use special syntax to explicitly typecast it yourself.

- **Operator Overloading**: This is the ability to use various operators on the user-defined data type of a given class. In the case of the Ivec and Fvec classes, once you declare a variable, you can add, subtract, multiply, and perform a range of operations. Each family of classes accepts a specified range of operators, and must comply by rules and restrictions regarding typecasting and operator overloading as defined in the header files.

Rules for Operators

To use operators with the Ivec classes you must use one of the following three syntax conventions:

```
[ Ivec_Class ] R = [ Ivec_Class ] A [ operator ] [ Ivec_Class ] B
```

**Example 1:**

```
I64vec1 R = I64vec1 A & I64vec1 B;
```

```
```

**Example 2:**

```
I64vec1 R = andnot(I64vec1 A, I64vec1 B);
```

```
[ Ivec_Class ] R [ operator ]= [ Ivec_Class ] A
```

**Example 3:**

```
I64vec1 R &= I64vec1 A;
```

- `[ operator ]` represents an operator (for example, `&`, `|`, or `^`)
- `[ Ivec_Class ]` represents an Ivec class
- `R, A, B` variables are declared using the pertinent Ivec classes

The table that follows shows automatic and explicit sign and size typecasting. "Explicit" means that it is illegal to mix different types without an explicit typecasting. "Automatic" means that you can mix types freely and the compiler will do the typecasting for you.

### Summary of Rules Major Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Sign Typecasting</th>
<th>Size Typecasting</th>
<th>Other Typecasting Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Logical</td>
<td>Automatic</td>
<td>Automatic (to left)</td>
<td>Explicit typecasting is required for different types used in non-logical expressions on the right side of the assignment.</td>
</tr>
<tr>
<td>Addition and Subtraction</td>
<td>Automatic</td>
<td>Explicit</td>
<td>N/A</td>
</tr>
<tr>
<td>Multiplication</td>
<td>Automatic</td>
<td>Explicit</td>
<td>N/A</td>
</tr>
<tr>
<td>Shift</td>
<td>Automatic</td>
<td>Explicit</td>
<td>Casting Required to ensure arithmetic shift.</td>
</tr>
</tbody>
</table>
### Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Sign Typecasting</th>
<th>Size Typecasting</th>
<th>Other Typecasting Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare</td>
<td>Automatic</td>
<td>Explicit</td>
<td>Explicit casting is required for signed classes for the less-than or greater-than operations.</td>
</tr>
<tr>
<td>Conditional Select</td>
<td>Automatic</td>
<td>Explicit</td>
<td>Explicit casting is required for signed classes for less-than or greater-than operations.</td>
</tr>
</tbody>
</table>

### Data Declaration and Initialization

The following table shows literal examples of constructor declarations and data type initialization for all class sizes. All values are initialized with the most significant element on the left and the least significant to the right.

#### Declaration and Initialization Data Types for Ivec Classes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Class</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>M128</td>
<td>i128vec1 A; iu8vec16 A;</td>
</tr>
<tr>
<td>Declaration</td>
<td>M64</td>
<td>i64vec1 A; iu8vec8 A;</td>
</tr>
<tr>
<td>__m128 Initialization</td>
<td>M128</td>
<td>i128vec1 A(__m128 m); iu16vec8(__m128 m);</td>
</tr>
<tr>
<td>__m64 Initialization</td>
<td>M64</td>
<td>i64vec1 A(__m64 m); iu8vec8 A(__m64 m);</td>
</tr>
<tr>
<td>__int64 Initialization</td>
<td>M64</td>
<td>i64vec1 A = __int64 m; iu8vec8 A = __int64 m;</td>
</tr>
<tr>
<td>int i Initialization</td>
<td>M64</td>
<td>i64vec1 A = int i; iu8vec8 A = int i;</td>
</tr>
<tr>
<td>int Initialization</td>
<td>I32vec2</td>
<td>i32vec2 A(int A1, int A0); i32vec2 A(signed int A1, signed int A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>iu32vec2 A(unsigned int A1, unsigned int A0);</td>
</tr>
<tr>
<td>int Initialization</td>
<td>I32vec4</td>
<td>i32vec4 A(int A3, int A2, int A1, int A0); i32vec4 A(signed int A3, signed int A2, signed int A1, signed int A0);</td>
</tr>
<tr>
<td>short int Initialization</td>
<td>I16vec4</td>
<td>i16vec4 A(short A3, short A2, short A1, short A0); i16vec4 A(signed short A3, short A2, signed short A1, signed short A0);</td>
</tr>
<tr>
<td>Operation</td>
<td>Class</td>
<td>Syntax</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>short int Initialization</td>
<td>I16vec8</td>
<td>Iu16vec4 A(unsigned short A3, ..., unsigned short A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I16vec8 A(short A7, short A6, ..., short A1, short A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is16vec8 A(signed A7, ..., signed short A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iu16vec8 A(unsigned short A7, ..., unsigned short A0);</td>
</tr>
<tr>
<td>char Initialization</td>
<td>I8vec8</td>
<td>I8vec8 A(char A7, char A6, ..., char A1, char A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is8vec8 A(signed char A7, ..., signed char A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iu8vec8 A(unsigned char A7, ..., unsigned char A0);</td>
</tr>
<tr>
<td>char Initialization</td>
<td>I8vec16</td>
<td>I8vec16 A(char A15, ..., char A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is8vec16 A(signed char A15, ..., signed char A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iu8vec16 A(unsigned char A15, ..., unsigned char A0);</td>
</tr>
</tbody>
</table>

**Assignment Operator**

Any Ivec object can be assigned to any other Ivec object; conversion on assignment from one Ivec object to another is automatic.

**Assignment Operator Examples**

```c
Is16vec4 A;
Is8vec8 B;
I64vec1 C;
A = B; /* assign Is8vec8 to Is16vec4 */
B = C; /* assign I64vec1 to Is8vec8 */
B = A & C; /* assign M64 result of '&' to Is8vec8 */
```

**Logical Operators**

The logical operators use the symbols and intrinsics listed in the following table.
### Bitwise Operation

<table>
<thead>
<tr>
<th>Bitwise Operation</th>
<th>Operator Symbols</th>
<th>Syntax Usage</th>
<th>Corresponding Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard</td>
<td>w/assign</td>
<td>Standard</td>
</tr>
<tr>
<td>AND</td>
<td>&amp;</td>
<td>&amp;=</td>
<td>R = A &amp; B</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
<td>R = A</td>
</tr>
<tr>
<td>XOR</td>
<td>^</td>
<td>^=</td>
<td>R = A^B</td>
</tr>
<tr>
<td>ANDNOT</td>
<td>andnot</td>
<td>N/A</td>
<td>R = A</td>
</tr>
</tbody>
</table>

### Logical Operators and Miscellaneous Exceptions

A and B converted to M64. Result assigned to \texttt{Iu8vec8}.

\begin{verbatim}
I64vec1 A;
Is8vec8 B;
Iu8vec8 C;
C = A & B;
\end{verbatim}

Same size and signedness operators return the nearest common ancestor.

\begin{verbatim}
I32vec2 R = Is32vec2 A ^ Iu32vec2 B;
A&B returns M64, which is cast to \texttt{Iu8vec8}.
C = Iu8vec8(A&B) + C;
\end{verbatim}

When A and B are of the same class, they return the same type. When A and B are of different classes, the return value is the return type of the nearest common ancestor.

The logical operator returns values for combinations of classes, listed in the following tables, apply when A and B are of different classes.

### Ivec Logical Operator Overloading

<table>
<thead>
<tr>
<th>Return (R)</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>NAND</th>
<th>A Operand</th>
<th>B Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>I64vec1 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>andnot</td>
<td>I[s]</td>
<td>I[s]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>u]64vec2 A</td>
<td>u]64vec2 B</td>
</tr>
<tr>
<td>I64vec2 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>andnot</td>
<td>I[s]</td>
<td>I[s]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>u]64vec2 A</td>
<td>u]64vec2 B</td>
</tr>
<tr>
<td>I32vec2 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>andnot</td>
<td>I[s]</td>
<td>I[s]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>u]32vec2 A</td>
<td>u]32vec2 B</td>
</tr>
<tr>
<td>I32vec4 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>andnot</td>
<td>I[s]</td>
<td>I[s]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>u]32vec4 A</td>
<td>u]32vec4 B</td>
</tr>
<tr>
<td>I16vec4 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>andnot</td>
<td>I[s]</td>
<td>I[s]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>u]16vec4 A</td>
<td>u]16vec4 B</td>
</tr>
<tr>
<td>I16vec8 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>andnot</td>
<td>I[s]</td>
<td>I[s]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>u]16vec8 A</td>
<td>u]16vec8 B</td>
</tr>
<tr>
<td>I8vec8 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>andnot</td>
<td>I[s</td>
<td>u]8vec8</td>
</tr>
</tbody>
</table>
For logical operators with assignment, the return value of \( R \) is always the same data type as the pre-declared value of \( R \) as listed in the table that follows.

**Ivec Logical Operator Overloading with Assignment**

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Left Side (R)</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>Right Side (Any Ivec Type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I128vec1</td>
<td>I128vec1 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>I[s</td>
</tr>
<tr>
<td>I64vec1</td>
<td>I64vec1 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>I[s</td>
</tr>
<tr>
<td>I64vec2</td>
<td>I64vec2 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]32vec2</td>
<td>I[x]32vec2 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]8vec16</td>
<td>I[x]8vec16 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]8vec8</td>
<td>I[x]8vec8 R</td>
<td>&amp;</td>
<td></td>
<td>^</td>
<td>I[s</td>
</tr>
</tbody>
</table>

**Addition and Subtraction Operators**

The addition and subtraction operators return the class of the nearest common ancestor when the right-side operands are of different signs. The following code provides examples of usage and miscellaneous exceptions.

**Syntax Usage for Addition and Subtraction Operators**

Return nearest common ancestor type, I16vec4.

\[
\text{Is16vec4 A;}
\text{Iu16vec4 B;}
\text{I16vec4 C;}
\text{C = A + B;}
\]

Returns type left-hand operand type.

\[
\text{Is16vec4 A;}
\text{Iu16vec4 B;}
\text{A += B;}
\text{B -= A;}
\]

Explicitly convert \( B \) to Is16vec4.

\[
\text{Is16vec4 A,C;}
\]
Iu32vec24 B;
C = A + C;
C = A + (Is16vec4)B;

Addition and Subtraction Operators with Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbols</th>
<th>Syntax</th>
<th>Corresponding Intrinsics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>R = A + B</td>
<td>_mm_add_epi64</td>
</tr>
<tr>
<td></td>
<td>+=</td>
<td>R += A</td>
<td>_mm_add_epi32 _mm_add_epi16 _mm_add_epi8</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>R = A - B</td>
<td>_mm_sub_epi64</td>
</tr>
<tr>
<td></td>
<td>-=</td>
<td>R -= A</td>
<td>_mm_sub_epi32 _mm_sub_epi16 _mm_sub_epi8</td>
</tr>
</tbody>
</table>

The following table lists addition and subtraction return values for combinations of classes when the right side operands are of different signedness. The two operands must be the same size, otherwise you must explicitly indicate the typecasting.

Addition and Subtraction Operator Overloading

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Available Operators</th>
<th>Right Side Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>R I64vec2 R</td>
<td>Add</td>
<td>Sub</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following table shows the return data type values for operands of the addition and subtraction operators with assignment. The left side operand determines the size and signedness of the return value. The right side operand must be the same size as the left operand; otherwise, you must use an explicit typecast.

Addition and Subtraction with Assignment

<table>
<thead>
<tr>
<th>Return Value (R)</th>
<th>Left Side (R)</th>
<th>Add</th>
<th>Sub</th>
<th>Right Side (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I[x]32vec4</td>
<td>I[x]32vec2 R</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]32vec2 R</td>
<td>I[x]32vec2 R</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
</tbody>
</table>
### Return Value (R) | Left Side (R) | Add | Sub | Right Side (A)
---|---|---|---|---
I[x]16vec8 | I[x]16vec8 | += | -= | I[s|u]16vec8 A;
I[x]16vec4 | I[x]16vec4 | += | -= | I[s|u]16vec4 A;
I[x]8vec16 | I[x]8vec16 | += | -= | I[s|u]8vec16 A;
I[x]8vec8 | I[x]8vec8 | += | -= | I[s|u]8vec8 A;

### Multiplication Operators

The multiplication operators can only accept and return data types from the I[s|u]16vec4 or I[s|u]16vec8 classes, as shown in the following example.

#### Syntax Usage for Multiplication Operators

Explicitly convert B to Is16vec4.

```c
Is16vec4 A, C;
Iu32vec2 B;
C = A * C;
C = A * (Is16vec4)B;
```

Return nearest common ancestor type, I16vec4

```c
Is16vec4 A;
Iu16vec4 B;
I16vec4 C;
C = A + B;
```

The `mul_high` and `mul_add` functions take Is16vec4 data only.

```c
Is16vec4 A, B, C, D;
C = mul_high(A, B);
D = mul_add(A, B);
```

### Multiplication Operators with Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*=</td>
<td>_mm_mullo_pi16</td>
</tr>
<tr>
<td>*</td>
<td>*=</td>
<td>_mm_mullo_epi16</td>
</tr>
<tr>
<td>mul_high</td>
<td>N/A</td>
<td>_mm_mulhi_pi16</td>
</tr>
<tr>
<td>mul_high</td>
<td>N/A</td>
<td>_mm_mulhi_epi16</td>
</tr>
<tr>
<td>mul_add</td>
<td>N/A</td>
<td>_mm_madd_pi16</td>
</tr>
<tr>
<td>mul_add</td>
<td>N/A</td>
<td>_mm_madd_epi16</td>
</tr>
</tbody>
</table>

The multiplication return operators always return the nearest common ancestor as listed in the table that follows. The two operands must be 16 bits in size, otherwise you must explicitly indicate typecasting.

### Multiplication Operator Overloading

<table>
<thead>
<tr>
<th>R</th>
<th>Mul</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>I16vec4 R</td>
<td>*</td>
<td>I[s</td>
<td>u]16vec4 A</td>
</tr>
</tbody>
</table>
The following table shows the return values and data type assignments for operands of the multiplication operators with assignment. All operands must be 16 bytes in size. If the operands are not the right size, you must use an explicit typecast.

### Multiplication with Assignment

<table>
<thead>
<tr>
<th>Return Value (R)</th>
<th>Left Side (R)</th>
<th>Mul</th>
<th>Right Side (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I[x]16vec8</td>
<td>I[x]16vec8</td>
<td>*</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]16vec4</td>
<td>I[x]16vec4</td>
<td>*=</td>
<td>I[s</td>
</tr>
</tbody>
</table>

### Shift Operators

The right shift argument can be any integer or Ivec value, and is implicitly converted to a M64 data type. The first or left operand of a `<<` can be of any type except `I[s|u]8vec[8|16]`.

#### Example Syntax Usage for Shift Operators

Automatic size and sign conversion.

```c
Is16vec4 A,C;
Iu32vec2 B;
C = A;
A&B returns I16vec4, which must be cast to Iu16vec4 to ensure logical shift, not arithmetic shift.
Is16vec4 A, C;
Iu16vec4 B, R;
R = (Iu16vec4)(A & B) C;
A&B returns I16vec4, which must be cast to Is16vec4 to ensure arithmetic shift, not logical shift.
R = (Is16vec4)(A & B) C;
```

### Shift Operators with Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbols</th>
<th>Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift Left</td>
<td><code>&lt;&lt;</code></td>
<td>R = A &lt;&lt; B</td>
<td><code>_mm_sll_si64</code></td>
</tr>
<tr>
<td></td>
<td><code>&amp;=</code></td>
<td>R &amp;= A</td>
<td><code>_mm_slli_si64</code></td>
</tr>
<tr>
<td>Shift Right</td>
<td><code>&gt;&gt;</code></td>
<td>R = A &gt;&gt; B</td>
<td><code>_mm_srl_si64</code></td>
</tr>
<tr>
<td></td>
<td><code>&gt;&gt;=</code></td>
<td>R &gt;&gt;= A</td>
<td><code>_mm_srli_si64</code></td>
</tr>
</tbody>
</table>
Right shift operations with signed data types use arithmetic shifts. All unsigned and intermediate classes correspond to logical shifts. The following table shows how the return type is determined by the first argument type.

### Shift Operator Overloading

<table>
<thead>
<tr>
<th>Option</th>
<th>R</th>
<th>Right Shift</th>
<th>Left Shift</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>I64vec1</td>
<td>&gt;&gt;</td>
<td>&gt;=</td>
<td>I64vec1 A</td>
<td>I64vec1 B</td>
</tr>
<tr>
<td>Logical</td>
<td>I32vec2</td>
<td>&gt;&gt;</td>
<td>&gt;=</td>
<td>I32vec2 A</td>
<td>I32vec2 B</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Is32vec2</td>
<td>&gt;&gt;</td>
<td>&gt;=</td>
<td>Is32vec2 A</td>
<td>I[s</td>
</tr>
<tr>
<td>Logical</td>
<td>Iu32vec2</td>
<td>&gt;&gt;</td>
<td>&gt;=</td>
<td>Iu32vec2 A</td>
<td>I[s</td>
</tr>
<tr>
<td>Logical</td>
<td>I16vec4</td>
<td>&gt;&gt;</td>
<td>&gt;=</td>
<td>I16vec4 A</td>
<td>I16vec4 B</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Is16vec4</td>
<td>&gt;&gt;</td>
<td>&gt;=</td>
<td>Is16vec4 A</td>
<td>I[s</td>
</tr>
<tr>
<td>Logical</td>
<td>Iu16vec4</td>
<td>&gt;&gt;</td>
<td>&gt;=</td>
<td>Iu16vec4 A</td>
<td>I[s</td>
</tr>
</tbody>
</table>

### Comparison Operators

The equality and inequality comparison operands can have mixed signedness, but they must be of the same size. The comparison operators for less-than and greater-than must be of the same sign and size.

#### Example of Syntax Usage for Comparison Operator

The nearest common ancestor is returned for compare for equal/not-equal operations.

```cpp
Iu8vec8 A;
Is8vec8 B;
I8vec8 C;
C = cmpneq(A,B);
```

Type cast needed for different-sized elements for equal/not-equal comparisons.

```cpp
Iu8vec8 A, C;
Is16vec4 B;
```
C = cmpeq(A, (Iu8vec8)B);

Type cast needed for sign or size differences for less-than and greater-than comparisons.

Iu16vec4 A;
Is16vec4 B, C;
C = cmpge((Is16vec4)A, B);
C = cmpgt(B, C);

**Inequality Comparison Symbols and Corresponding Intrinsics**

<table>
<thead>
<tr>
<th>Compare For:</th>
<th>Operators</th>
<th>Syntax</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>cmpeq</td>
<td>R = cmpeq(A, B)</td>
<td>_mm_cmpeq_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_cmpeq_pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_cmpeq_pi8</td>
</tr>
<tr>
<td>Inequality</td>
<td>cmpneq</td>
<td>R = cmpneq(A, B)</td>
<td>_mm_cmpeq_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_cmpeq_pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_cmpeq_pi8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_andnot_si64</td>
</tr>
<tr>
<td>Greater Than</td>
<td>cmpgt</td>
<td>R = cmpgt(A, B)</td>
<td>_mm_cmpgt_pi32</td>
</tr>
<tr>
<td>Greater Than</td>
<td></td>
<td></td>
<td>_mm_cmpgt_pi16</td>
</tr>
<tr>
<td>or Equal To</td>
<td>cmpge</td>
<td>R = cmpge(A, B)</td>
<td>_mm_cmpgt_pi32</td>
</tr>
<tr>
<td>Less Than</td>
<td>cmplt</td>
<td>R = cmplt(A, B)</td>
<td>_mm_cmpgt_pi32</td>
</tr>
<tr>
<td>Less Than</td>
<td></td>
<td></td>
<td>_mm_cmpgt_pi16</td>
</tr>
<tr>
<td>or Equal To</td>
<td>cmple</td>
<td>R = cmple(A, B)</td>
<td>_mm_cmpgt_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_andnot_si64</td>
</tr>
</tbody>
</table>

Comparison operators have the restriction that the operands must be the size and sign as listed in the Compare Operator Overloading table.

**Compare Operator Overloading**

<table>
<thead>
<tr>
<th>R</th>
<th>Comparison</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>I32vec2  R</td>
<td>cmpeq</td>
<td>I[s</td>
<td>u]32vec2 B</td>
</tr>
<tr>
<td>I32vec2  R</td>
<td>cmpne</td>
<td>I[s</td>
<td>u]32vec2 B</td>
</tr>
<tr>
<td>I16vec4  R</td>
<td></td>
<td>I[s</td>
<td>u]16vec4 B</td>
</tr>
<tr>
<td>I8vec8   R</td>
<td></td>
<td>I[s</td>
<td>u]8vec8 B</td>
</tr>
<tr>
<td>I32vec2  R</td>
<td>cmpgt</td>
<td>Is32vec2 B</td>
<td>Is32vec2 B</td>
</tr>
<tr>
<td>I32vec2  R</td>
<td>cmpge</td>
<td>Is32vec2 B</td>
<td>Is32vec2 B</td>
</tr>
<tr>
<td>I32vec2  R</td>
<td>cmplt</td>
<td>Is32vec2 B</td>
<td>Is32vec2 B</td>
</tr>
<tr>
<td>I32vec2  R</td>
<td>cmple</td>
<td>Is32vec2 B</td>
<td>Is32vec2 B</td>
</tr>
</tbody>
</table>
### Conditional Select Operators

For conditional select operands, the third and fourth operands determine the type returned. Third and fourth operands with same size, but different signedness, return the nearest common ancestor data type.

#### Conditional Select Syntax Usage

Return the nearest common ancestor data type if third and fourth operands are of the same size, but different signs.

```cpp
I16vec4 R = select_neq(Is16vec4, Is16vec4, Is16vec4, Iu16vec4);
```

#### Conditional Select for Equality

R0 := (A0 == B0) ? C0 : D0;
R1 := (A1 == B1) ? C1 : D1;
R2 := (A2 == B2) ? C2 : D2;
R3 := (A3 == B3) ? C3 : D3;

#### Conditional Select for Inequality

R0 := (A0 != B0) ? C0 : D0;
R1 := (A1 != B1) ? C1 : D1;
R2 := (A2 != B2) ? C2 : D2;
R3 := (A3 != B3) ? C3 : D3;

#### Conditional Select Symbols and Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Conditional Select For:</th>
<th>Operators</th>
<th>Syntax</th>
<th>Corresponding Intrinsic</th>
<th>Additional Intrinsic (Applies to All)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>select_eq</td>
<td>R = select_eq(A, B, C, D)</td>
<td>_mm_cmpeq_pi32</td>
<td>_mm_and_si64 _mm_or_si64 _mm_andnot_si64</td>
</tr>
<tr>
<td>Inequality</td>
<td>select_neq</td>
<td>R = select_neq(A, B, C, D)</td>
<td>_mm_cmpeq_pi32</td>
<td>_mm_and_si64 _mm_or_si64 _mm_andnot_si64</td>
</tr>
<tr>
<td>Greater Than</td>
<td>select_gt</td>
<td>R = select_gt(A, B, C, D)</td>
<td>_mm_cmpgt_pi32</td>
<td>_mm_and_si64 _mm_or_si64 _mm_andnot_si64</td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>select_ge</td>
<td>R = select_ge(A, B, C, D)</td>
<td>_mm_cmpge_pi32</td>
<td>_mm_and_si64 _mm_or_si64 _mm_andnot_si64</td>
</tr>
<tr>
<td>Less Than</td>
<td>select_lt</td>
<td>R = select_lt(A, B, C, D)</td>
<td>_mm_cmplt_pi32</td>
<td>_mm_and_si64 _mm_or_si64 _mm_andnot_si64</td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>select_le</td>
<td>R = select_le(A, B, C, D)</td>
<td>_mm_cmple_pi32</td>
<td>_mm_and_si64 _mm_or_si64 _mm_andnot_si64</td>
</tr>
</tbody>
</table>
All conditional select operands must be of the same size. The return data type is the nearest common ancestor of operands C and D. For conditional select operations using greater-than or less-than operations, the first and second operands must be signed as listed in the table that follows.

### Conditional Select Operator Overloading

<table>
<thead>
<tr>
<th>R</th>
<th>Comparison</th>
<th>A and B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>I32vec2 R</td>
<td>select_eq</td>
<td>I[s</td>
<td>u]32vec2</td>
<td>I[s</td>
</tr>
<tr>
<td>I32vec2 R</td>
<td>select_ne</td>
<td>I[s</td>
<td>u]32vec2</td>
<td>I[s</td>
</tr>
<tr>
<td>I64vec4 R</td>
<td>select_gt</td>
<td>I[s</td>
<td>u]8vec8</td>
<td>I[s</td>
</tr>
<tr>
<td>I64vec4 R</td>
<td>select_ge</td>
<td>I[s</td>
<td>u]16vec4</td>
<td>I[s</td>
</tr>
<tr>
<td>I64vec4 R</td>
<td>select_lt</td>
<td>I[s</td>
<td>u]16vec4</td>
<td>I[s</td>
</tr>
<tr>
<td>I64vec4 R</td>
<td>select_le</td>
<td>I[s</td>
<td>u]16vec4</td>
<td>I[s</td>
</tr>
</tbody>
</table>

The following table shows the mapping of return values from R0 to R7 for any number of elements. The same return value mappings also apply when there are fewer than four return values.

### Conditional Select Operator Return Value Mapping

<table>
<thead>
<tr>
<th>Return Value</th>
<th>A Operands</th>
<th>Available Operators</th>
<th>B Operands</th>
<th>C and D Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:= A0</td>
<td>== != &gt;= &lt;</td>
<td>B0</td>
<td>C0 : D0</td>
<td></td>
</tr>
<tr>
<td>R1:= A0</td>
<td>== != &gt;= &lt;</td>
<td>B0</td>
<td>C1 : D1</td>
<td></td>
</tr>
<tr>
<td>R2:= A0</td>
<td>== != &gt;= &lt;</td>
<td>B0</td>
<td>C2 : D2</td>
<td></td>
</tr>
<tr>
<td>R3:= A0</td>
<td>== != &gt;= &lt;</td>
<td>B0</td>
<td>C3 : D3</td>
<td></td>
</tr>
<tr>
<td>R4:= A0</td>
<td>== != &gt;= &lt;</td>
<td>B0</td>
<td>C4 : D4</td>
<td></td>
</tr>
<tr>
<td>R5:= A0</td>
<td>== != &gt;= &lt;</td>
<td>B0</td>
<td>C5 : D5</td>
<td></td>
</tr>
<tr>
<td>R6:= A0</td>
<td>== != &gt;= &lt;</td>
<td>B0</td>
<td>C6 : D6</td>
<td></td>
</tr>
<tr>
<td>R7:= A0</td>
<td>== != &gt;= &lt;</td>
<td>B0</td>
<td>C7 : D7</td>
<td></td>
</tr>
</tbody>
</table>

### Debug Operations

The debug operations do not map to any compiler intrinsics for MMX™ instructions. They are provided for debugging programs only. Use of these operations may result in loss of performance, so you should not use them outside of debugging.

### Output

The four 32-bit values of A are placed in the output buffer and printed in the following format (default in decimal):
cout << Is32vec4 A;
cout << Iu32vec4 A;
cout << hex << Iu32vec4 A; /* print in hex format */
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"

Corresponding Intrinsics: none

The two 32-bit values of A are placed in the output buffer and printed in the following format (default in decimal):
cout << Is32vec2 A;
cout << Iu32vec2 A;
cout << hex << Iu32vec2 A; /* print in hex format */
"[1]:A1 [0]:A0"

Corresponding Intrinsics: none

The eight 16-bit values of A are placed in the output buffer and printed in the following format (default in decimal):
cout << Is16vec8 A;
cout << Iu16vec8 A;
cout << hex << Iu16vec8 A; /* print in hex format */
"[7]:A7 [6]:A6 [5]:A5 [4]:A4 [3]:A3 [2]:A2 [1]:A1 [0]:A0"

Corresponding Intrinsics: none

The four 16-bit values of A are placed in the output buffer and printed in the following format (default in decimal):
cout << Is16vec4 A;
cout << Iu16vec4 A;
cout << hex << Iu16vec4 A; /* print in hex format */
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"

Corresponding Intrinsics: none

The sixteen 8-bit values of A are placed in the output buffer and printed in the following format (default is decimal):
cout << Is8vec16 A; cout << Iu8vec16 A; cout << hex << Iu8vec8 A;
/* print in hex format instead of decimal*/

Corresponding Intrinsics: none

The eight 8-bit values of A are placed in the output buffer and printed in the following format (default is decimal):
cout << Is8vec8 A; cout << Iu8vec8 A; cout << hex << Iu8vec8 A;
/* print in hex format instead of decimal*/
"[7]:A7 [6]:A6 [5]:A5 [4]:A4 [3]:A3 [2]:A2 [1]:A1 [0]:A0"
Corresponding Intrinsics: none

**Element Access Operators**

```c
int R = Is64vec2 A[i];
unsigned int R = Iu64vec2 A[i];
int R = Is32vec4 A[i];
unsigned int R = Iu32vec4 A[i];
int R = Is32vec2 A[i];
unsigned int R = Iu32vec2 A[i];
short R = Is16vec8 A[i];
unsigned short R = Iu16vec8 A[i];
short R = Is16vec4 A[i];
unsigned short R = Iu16vec4 A[i];
signed char R = Is8vec16 A[i];
unsigned char R = Iu8vec16 A[i];
signed char R = Is8vec8 A[i];
unsigned char R = Iu8vec8 A[i];
```

Access and read element \( i \) of \( A \). If DEBUG is enabled and the user tries to access an element outside of \( A \), a diagnostic message is printed and the program aborts.

Corresponding Intrinsics: none

**Element Assignment Operators**

```c
Is64vec2 A[i] = int R;
Is32vec4 A[i] = int R;
Iu32vec4 A[i] = unsigned int R;
Is32vec2 A[i] = int R;
Iu32vec2 A[i] = unsigned int R;
Is16vec8 A[i] = short R;
Iu16vec8 A[i] = unsigned short R;
Is16vec4 A[i] = short R;
Iu16vec4 A[i] = unsigned short R;
Is8vec16 A[i] = signed char R;
Iu8vec16 A[i] = unsigned char R;
Is8vec8 A[i] = signed char R;
Iu8vec8 A[i] = unsigned char R;
```

Assign \( R \) to element \( i \) of \( A \). If DEBUG is enabled and the user tries to assign a value to an element outside of \( A \), a diagnostic message is printed and the program aborts.

Corresponding Intrinsics: none
Unpack Operators

Interleave the 64-bit value from the high half of A with the 64-bit value from the high half of B.

I64vec2 unpack_high(I64vec2 A, I64vec2 B);
I64vec2 unpack_high(Is64vec2 A, Is64vec2 B);
I64vec2 unpack_high(Iu64vec2 A, Iu64vec2 B);
R0 = A1;
R1 = B1;

Corresponding intrinsic: _mm_unpackhi_epi64

Interleave the two 32-bit values from the high half of A with the two 32-bit values from the high half of B.

I32vec4 unpack_high(I32vec4 A, I32vec4 B);
I32vec4 unpack_high(Is32vec4 A, Is32vec4 B);
I32vec4 unpack_high(Iu32vec4 A, Iu32vec4 B);
R0 = A1;
R1 = B1;
R2 = A2;
R3 = B2;

Corresponding intrinsic: _mm_unpackhi_epi32

Interleave the 32-bit value from the high half of A with the 32-bit value from the high half of B.

I32vec2 unpack_high(I32vec2 A, I32vec2 B);
I32vec2 unpack_high(Is32vec2 A, Is32vec2 B);
I32vec2 unpack_high(Iu32vec2 A, Iu32vec2 B);
R0 = A1;
R1 = B1;

Corresponding intrinsic: _mm_unpackhi_pi32

Interleave the four 16-bit values from the high half of A with the two 16-bit values from the high half of B.

I16vec8 unpack_high(I16vec8 A, I16vec8 B);
I16vec8 unpack_high(Is16vec8 A, Is16vec8 B);
I16vec8 unpack_high(Iu16vec8 A, Iu16vec8 B);
R0 = A2;
R1 = B2;
R2 = A3;
R3 = B3;

Corresponding intrinsic: _mm_unpackhi_epi16

Interleave the two 16-bit values from the high half of A with the two 16-bit values from the high half of B.

I16vec4 unpack_high(I16vec4 A, I16vec4 B);
I16vec4 unpack_high(Is16vec4 A, Is16vec4 B);
I16vec4 unpack_high(Iu16vec4 A, Iu16vec4 B);
R0 = A2; R1 = B2;
R2 = A3; R3 = B3;

Corresponding intrinsic: _mm_unpackhi_pi16

Interleave the four 8-bit values from the high half of A with the four 8-bit values from the high half of B.
I8vec8 unpack_high(I8vec8 A, I8vec8 B);
I8vec8 unpack_high(Is8vec8 A, I8vec8 B);
I8vec8 unpack_high(Iu8vec8 A, I8vec8 B);
R0 = A4;
R1 = B4;
R2 = A5;
R3 = B5;
R4 = A6;
R5 = B6;
R6 = A7;
R7 = B7;

**Corresponding intrinsic:** _mm_unpackhi_pi8
Interleave the sixteen 8-bit values from the high half of A with the four 8-bit values from the high half of B.

I8vec16 unpack_high(I8vec16 A, I8vec16 B);
I8vec16 unpack_high(Is8vec16 A, I8vec16 B);
I8vec16 unpack_high(Iu8vec16 A, I8vec16 B);
R0 = A8;
R1 = B8;
R2 = A9;
R3 = B9;
R4 = A10;
R5 = B10;
R6 = A11;
R7 = B11;
R8 = A12;
R8 = B12;
R2 = A13;
R3 = B13;
R4 = A14;
R5 = B14;
R6 = A15;
R7 = B15;

**Corresponding intrinsic:** _mm_unpackhi_epi16
Interleave the 32-bit value from the low half of A with the 32-bit value from the low half of B

I64vec2 unpack_low(I64vec2 A, I64vec2 B);
I64vec2 unpack_low(Is64vec2 A, Is64vec2 B);
I64vec2 unpack_low(Iu64vec2 A, Iu64vec2 B);
R0 = A0;
R1 = B0;

**Corresponding intrinsic:** _mm_unpacklo_epi32
Interleave the 64-bit value from the low half of A with the 64-bit values from the low half of B

I64vec2 unpack_low(I64vec2 A, I64vec2 B);
I64vec2 unpack_low(Is64vec2 A, Is64vec2 B);
I64vec2 unpack_low(Iu64vec2 A, Iu64vec2 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;

**Corresponding intrinsic:** _mm_unpacklo_epi32
Interleave the two 32-bit values from the low half of A with the two 32-bit values from the low half of B

I64vec2 unpack_low(I64vec2 A, I64vec2 B);
I64vec2 unpack_low(Is64vec2 A, Is64vec2 B);
I64vec2 unpack_low(Iu64vec2 A, Iu64vec2 B);
I32vec4 unpack_low(I32vec4 A, I32vec4 B);
Is32vec4 unpack_low(Is32vec4 A, Is32vec4 B);
Iu32vec4 unpack_low(Iu32vec4 A, Iu32vec4 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
Corresponding intrinsic: _mm_unpacklo_epi32
Interleave the 32-bit value from the low half of A with the 32-bit value from the low half of B.

I32vec2 unpack_low(I32vec2 A, I32vec2 B);
Is32vec2 unpack_low(Is32vec2 A, Is32vec2 B);
Iu32vec2 unpack_low(Iu32vec2 A, Iu32vec2 B);
R0 = A0;
R1 = B0;
Corresponding intrinsic: _mm_unpacklo_pi32
Interleave the two 16-bit values from the low half of A with the two 16-bit values from the low half of B.

I16vec8 unpack_low(I16vec8 A, I16vec8 B);
Is16vec8 unpack_low(Is16vec8 A, Is16vec8 B);
Iu16vec8 unpack_low(Iu16vec8 A, Iu16vec8 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
R4 = A2;
R5 = B2;
R6 = A3;
R7 = B3;
Corresponding intrinsic: _mm_unpacklo_epi16
Interleave the two 16-bit values from the low half of A with the two 16-bit values from the low half of B.

I16vec4 unpack_low(I16vec4 A, I16vec4 B);
Is16vec4 unpack_low(Is16vec4 A, Is16vec4 B);
Iu16vec4 unpack_low(Iu16vec4 A, Iu16vec4 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
Corresponding intrinsic: _mm_unpacklo_pi16
Interleave the four 8-bit values from the high low of A with the four 8-bit values from the low half of B.

I8vec16 unpack_low(I8vec16 A, I8vec16 B);
Is8vec16 unpack_low(Is8vec16 A, Is8vec16 B);
Iu8vec16 unpack_low(Iu8vec16 A, Iu8vec16 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
R4 = A2;
R5 = B2;
R6 = A3;
R7 = B3;
R8 = A4;
R9 = B4;
R10 = A5;
R11 = B5;
R12 = A6;
R13 = B6;
R14 = A7;
R15 = B7;

Corresponding intrinsic: _mm_unpacklo_epi8

Interleave the four 8-bit values from the high low of A with the four 8-bit values from the low half of B.

I8vec8 unpack_low(I8vec8 A, I8vec8 B);
Is8vec8 unpack_low(Is8vec8 A, Is8vec8 B);
Iu8vec8 unpack_low(Iu8vec8 A, Iu8vec8 B);

Pack Operators

Pack the eight 32-bit values found in A and B into eight 16-bit values with signed saturation.

Is16vec8 pack_sat(Is32vec2 A, Is32vec2 B);
Corresponding intrinsic: _mm_packs_epi32

Pack the four 32-bit values found in A and B into eight 16-bit values with signed saturation.

Is16vec4 pack_sat(Is32vec2 A, Is32vec2 B);
Corresponding intrinsic: _mm_packs_pi32

Pack the sixteen 16-bit values found in A and B into sixteen 8-bit values with signed saturation.

Is8vec16 pack_sat(Is16vec4 A, Is16vec4 B);
Corresponding intrinsic: _mm_packs_epi16

Pack the eight 16-bit values found in A and B into eight 8-bit values with signed saturation.

Is8vec8 pack_sat(Is16vec4 A, Is16vec4 B);
Corresponding intrinsic: _mm_packs_pi16

Pack the sixteen 16-bit values found in A and B into sixteen 8-bit values with unsigned saturation.

Iu8vec16 packu_sat(Is16vec4 A, Is16vec4 B);
Corresponding intrinsic: _mm_packus_epi16

Pack the eight 16-bit values found in A and B into eight 8-bit values with unsigned saturation.

Iu8vec8 packu_sat(Is16vec4 A, Is16vec4 B);
Corresponding intrinsic: _mm_packs_pu16
Clear MMX™ State Operator

Empty the MMX™ registers and clear the MMX state. Read the guidelines for using the EMMS instruction intrinsic.

```c
void empty(void);
```

Corresponding intrinsic: `__mm_empty`

### Integer Functions for Streaming SIMD Extensions

**NOTE**
You must include `fvec.h` header file for the following functionality.

Compute the element-wise maximum of the respective signed integer words in \( A \) and \( B \).

```c
Is16vec4 simd_max(Is16vec4 A, Is16vec4 B);
```

Corresponding intrinsic: `__mm_max_pi16`

Compute the element-wise minimum of the respective signed integer words in \( A \) and \( B \).

```c
Is16vec4 simd_min(Is16vec4 A, Is16vec4 B);
```

Corresponding intrinsic: `__mm_min_pi16`

Compute the element-wise maximum of the respective unsigned bytes in \( A \) and \( B \).

```c
Iu8vec8 simd_max(Iu8vec8 A, Iu8vec8 B);
```

Corresponding intrinsic: `__mm_max_pu8`

Compute the element-wise minimum of the respective unsigned bytes in \( A \) and \( B \).

```c
Iu8vec8 simd_min(Iu8vec8 A, Iu8vec8 B);
```

Corresponding intrinsic: `__mm_min_pu8`

Create an 8-bit mask from the most significant bits of the bytes in \( A \).

```c
int move_mask(I8vec8 A);
```

Corresponding intrinsic: `__mm_movemask_pi8`

Conditionally store byte elements of \( A \) to address \( p \). The high bit of each byte in the selector \( B \) determines whether the corresponding byte in \( A \) will be stored.

```c
void mask_move(I8vec8 A, I8vec8 B, signed char *p);
```

Corresponding intrinsic: `__mm_maskmove_si64`

Store the data in \( A \) to the address \( p \) without polluting the caches. \( A \) can be any Ivec type.

```c
void store_nta(__m64 *p, M64 A);
```

Corresponding intrinsic: `__mm_stream_pi`

Compute the element-wise average of the respective unsigned 8-bit integers in \( A \) and \( B \).

```c
Iu8vec8 simd_avg(Iu8vec8 A, Iu8vec8 B);
```

Corresponding intrinsic: `__mm_avg_pu8`

Compute the element-wise average of the respective unsigned 16-bit integers in \( A \) and \( B \).

```c
Iu16vec4 simd_avg(Iu16vec4 A, Iu16vec4 B);
```

Corresponding intrinsic: `__mm_avg_pu16`

### Conversions between Fvec and Ivec

Convert the lower double-precision floating-point value of \( A \) to a 32-bit integer with truncation.

```c
int F64vec2ToInt(F64vec42 A);
```

\( r := \text{(int)A0}; \)
Convert the four floating-point values of \( A \) to two the two least significant double-precision floating-point values.

\[
\text{F64vec2 } \text{F32vec4ToF64vec2(F32vec4 } A) ;
\]
\[
r0 := \text{(double)}A0 ;
\]
\[
r1 := \text{(double)}A1 ;
\]

Convert the two double-precision floating-point values of \( A \) to two single-precision floating-point values.

\[
\text{F32vec4 } \text{F64vec2ToF32vec4(F64vec2 } A) ;
\]
\[
r0 := \text{(float)}A0 ;
\]
\[
r1 := \text{(float)}A1 ;
\]

Convert the signed \( \text{int} \) in \( B \) to a double-precision floating-point value and pass the upper double-precision value from \( A \) through to the result.

\[
\text{F64vec2 } \text{InttoF64vec2(F64vec2 } A, \text{ int } B) ;
\]
\[
r0 := \text{(double)}B ;
\]
\[
r1 := A1 ;
\]

Convert the lower floating-point value of \( A \) to a 32-bit integer with truncation.

\[
\text{int } \text{F32vec4ToInt(F32vec4 } A) ;
\]
\[
r := \text{(int)}A0 ;
\]

Convert the two lower floating-point values of \( A \) to two 32-bit integer with truncation, returning the integers in packed form.

\[
\text{Is32vec2 } \text{F32vec4ToIs32vec2 (F32vec4 } A) ;
\]
\[
r0 := \text{(int)}A0 ;
\]
\[
r1 := \text{(int)}A1 ;
\]

Convert the 32-bit integer value \( B \) to a floating-point value; the upper three floating-point values are passed through from \( A \).

\[
\text{F32vec4 } \text{IntToF32vec4(F32vec4 } A, \text{ int } B) ;
\]
\[
r0 := \text{(float)}B ;
\]
\[
r1 := A1 ;
\]
\[
r2 := A2 ;
\]
\[
r3 := A3 ;
\]

Convert the two 32-bit integer values in packed form in \( B \) to two floating-point values; the upper two floating-point values are passed through from \( A \).

\[
\text{F32vec4 } \text{Is32vec2ToF32vec4(F32vec4 } A, \text{ Is32vec2 } B) ;
\]
\[
r0 := \text{(float)}B0 ;
\]
\[
r1 := \text{(float)}B1 ;
\]
\[
r2 := A2 ;
\]
\[
r3 := A3 ;
\]

**Floating-point Vector Classes**

The floating-point vector classes, \( \text{F64vec2} \), \( \text{F32vec4} \), and \( \text{F32vec1} \), provide an interface to SIMD operations. The class specifications are as follows:

\[
\text{F64vec2 } A(\text{double } x, \text{ double } y) ;
\]
\[
\text{F32vec4 } A(\text{float } z, \text{ float } y, \text{ float } x, \text{ float } w) ;
\]
\[
\text{F32vec1 } B(\text{float } w) ;
\]

The packed floating-point input values are represented with the right-most value lowest as shown in the following table.

**Single-Precision Floating-point Elements**
**Fvec Notation Conventions**

This reference uses the following conventions for syntax and return values.

**Fvec Classes Syntax Notation**

Fvec classes use the syntax conventions shown in the following examples:

\[
\]

**Example 1:**

\[
F64vec2 R = F64vec2 A \& F64vec2 B;
\]

**Example 2:**

\[
[Fvec\_Class] R = [operator]([Fvec\_Class] A,[Fvec\_Class] B);
\]

**Example 3:**

\[
F64vec2 R &= F64vec2 A;
\]

where

[operator] is an operator (for example, & , |, or ^ )

[Fvec\_Class] is any Fvec class ( F64vec2, F32vec4, or F32vec1 )

R, A, B are declared Fvec variables of the type indicated.

**Return Value Notation**

Because the Fvec classes have packed elements, the return values typically follow the conventions presented in the Return Value Convention Notation Mappings table. F32vec4 returns four single-precision, floating-point values (R0, R1, R2, and R3); F64vec2 returns two double-precision, floating-point values, and F32vec1 returns the lowest single-precision floating-point value (R0).

**Return Value Convention Notation Mappings**

<table>
<thead>
<tr>
<th>Example 1:</th>
<th>Example 2:</th>
<th>Example 3:</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 := A0 &amp; B0;</td>
<td>R0 := A0 andnot B0;</td>
<td>R0 &amp;= A0;</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

F32vec4 returns **four** packed single-precision floating point values (R0, R1, R2, and R3). F32vec2 returns **one** single-precision floating point value (R0).
Data Alignment

Memory operations using the Intel® Streaming SIMD Extensions should be performed on 16-byte-aligned data whenever possible. Memory operations using the Intel® Advanced Vector Extensions should be performed on 32-byte-aligned data whenever possible.

F32vec4 and F64vec2 object variables are properly aligned by default. Note that floating point arrays are not automatically aligned. To get 16-byte alignment, you can use the alignment __declspec:

```c
__declspec( align(16) ) float A[4];
```

Conversions

All Fvec object variables can be implicitly converted to __m128 data types. For example, the results of computations performed on F32vec4 or F32vec1 object variables can be assigned to __m128 data types.

```c
__m128d mm = A & B; /* where A,B are F64vec2 object variables */
__m128 mm = A & B; /* where A,B are F32vec4 object variables */
__m128 mm = A & B; /* where A,B are F32vec1 object variables */
```

Constructors and Initialization

The following table shows how to create and initialize F32vec objects with the Fvec classes.

### Constructors and Initialization for Fvec Classes

<table>
<thead>
<tr>
<th>Example</th>
<th>Intrinsic</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constructor Declaration</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F64vec2 A;</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>F32vec4 B;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F32vec1 C;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>__m128 Object Initialization</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F64vec2 A(__m128d mm);</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>F32vec4 B(__m128 mm);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F32vec1 C(__m128 mm);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Double Initialization</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>/* Initializes two doubles. */</td>
<td>_mm_set_pd</td>
<td></td>
</tr>
<tr>
<td>F64vec2 A(double d0, double d1);</td>
<td>A0 := d0;</td>
<td>A1 := d1;</td>
</tr>
</tbody>
</table>
## Double Initialization

F64vec2 A = F64vec2(double d0, double d1);

F64vec2 A(double d0);  
  _mm_set1_pd
  A0 := d0;  
  /* Initializes both return values with the same double precision value */.

## Float Initialization

F32vec4 A(float f3, float f2, float f1, float f0);

F32vec4 A = F32vec4(float f3, float f2, float f1, float f0);

F32vec4 A(float f0);  
  _mm_set_ps
  A0 := f0;  
  A1 := f0;  
  A2 := f0;  
  A3 := f0;  
  /* Initializes all return values with the same floating point value. */

F32vec4 A(double d0);  
  _mm_set1_ps(d)
  A0 := d0;  
  A1 := d0;  
  A2 := d0;  
  A3 := d0;  
  /* Initializes all return values with the same double-precision value. */

F32vec1 A(double d0);  
  _mm_set_ss(d)
  A0 := d0;  
  A1 := 0;  
  A2 := 0;  
  A3 := 0;  
  /* Initializes the lowest value of A with d0 and the other values with 0. */

F32vec1 B(float f0);  
  _mm_set_ss
  B0 := f0;  
  B1 := 0;  
  B2 := 0;  
  B3 := 0;  
  /* Initializes the lowest value of B with f0 and the other values with 0. */

F32vec1 B(int I);  
  _mm_cvtsi32_ss
  B0 := f0;  
  B1 := {}  
  B2 := {}  
  B3 := {}  
  /* Initializes the lowest value of B with f0, other values are undefined. */

## Arithmetic Operators

The following table lists the arithmetic operators of the Fvec classes and generic syntax. The operators have been divided into standard and advanced operations, which are described in more detail later in this section.
## Fvec Arithmetic Operators

<table>
<thead>
<tr>
<th>Category</th>
<th>Operation</th>
<th>Operators</th>
<th>Generic Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>Addition</td>
<td>+</td>
<td>R = A + B; R += A;</td>
</tr>
<tr>
<td></td>
<td>Subtraction</td>
<td>-</td>
<td>R = A - B; R -= A;</td>
</tr>
<tr>
<td></td>
<td>Multiplication</td>
<td>*</td>
<td>R = A * B; R *= A;</td>
</tr>
<tr>
<td></td>
<td>Division</td>
<td>/</td>
<td>R = A / B; R /= A;</td>
</tr>
<tr>
<td>Advanced</td>
<td>Square Root</td>
<td>sqrt</td>
<td>R = sqrt(A);</td>
</tr>
<tr>
<td></td>
<td>Reciprocal</td>
<td>rcp</td>
<td>R = rcp(A);</td>
</tr>
<tr>
<td></td>
<td>(Newton-Raphson)</td>
<td>rcp_nr</td>
<td>R = rcp_nr(A);</td>
</tr>
<tr>
<td></td>
<td>Reciprocal Square Root</td>
<td>rsqrt</td>
<td>R = rsqrt(A);</td>
</tr>
<tr>
<td></td>
<td>(Newton-Raphson)</td>
<td>rsqrt_nr</td>
<td>R = rsqrt_nr(A);</td>
</tr>
</tbody>
</table>

### Standard Arithmetic Operator Usage

The following two tables show the return values for each class of the standard arithmetic operators, which use the syntax styles described earlier in the Return Value Notation section.

#### Standard Arithmetic Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>Operators</th>
<th>B</th>
<th>F32vec</th>
<th>F64vec</th>
<th>F32vec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>R0:=</td>
<td>A0</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>B0</td>
</tr>
<tr>
<td>R1:=</td>
<td>A1</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>B1</td>
</tr>
<tr>
<td>R2:=</td>
<td>A2</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>B2</td>
</tr>
<tr>
<td>R3:=</td>
<td>A3</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>B3</td>
</tr>
</tbody>
</table>

#### Arithmetic with Assignment Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>Operators</th>
<th>A</th>
<th>F32vec</th>
<th>F64vec</th>
<th>F32vec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>R0:=</td>
<td>+=</td>
<td>-=</td>
<td>*=</td>
<td>/=</td>
<td>A0</td>
</tr>
<tr>
<td>R1:=</td>
<td>+=</td>
<td>-=</td>
<td>*=</td>
<td>/=</td>
<td>A1</td>
</tr>
<tr>
<td>R2:=</td>
<td>+=</td>
<td>-=</td>
<td>*=</td>
<td>/=</td>
<td>A2</td>
</tr>
<tr>
<td>R3:=</td>
<td>+=</td>
<td>-=</td>
<td>*=</td>
<td>/=</td>
<td>A3</td>
</tr>
</tbody>
</table>

This table lists standard arithmetic operator syntax and intrinsics.

### Standard Arithmetic Operations for Fvec Classes
<table>
<thead>
<tr>
<th>Operation</th>
<th>Returns</th>
<th>Example Syntax</th>
<th>Intrinsic Usage</th>
</tr>
</thead>
</table>
| Addition  | 4 floats | F32vec4 R = F32vec4  
A + F32vec4 B;  
F32vec4 R +=  
F32vec4 A; | _mm_add_ps |
| 2 doubles | F64vec2 R = F64vec2  
A + F32vec2 B;  
F64vec2 R +=  
F64vec2 A; | _mm_add_pd |
| 1 float   | F32vec1 R = F32vec1  
A + F32vec1 B;  
F32vec1 R +=  
F32vec1 A; | _mm_add_ss |
| Subtraction | 4 floats | F32vec4 R = F32vec4  
A - F32vec4 B;  
F32vec4 R -=  
F32vec4 A; | _mm_sub_ps |
| 2 doubles | F64vec2 R = F64vec2  
A - F32vec2 B;  
F64vec2 R -=  
F64vec2 A; | _mm_sub_pd |
| 1 float   | F32vec1 R = F32vec1  
A - F32vec1 B;  
F32vec1 R -=  
F32vec1 A; | _mm_sub_ss |
| Multiplication | 4 floats | F32vec4 R = F32vec4  
A * F32vec4 B;  
F32vec4 R *=  
F32vec4 A; | _mm_mul_ps |
| 2 doubles | F64vec2 R = F64vec2  
A * F364vec2 B;  
F64vec2 R *=  
F64vec2 A; | _mm_mul_pd |
| 1 float   | F32vec1 R = F32vec1  
A * F32vec1 B;  
F32vec1 R *=  
F32vec1 A; | _mm_mul_ss |
| Division  | 4 floats | F32vec4 R = F32vec4  
A / F32vec4 B;  
F32vec4 R /=  
F32vec4 A; | _mm_div_ps |
### Advanced Arithmetic Operator Usage

The following table shows the return values classes of the advanced arithmetic operators, which use the syntax styles described earlier in the Return Value Notation section.

#### Advanced Arithmetic Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>Operators</th>
<th>A</th>
<th>F32vec 4</th>
<th>F64vec 2</th>
<th>F32vec 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:=</td>
<td>sqrt rcp rsqrt rcp_nr rsqrt_nr</td>
<td>A0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>R1:=</td>
<td>sqrt rcp rsqrt rcp_nr rsqrt_nr</td>
<td>A1</td>
<td>X</td>
<td>X</td>
<td>N/A</td>
</tr>
<tr>
<td>R2:=</td>
<td>sqrt rcp rsqrt rcp_nr rsqrt_nr</td>
<td>A2</td>
<td>X</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>R3:=</td>
<td>sqrt rcp rsqrt rcp_nr rsqrt_nr</td>
<td>A3</td>
<td>X</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>f :=</td>
<td>add_horizontal</td>
<td>(A0 + A1 + A2 + A3)</td>
<td>X</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>d :=</td>
<td>add_horizontal</td>
<td>(A0 + A1)</td>
<td>N/A</td>
<td>X</td>
<td>N/A</td>
</tr>
</tbody>
</table>

This table shows examples for advanced arithmetic operators.

#### Advanced Arithmetic Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square Root</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = sqrt(F32vec4 A);</td>
<td>mm_sqrt_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = sqrt(F64vec2 A);</td>
<td>mm_sqrt_pd</td>
</tr>
<tr>
<td>Returns</td>
<td>Example Syntax Usage</td>
<td>Intrinsic</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td><strong>Square Root</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = sqrt(F32vec1 A);</td>
<td>_mm_sqrt_ss</td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = rcp(F32vec4 A);</td>
<td>_mm_rcp_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = rcp(F64vec2 A);</td>
<td>_mm_rcp_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = rcp(F32vec1 A);</td>
<td>_mm_rcp_ss</td>
</tr>
<tr>
<td><strong>Reciprocal Square Root</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = rsqrt(F32vec4 A);</td>
<td>_mm_rsqrt_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = rsqrt(F64vec2 A);</td>
<td>_mm_rsqrt_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = rsqrt(F32vec1 A);</td>
<td>_mm_rsqrt_ss</td>
</tr>
<tr>
<td><strong>Reciprocal Newton Raphson</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = rcp_nr(F32vec4 A);</td>
<td>_mm_sub_ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_add_ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_mul_ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_rcp_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = rcp_nr(F64vec2 A);</td>
<td>_mm_sub_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_add_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_mul_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_rcp_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = rcp_nr(F32vec1 A);</td>
<td>_mm_sub_ss</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_add_ss</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_mul_ss</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_rcp_ss</td>
</tr>
<tr>
<td><strong>Reciprocal Square Root Newton Raphson</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 float</td>
<td>F32vec4 R = rsqrt_nr(F32vec4 A);</td>
<td>_mm_sub_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_mul_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_rsqrt_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = rsqrt_nr(F64vec2 A);</td>
<td>_mm_sub_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_mul_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_mm_rsqrt_pd</td>
</tr>
</tbody>
</table>
Reciprocal Square Root Newton Raphson

1 float
F32vec1 R = rsqrt_nr(F32vec1 A);
_mm_sub_ss
_mm_mul_ss
_mm_rsqrt_ss

Horizontal Add

1 float
float f = add_horizontal(F32vec4 A);
_mm_add_ss
_mm_shuffle_ss

1 double
double d = add_horizontal(F64vec2 A);
_mm_add_sd
_mm_shuffle_sd

Minimum and Maximum Operators

Compute the minimums of the two double precision floating-point values of A and B.

F64vec2 R = simd_min(F64vec2 A, F64vec2 B)
R0 := min(A0,B0);
R1 := min(A1,B1);

Corresponding intrinsic: _mm_min_pd

Compute the minimums of the four single precision floating-point values of A and B.

F32vec4 R = simd_min(F32vec4 A, F32vec4 B)
R0 := min(A0,B0);
R1 := min(A1,B1);
R2 := min(A2,B2);
R3 := min(A3,B3);

Corresponding intrinsic: _mm_min_ps

Compute the minimum of the lowest single precision floating-point values of A and B.

F32vec1 R = simd_min(F32vec1 A, F32vec1 B)
R0 := min(A0,B0);

Corresponding intrinsic: _mm_min_ss

Compute the maximums of the two double precision floating-point values of A and B.

F64vec2 simd_max(F64vec2 A, F64vec2 B)
R0 := max(A0,B0);
R1 := max(A1,B1);

Corresponding intrinsic: _mm_max_pd

Compute the maximums of the four single precision floating-point values of A and B.

F32vec4 R = simd_max(F32vec4 A, F32vec4 B)
R0 := max(A0,B0);
R1 := max(A1,B1);
R2 := max(A2,B2);
R3 := max(A3,B3);

Corresponding intrinsic: _mm_max_ps

Compute the maximum of the lowest single precision floating-point values of A and B.

F32vec1 simd_max(F32vec1 A, F32vec1 B)
R0 := max(A0,B0);

Corresponding intrinsic: _mm_max_ss
Logical Operators

The following table lists the logical operators of the Fvec classes and generic syntax. The logical operators for F32vec1 classes use only the lower 32 bits.

Fvec Logical Operators Return Value Mapping

<table>
<thead>
<tr>
<th>Bitwise Operation</th>
<th>Operators</th>
<th>Generic Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>&amp;</td>
<td>R = A &amp; B;</td>
</tr>
<tr>
<td></td>
<td>&amp;=</td>
<td>R &amp;= A;</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>=</td>
</tr>
<tr>
<td><strong>XOR</strong></td>
<td>^</td>
<td>R = A ^ B;</td>
</tr>
<tr>
<td></td>
<td>^=</td>
<td>R ^= A;</td>
</tr>
<tr>
<td>andnot</td>
<td>andnot</td>
<td>R = andnot(A);</td>
</tr>
</tbody>
</table>

The following table lists standard logical operators syntax and corresponding intrinsics. Note that there is no corresponding scalar intrinsic for the F32vec1 classes, which accesses the lower 32 bits of the packed vector intrinsics.

Logical Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returns</th>
<th>Example Syntax</th>
<th>Intrinsic Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>4 floats</td>
<td>F32vec4 &amp; = F32vec4 A &amp; F32vec4 B;</td>
<td>_mm_and_ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 &amp; &amp;= F32vec4 A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>F64vec2 R = F64vec2 A &amp; F64vec2 B;</td>
<td>_mm_and_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 R &amp;= F64vec2 A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 float</td>
<td>F32vec1 R = F32vec1 A &amp; F32vec1 B;</td>
<td>_mm_and_ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 R &amp;= F32vec1 A;</td>
<td></td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>4 floats</td>
<td>F32vec4 R = F32vec4 A</td>
<td>F32vec4 B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 R</td>
<td>= F32vec4 A;</td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>F64vec2 R = F64vec2 A</td>
<td>F64vec2 B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 R</td>
<td>= F64vec2 A;</td>
</tr>
<tr>
<td></td>
<td>1 float</td>
<td>F32vec1 R = F32vec1 A</td>
<td>F32vec1 B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 R</td>
<td>= F32vec1 A;</td>
</tr>
<tr>
<td><strong>XOR</strong></td>
<td>4 floats</td>
<td>F32vec4 R = F32vec4 A ^ F32vec4 B;</td>
<td>_mm_xor_ps</td>
</tr>
</tbody>
</table>
### Compare Operators

The operators described in this section compare the single precision floating-point values of \( A \) and \( B \). Comparison between objects of any `Fvec` class return the same class being compared.

The following table lists the compare operators for the `Fvec` classes.

#### Compare Operators and Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Compare For:</th>
<th>Operators</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>cmpeq</td>
<td>( R = \text{cmpeq}(A, B) )</td>
</tr>
<tr>
<td>Inequality</td>
<td>cmpneq</td>
<td>( R = \text{cmpneq}(A, B) )</td>
</tr>
<tr>
<td>Greater Than</td>
<td>cmpgt</td>
<td>( R = \text{cmpgt}(A, B) )</td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>cmpge</td>
<td>( R = \text{cmpge}(A, B) )</td>
</tr>
<tr>
<td>Not Greater Than</td>
<td>cmpngt</td>
<td>( R = \text{cmpngt}(A, B) )</td>
</tr>
<tr>
<td>Not Greater Than or Equal To</td>
<td>cmpnge</td>
<td>( R = \text{cmpnge}(A, B) )</td>
</tr>
<tr>
<td>Less Than</td>
<td>cmplt</td>
<td>( R = \text{cmplt}(A, B) )</td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>cmple</td>
<td>( R = \text{cmple}(A, B) )</td>
</tr>
<tr>
<td>Not Less Than</td>
<td>cmpnlt</td>
<td>( R = \text{cmpnlt}(A, B) )</td>
</tr>
<tr>
<td>Not Less Than or Equal To</td>
<td>cmpnle</td>
<td>( R = \text{cmpnle}(A, B) )</td>
</tr>
</tbody>
</table>

#### Compare Operators

The mask is set to \( 0xffffffff \) for each floating-point value where the comparison is true and \( 0x00000000 \) where the comparison is false. The following table shows the return values for each class of the compare operators, which use the syntax described earlier in the Return Value Notation section.
Compare Operator Return Value Mapping

<table>
<thead>
<tr>
<th>( R )</th>
<th>( A_0 )</th>
<th>For Any Operators</th>
<th>( B )</th>
<th>If True</th>
<th>If False</th>
<th>( F_{32vec} )</th>
<th>( F_{64vec} )</th>
<th>( F_{32vec} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_0 ) := ( 1 ) ( \neg ) ( A )</td>
<td>( 1 ) ( \neg ) ( A ) ( \text{cmp}[\text{eq}</td>
<td>\text{lt}</td>
<td>\text{le}</td>
<td>\text{gt}</td>
<td>\text{ge}] ) ( \text{cmp}[\text{ne}</td>
<td>\text{nlt}</td>
<td>\text{nle}</td>
<td>\text{ngt}</td>
</tr>
<tr>
<td>( R_1 ) := ( 1 ) ( \neg ) ( A )</td>
<td>( 1 ) ( \neg ) ( A ) ( \text{cmp}[\text{eq}</td>
<td>\text{lt}</td>
<td>\text{le}</td>
<td>\text{gt}</td>
<td>\text{ge}] ) ( \text{cmp}[\text{ne}</td>
<td>\text{nlt}</td>
<td>\text{nle}</td>
<td>\text{ngt}</td>
</tr>
<tr>
<td>( R_2 ) := ( 1 ) ( \neg ) ( A )</td>
<td>( 1 ) ( \neg ) ( A ) ( \text{cmp}[\text{eq}</td>
<td>\text{lt}</td>
<td>\text{le}</td>
<td>\text{gt}</td>
<td>\text{ge}] ) ( \text{cmp}[\text{ne}</td>
<td>\text{nlt}</td>
<td>\text{nle}</td>
<td>\text{ngt}</td>
</tr>
<tr>
<td>( R_3 ) := ( 1 ) ( \neg ) ( A )</td>
<td>( 1 ) ( \neg ) ( A ) ( \text{cmp}[\text{eq}</td>
<td>\text{lt}</td>
<td>\text{le}</td>
<td>\text{gt}</td>
<td>\text{ge}] ) ( \text{cmp}[\text{ne}</td>
<td>\text{nlt}</td>
<td>\text{nle}</td>
<td>\text{ngt}</td>
</tr>
</tbody>
</table>

The following table shows examples for arithmetic operators and intrinsics.

Compare Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compare for Equality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>( F_{32vec} ) ( R = \text{cmpeq}(F_{32vec} \ A) );</td>
<td>_mm_cmpeq_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>( F_{64vec} ) ( R = \text{cmpeq}(F_{64vec} \ A) );</td>
<td>_mm_cmpeq_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>( F_{32vec} ) ( R = \text{cmpeq}(F_{32vec} \ A) );</td>
<td>_mm_cmpeq_ss</td>
</tr>
<tr>
<td><strong>Compare for Inequality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>( F_{32vec} ) ( R = \text{cmpneq}(F_{32vec} \ A) );</td>
<td>_mm_cmpneq_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>( F_{64vec} ) ( R = \text{cmpneq}(F_{64vec} \ A) );</td>
<td>_mm_cmpneq_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>( F_{32vec} ) ( R = \text{cmpneq}(F_{32vec} \ A) );</td>
<td>_mm_cmpneq_ss</td>
</tr>
<tr>
<td>Compare for Less Than</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>4 floats</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>F32vec4 R = cmplt(F32vec4 A);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_mm_cmplt_ps</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2 doubles</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>F64vec2 R = cmplt(F64vec2 A);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_mm_cmplt_pd</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1 float</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>F32vec1 R = cmplt(F32vec1 A);</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>_mm_cmplt_ss</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Less Than or Equal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4 floats</strong></td>
</tr>
<tr>
<td><code>F32vec4 R = cmple(F32vec4 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmple_ps</code></td>
</tr>
<tr>
<td><strong>2 doubles</strong></td>
</tr>
<tr>
<td><code>F64vec2 R = cmple(F64vec2 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmple_pd</code></td>
</tr>
<tr>
<td><strong>1 float</strong></td>
</tr>
<tr>
<td><code>F32vec1 R = cmple(F32vec1 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmple_pd</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Greater Than</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4 floats</strong></td>
</tr>
<tr>
<td><code>F32vec4 R = cmpgt(F32vec4 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmpgt_ps</code></td>
</tr>
<tr>
<td><strong>2 doubles</strong></td>
</tr>
<tr>
<td><code>F64vec2 R = cmpgt(F64vec2 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmpgt_pd</code></td>
</tr>
<tr>
<td><strong>1 float</strong></td>
</tr>
<tr>
<td><code>F32vec1 R = cmpgt(F32vec1 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmpgt_ss</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Greater Than or Equal To</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4 floats</strong></td>
</tr>
<tr>
<td><code>F32vec4 R = cmpge(F32vec4 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmpge_ps</code></td>
</tr>
<tr>
<td><strong>2 doubles</strong></td>
</tr>
<tr>
<td><code>F64vec2 R = cmpge(F64vec2 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmpge_pd</code></td>
</tr>
<tr>
<td><strong>1 float</strong></td>
</tr>
<tr>
<td><code>F32vec1 R = cmpge(F32vec1 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmpge_ss</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Not Less Than</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4 floats</strong></td>
</tr>
<tr>
<td><code>F32vec4 R = cmpnlt(F32vec4 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmpnlt_ps</code></td>
</tr>
<tr>
<td><strong>2 doubles</strong></td>
</tr>
<tr>
<td><code>F64vec2 R = cmpnlt(F64vec2 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmpnlt_pd</code></td>
</tr>
<tr>
<td><strong>1 float</strong></td>
</tr>
<tr>
<td><code>F32vec1 R = cmpnlt(F32vec1 A);</code></td>
</tr>
<tr>
<td><code>_mm_cmpnlt_ss</code></td>
</tr>
</tbody>
</table>
### Conditional Select Operators for Fvec Classes

Each conditional function compares single-precision floating-point values of A and B. The C and D parameters are used for return value. Comparison between objects of any Fvec class returns the same class.

#### Conditional Select Operators for Fvec Classes

**Conditional Select for:**

<table>
<thead>
<tr>
<th>Operators</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>select_eq</td>
</tr>
<tr>
<td>Inequality</td>
<td>select_neq</td>
</tr>
<tr>
<td>Greater Than</td>
<td>select_gt</td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>select_ge</td>
</tr>
<tr>
<td>Not Greater Than</td>
<td>select_gt</td>
</tr>
<tr>
<td>Not Greater Than or Equal To</td>
<td>select_ge</td>
</tr>
<tr>
<td>Less Than</td>
<td>select_lt</td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>select_le</td>
</tr>
</tbody>
</table>
### Conditional Select for:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Less Than</td>
<td>R = select_nlt(A, B)</td>
</tr>
<tr>
<td>Not Less Than or Equal To</td>
<td>R = select_nle(A, B)</td>
</tr>
</tbody>
</table>

### Conditional Select Operator Usage

For conditional select operators, the return value is stored in C if the comparison is true or in D if false. The following table shows the return values for each class of the conditional select operators, using the Return Value Notation described earlier.

#### Compare Operator Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>A0</th>
<th>Operators</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 :=</td>
<td>(A1</td>
<td>select_eq</td>
<td>lt</td>
<td>le</td>
<td>ge</td>
<td>A1)</td>
<td>B0)</td>
<td>C0</td>
</tr>
<tr>
<td></td>
<td>!(A1</td>
<td>select_ne</td>
<td>nlt</td>
<td>nle</td>
<td>ngt</td>
<td></td>
<td>B0)</td>
<td>C0</td>
</tr>
<tr>
<td>R1 :=</td>
<td>(A2</td>
<td>select_eq</td>
<td>lt</td>
<td>le</td>
<td>ge</td>
<td>A2)</td>
<td>B1)</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td>!(A2</td>
<td>select_ne</td>
<td>nlt</td>
<td>nle</td>
<td>ngt</td>
<td></td>
<td>B1)</td>
<td>C1</td>
</tr>
<tr>
<td>R2 :=</td>
<td>(A2</td>
<td>select_eq</td>
<td>lt</td>
<td>le</td>
<td>ge</td>
<td>A2)</td>
<td>B2)</td>
<td>C2</td>
</tr>
<tr>
<td></td>
<td>!(A2</td>
<td>select_ne</td>
<td>nlt</td>
<td>nle</td>
<td>ngt</td>
<td></td>
<td>B2)</td>
<td>C2</td>
</tr>
<tr>
<td>R3 :=</td>
<td>(A3</td>
<td>select_eq</td>
<td>lt</td>
<td>le</td>
<td>ge</td>
<td>A3)</td>
<td>B3)</td>
<td>C3</td>
</tr>
<tr>
<td></td>
<td>!(A3</td>
<td>select_ne</td>
<td>nlt</td>
<td>nle</td>
<td>ngt</td>
<td></td>
<td>B3)</td>
<td>C3</td>
</tr>
</tbody>
</table>

The following table shows examples for conditional select operations and corresponding intrinsics.

#### Conditional Select Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare for Equality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = select_eq(F32vec4 A);</td>
<td>_mm_cmpeq_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = select_eq(F64vec2 A);</td>
<td>_mm_cmpeq_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = select_eq(F32vec1 A);</td>
<td>_mm_cmpeq_ss</td>
</tr>
</tbody>
</table>
### Compare for Inequality

<table>
<thead>
<tr>
<th>Type</th>
<th>Region</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec4 R =</td>
<td>_mm_cmpneq_ps</td>
</tr>
<tr>
<td></td>
<td>select_neq(F32vec4 A);</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R =</td>
<td>_mm_cmpneq_pd</td>
</tr>
<tr>
<td></td>
<td>select_neq(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_cmpneq_ss</td>
</tr>
<tr>
<td></td>
<td>select_neq(F32vec1 A);</td>
<td></td>
</tr>
</tbody>
</table>

### Compare for Less Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Region</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec4 R =</td>
<td>_mm_cmplt_ps</td>
</tr>
<tr>
<td></td>
<td>select_lt(F32vec4 A);</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R =</td>
<td>_mm_cmplt_pd</td>
</tr>
<tr>
<td></td>
<td>select_lt(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_cmplt_ss</td>
</tr>
<tr>
<td></td>
<td>select_lt(F32vec1 A);</td>
<td></td>
</tr>
</tbody>
</table>

### Compare for Less Than or Equal

<table>
<thead>
<tr>
<th>Type</th>
<th>Region</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec4 R =</td>
<td>_mm_cmple_ps</td>
</tr>
<tr>
<td></td>
<td>select_le(F32vec4 A);</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R =</td>
<td>_mm_cmple_pd</td>
</tr>
<tr>
<td></td>
<td>select_le(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_cmple_ps</td>
</tr>
<tr>
<td></td>
<td>select_le(F32vec1 A);</td>
<td></td>
</tr>
</tbody>
</table>

### Compare for Greater Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Region</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec4 R =</td>
<td>_mm_cmpgt_ps</td>
</tr>
<tr>
<td></td>
<td>select_gt(F32vec4 A);</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R =</td>
<td>_mm_cmpgt_pd</td>
</tr>
<tr>
<td></td>
<td>select_gt(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_cmpgt_ss</td>
</tr>
<tr>
<td></td>
<td>select_gt(F32vec1 A);</td>
<td></td>
</tr>
</tbody>
</table>

### Compare for Greater Than or Equal To

<table>
<thead>
<tr>
<th>Type</th>
<th>Region</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1 R =</td>
<td>_mm_cmpge_ps</td>
</tr>
<tr>
<td></td>
<td>select_ge(F32vec4 A);</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R =</td>
<td>_mm_cmpge_pd</td>
</tr>
<tr>
<td></td>
<td>select_ge(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R =</td>
<td>_mm_cmpge_ss</td>
</tr>
<tr>
<td></td>
<td>select_ge(F32vec1 A);</td>
<td></td>
</tr>
</tbody>
</table>
### Compare for Not Less Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1</td>
<td>R = select_nlt(F32vec4 A); mm_cmpnlt_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td>R = select_nlt(F64vec2 A); mm_cmpnlt_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td>R = select_nlt(F32vec1 A); mm_cmpnlt_ss</td>
</tr>
</tbody>
</table>

### Compare for Not Less or Equal

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1</td>
<td>R = select_nle(F32vec4 A); mm_cmpnle_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td>R = select_nle(F64vec2 A); mm_cmpnle_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td>R = select_nle(F32vec1 A); mm_cmpnle_ss</td>
</tr>
</tbody>
</table>

### Compare for Not Greater Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1</td>
<td>R = select_ngt(F32vec4 A); mm_cmpngt_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td>R = select_ngt(F64vec2 A); mm_cmpngt_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td>R = select_ngt(F32vec1 A); mm_cmpngt_ss</td>
</tr>
</tbody>
</table>

### Compare for Not Greater Than or Equal

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1</td>
<td>R = select_nge(F32vec4 A); mm_cmnpnge_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td>R = select_nge(F64vec2 A); mm_cmnpnge_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td>R = select_nge(F32vec1 A); mm_cmnpnge_ss</td>
</tr>
</tbody>
</table>

### Cacheability Support Operators

Stores (non-temporal) the two double-precision, floating-point values of A. Requires a 16-byte aligned address.

```c
void store_nta(double *p, F64vec2 A);
Corresponding intrinsic: _mm_stream_pd
```

Stores (non-temporal) the four single-precision, floating-point values of A. Requires a 16-byte aligned address.

```c
void store_nta(float *p, F32vec4 A);
Corresponding intrinsic: _mm_stream_ps
```
Debug Operations
The debug operations do not map to any compiler intrinsics for MMX™ technology or Intel® Streaming SIMD Extensions. They are provided for debugging programs only. Use of these operations may result in loss of performance, so you should not use them outside of debugging.

Output Operations
The two single, double-precision floating-point values of \( A \) are placed in the output buffer and printed in decimal format as follows:

\[
\text{cout} \ll \text{F64vec2} \ A; \\
"[1]:A1 [0]:A0"
\]

Corresponding intrinsics: none

The four, single-precision floating-point values of \( A \) are placed in the output buffer and printed in decimal format as follows:

\[
\text{cout} \ll \text{F32vec4} \ A; \\
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"
\]

Corresponding intrinsics: none

The lowest, single-precision floating-point value of \( A \) is placed in the output buffer and printed.

\[
\text{cout} \ll \text{F32vec1} \ A; \\
\]

Corresponding intrinsics: none

Element Access Operations
\[
double \ d = \text{F64vec2} \ A[\text{int} \ i]
\]
Read one of the two, double-precision floating-point values of \( A \) without modifying the corresponding floating-point value. Permitted values of \( i \) are 0 and 1. For example:

\[
double \ d = \text{F64vec2} \ A[1]; \\
\]
Corresponding intrinsics: none

Read one of the four, single-precision floating-point values of \( A \) without modifying the corresponding floating point value. Permitted values of \( i \) are 0, 1, 2, and 3. For example:

\[
float \ f = \text{F32vec4} \ A[\text{int} \ i]
\]
If DEBUG is enabled and \( i \) is not one of the permitted values (0-3), a diagnostic message is printed and the program aborts.

\[
float \ f = \text{F32vec4} \ A[2]; \\
\]
Corresponding intrinsics: none

Element Assignment Operations
\[
\text{F64vec4} \ A[\text{int} \ i] = \text{double} \ d;
\]
Modify one of the two, double-precision floating-point values of \( A \). Permitted values of \( \text{int} \ i \) are 0 and 1. For example:

\[
\text{F32vec4} \ A[1] = \text{double} \ d; \\
\text{F32vec4} \ A[\text{int} \ i] = \text{float} \ f;
\]
Modify one of the four, single-precision floating-point values of \( A \). Permitted values of \( \text{int } i \) are 0, 1, 2, and 3. For example:

If DEBUG is enabled and \( \text{int } i \) is not one of the permitted values (0-3), a diagnostic message is printed and the program aborts.

\[
\text{F32vec4 } A[3] = \text{float } f;
\]

Corresponding intrinsics: none.

**Load and Store Operators**

Loads two, double-precision floating-point values, copying them into the two, floating-point values of \( A \). No assumption is made for alignment.

\[
\text{void loadu(F64vec2 } A, \text{ double } *p) \\
\text{Corresponding intrinsic: } _\text{mm_loadu}_\text{pd}
\]

Stores the two, double-precision floating-point values of \( A \). No assumption is made for alignment.

\[
\text{void storeu(float } *p, \text{ F64vec2 } A); \\
\text{Corresponding intrinsic: } _\text{mm_storeu}_\text{pd}
\]

Loads four, single-precision floating-point values, copying them into the four floating-point values of \( A \). No assumption is made for alignment.

\[
\text{void loadu(F32vec4 } A, \text{ double } *p) \\
\text{Corresponding intrinsic: } _\text{mm_loadu}_\text{ps}
\]

Stores the four, single-precision floating-point values of \( A \). No assumption is made for alignment.

\[
\text{void storeu(float } *p, \text{ F32vec4 } A); \\
\text{Corresponding intrinsic: } _\text{mm_storeu}_\text{ps}
\]

**Unpack Operators**

Selects and interleaves the lower, double-precision floating-point values from \( A \) and \( B \).

\[
\text{F64vec2 } R = \text{unpack_low(F64vec2 } A, \text{ F64vec2 } B); \\
\text{Corresponding intrinsic: } _\text{mm_unpacklo}_\text{pd}(a, b)
\]

Selects and interleaves the higher, double-precision floating-point values from \( A \) and \( B \).

\[
\text{F64vec2 } R = \text{unpack_high(F64vec2 } A, \text{ F64vec2 } B); \\
\text{Corresponding intrinsic: } _\text{mm_unpackhi}_\text{pd}(a, b)
\]

Selects and interleaves the lower two, single-precision floating-point values from \( A \) and \( B \).

\[
\text{F32vec4 } R = \text{unpack_low(F32vec4 } A, \text{ F32vec4 } B); \\
\text{Corresponding intrinsic: } _\text{mm_unpacklo}_\text{ps}(a, b)
\]

Selects and interleaves the higher two, single-precision floating-point values from \( A \) and \( B \).

\[
\text{F32vec4 } R = \text{unpack_high(F32vec4 } A, \text{ F32vec4 } B); \\
\text{Corresponding intrinsic: } _\text{mm_unpackhi}_\text{ps}(a, b)
\]

**Move Mask Operators**

Creates a 2-bit mask from the most significant bits of the two, double-precision floating-point values of \( A \), as follows:

\[
\text{int } i = \text{move_mask(F64vec2 } A) \\
i := \text{sign}(a1)<<1 | \text{sign}(a0)<<0 \\
\text{Corresponding intrinsic: } _\text{mm_movemask}_\text{pd}
\]

Creates a 4-bit mask from the most significant bits of the four, single-precision floating-point values of \( A \), as follows:
int i = move_mask(F32vec4 A)
i := sign(a3)<<3 | sign(a2)<<2 | sign(a1)<<1 | sign(a0)<<0
Corresponding intrinsic: _mm_movemask_ps

Classes Quick Reference

This appendix contains tables listing operators to perform various SIMD operations, corresponding intrinsics to perform those operations, and the classes that implement those operations. The classes listed here belong to the Intel® C++ Class Libraries for SIMD Operations.

In the following tables,

- N/A indicates that the operator is not implemented in that particular class. For example, in the Logical Operations table, the Andnot operator is not implemented in the F32vec4 and F32vec1 classes.
- All other entries under Classes indicate that those operators are implemented in those particular classes, and the entries under the Classes columns provide the suffix for the corresponding intrinsic. For example, consider the Arithmetic Operations: Part1 table, where the corresponding intrinsic is _mm_add_[x] and the entry epi16 is under the I16vec8 column. It means that the I16vec8 class implements the addition operators and the corresponding intrinsic is _mm_add_epi16.

Logical Operations:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;, &amp;=</td>
<td><em>mm_and</em>[x]</td>
<td>si128</td>
</tr>
<tr>
<td></td>
<td>,</td>
<td>=</td>
</tr>
<tr>
<td>^, ^=</td>
<td><em>mm_xor</em>[x]</td>
<td>si128</td>
</tr>
<tr>
<td>Andnot</td>
<td><em>mm_andnot</em>[x]</td>
<td>si128</td>
</tr>
</tbody>
</table>

Arithmetic Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, +=</td>
<td><em>mm_add</em>[x]</td>
<td>epi64</td>
</tr>
<tr>
<td>-, -=</td>
<td><em>mm_sub</em>[x]</td>
<td>epi64</td>
</tr>
<tr>
<td>*</td>
<td>*=</td>
<td><em>mm_mullo</em>[x]</td>
</tr>
<tr>
<td>/</td>
<td>/=</td>
<td><em>mm_div</em>[x]</td>
</tr>
<tr>
<td>mul_high</td>
<td><em>mm_mulhi</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>mul_add</td>
<td><em>mm_madd</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>sqrt</td>
<td><em>mm_sqrt</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>rcp</td>
<td><em>mm_rcp</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>rcp_nr</td>
<td><em>mm_rcp</em>[x]</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I64vec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>rsqrt</td>
<td><em>mm_rsqrt</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>rsqrt_nr</td>
<td><em>mm_rsqrt</em>[x]</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Arithmetic Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I32vec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>+, +=</td>
<td><em>mm_add</em>[x]</td>
<td>pi32</td>
</tr>
<tr>
<td>-, -=</td>
<td><em>mm_sub</em>[x]</td>
<td>pi32</td>
</tr>
<tr>
<td>*, *=</td>
<td><em>mm_mullo</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>/, /=</td>
<td><em>mm_div</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>mul_high</td>
<td><em>mm_mulhi</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>mul_add</td>
<td><em>mm_madd</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>sqrt</td>
<td><em>mm_sqrt</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>rcp</td>
<td><em>mm_rcp</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>rcp_nr</td>
<td><em>mm_rcp</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>rsqrt</td>
<td><em>mm_rsqrt</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>rsqrt_nr</td>
<td><em>mm_rsqrt</em>[x]</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Shift Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I128vec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>&gt;&gt;, &gt;&gt;&gt;=</td>
<td><em>mm_srl</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td><em>mm_srl</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td><em>mm_sra</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td><em>mm_sra</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>&lt;&lt;, &lt;&lt;=</td>
<td><em>mm_sll</em>[x]</td>
<td>N/A</td>
</tr>
</tbody>
</table>

---

Compiler Reference
### Shift Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>mm_slli</em>[x]</td>
<td>I128vec1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>mm_srl</em>[x]</td>
<td>I64vec1</td>
</tr>
<tr>
<td></td>
<td><em>mm_slli</em>[x]</td>
<td>si64</td>
</tr>
<tr>
<td></td>
<td><em>mm_sra</em>[x]</td>
<td>si64</td>
</tr>
<tr>
<td></td>
<td><em>mm_srai</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td><em>mm_shl</em>[x]</td>
<td>epi64</td>
</tr>
<tr>
<td></td>
<td><em>mm_sshr</em>[x]</td>
<td>epi64</td>
</tr>
</tbody>
</table>

* Note that _mm_andnot_[y] intrinsics do not apply to the fvec classes.

### Comparison Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>mm_cmpeq</em>[x]</td>
<td>I32vec4</td>
</tr>
<tr>
<td></td>
<td><em>mm_cmpeq</em>[x]</td>
<td>epi32</td>
</tr>
<tr>
<td></td>
<td><em>mm_cmpeq</em>[x]</td>
<td>epi32</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>si128</td>
</tr>
<tr>
<td></td>
<td><em>mm_cmpgt</em>[x]</td>
<td>epi32</td>
</tr>
<tr>
<td></td>
<td><em>mm_cmple</em>[x]</td>
<td>epi32</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>si128</td>
</tr>
<tr>
<td></td>
<td><em>mm_cmpngt</em>[x]</td>
<td>epi32</td>
</tr>
<tr>
<td></td>
<td>_mm_cmpnge</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>_mm_cmplt</td>
<td>epi32</td>
</tr>
<tr>
<td></td>
<td>_mm_cmple</td>
<td>epi32</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>si128</td>
</tr>
<tr>
<td></td>
<td>_mm_cmpnlt</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>_mm_cmpnle</td>
<td>N/A</td>
</tr>
</tbody>
</table>

* Note that _mm_andnot_[y] intrinsics do not apply to the fvec classes.

### Comparison Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>mm_cmpeq</em>[x]</td>
<td>F64vec2</td>
</tr>
<tr>
<td></td>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd</td>
</tr>
</tbody>
</table>
Operators | Corresponding Intrinsic | Classes
--- | --- | ---
| | _mm_andnot_[y]* | F64vec2 F32vec4 F32vec1
| cmplt | _mm_cmplt_[x] | pd ps ss
| cmple | _mm_cmple_[x] | pd ps ss
| cmpngt | _mm_cmpngt_[x] | pd ps ss
| cmpnle | _mm_cmplnle_[x] | pd ps ss

* Note that _mm_andnot_[y] intrinsics do not apply to the fvec classes.

Conditional Select Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_eq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>I32vec I16vec I8vec 4 8 6 2 4 8</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>epi32 epi16 epi8 pi32 pi16 pi8</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td>select_neq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>I32vec I16vec I8vec 4 8 6 2 4 8</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>epi32 epi16 epi8 pi32 pi16 pi8</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td>select_gt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>I32vec I16vec I8vec 4 8 6 2 4 8</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>epi32 epi16 epi8 pi32 pi16 pi8</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td>select_ge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>I32vec I16vec I8vec 4 8 6 2 4 8</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>epi32 epi16 epi8 pi32 pi16 pi8</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td>select_lt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>I32vec I16vec I8vec 4 8 6 2 4 8</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>epi32 epi16 epi8 pi32 pi16 pi8</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td>select_le</td>
<td><em>mm_cmple</em>[x]</td>
<td>I32vec I16vec I8vec 4 8 6 2 4 8</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>epi32 epi16 epi8 pi32 pi16 pi8</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>s128 s128 s128 s128 s128 s128</td>
</tr>
<tr>
<td>Operators</td>
<td>Corresponding Intrinsic</td>
<td>Classes</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><em>mm_or</em>[y]</td>
<td>si128</td>
<td>si128</td>
</tr>
<tr>
<td><em>mm_cmpgt</em>[x]</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><em>mm_cmpge</em>[x]</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><em>mm_cmplt</em>[x]</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><em>mm_cmple</em>[x]</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

* Note that _mm_andnot_[y] intrinsics do not apply to the fvec classes.

Conditional Select Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd</td>
<td>ps</td>
</tr>
<tr>
<td><em>mm_and</em>[y]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_or</em>[y]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd</td>
<td>ps</td>
</tr>
<tr>
<td><em>mm_and</em>[y]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_or</em>[y]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_cmpgt</em>[x]</td>
<td>pd</td>
<td>ps</td>
</tr>
<tr>
<td><em>mm_and</em>[y]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_or</em>[y]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_cmplt</em>[x]</td>
<td>pd</td>
<td>ps</td>
</tr>
<tr>
<td><em>mm_and</em>[y]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_or</em>[y]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_cmple</em>[x]</td>
<td>pd</td>
<td>ps</td>
</tr>
<tr>
<td><em>mm_and</em>[y]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm_or</em>[y]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Note that _mm_andnot_[y] intrinsics do not apply to the fvec classes.
### Packing and Unpacking Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I64vec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>unpack_high</td>
<td><em>mm_unpackhi</em>[x]</td>
<td>epi64</td>
</tr>
<tr>
<td>unpack_low</td>
<td><em>mm_unpacklo</em>[x]</td>
<td>epi64</td>
</tr>
<tr>
<td>pack_sat</td>
<td><em>mm_packs</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>packu_sat</td>
<td><em>mm_packus</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>sat_add</td>
<td><em>mm_adds</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>sat_sub</td>
<td><em>mm_subs</em>[x]</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Packing and Unpacking Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I16vec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>unpack_high</td>
<td><em>mm_unpackhi</em>[x]</td>
<td>pi16</td>
</tr>
<tr>
<td>unpack_low</td>
<td><em>mm_unpacklo</em>[x]</td>
<td>pi16</td>
</tr>
<tr>
<td>pack_sat</td>
<td><em>mm_packs</em>[x]</td>
<td>pi16</td>
</tr>
<tr>
<td>packu_sat</td>
<td><em>mm_packus</em>[x]</td>
<td>pu16</td>
</tr>
<tr>
<td>sat_add</td>
<td><em>mm_adds</em>[x]</td>
<td>pi16</td>
</tr>
<tr>
<td>sat_sub</td>
<td><em>mm_subs</em>[x]</td>
<td>pi16</td>
</tr>
</tbody>
</table>

### Conversions Operations:

Conversion operations can be performed using intrinsics only. There are no classes implemented to correspond to these intrinsics.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>F64vec2ToInt</td>
<td>_mm_cvttsd_si32</td>
</tr>
<tr>
<td>F32vec4ToF64vec2</td>
<td>_mm_cvtps_pd</td>
</tr>
<tr>
<td>F64vec2ToF32vec4</td>
<td>_mm_cvtpd_ps</td>
</tr>
<tr>
<td>IntToF64vec2</td>
<td>_mm_cvtsi32_sd</td>
</tr>
<tr>
<td>F32vec4ToInt</td>
<td>_mm_cvtt_ss2si</td>
</tr>
<tr>
<td>F32vec4ToIs32vec2</td>
<td>_mm_cvtpsi32</td>
</tr>
<tr>
<td>IntToF32vec4</td>
<td>_mm_cvtsi32_ss</td>
</tr>
<tr>
<td>Is32vec2ToF32vec4</td>
<td>_mm_cvtpi32_ps</td>
</tr>
</tbody>
</table>
Programming Example

This sample program uses the F32vec4 class to average the elements of a twenty element floating point array.

```c
#include <fvec.h>

#define SHUFFLE(a,b,i) (F32vec4)_mm_shuffle_ps(a,b,i)
#include <stdio.h>
#define SIZE 20

float result;
_MM_ALIGNMENT float array[SIZE];

void Add20ArrayElements (F32vec4 *array, float *result) {
    F32vec4 vec0, vec1;
    vec0 = _mm_load_ps ((float *) array); // Load array's first four floats

    vec0 += array[1]; // Add elements 5-8
    vec0 += array[2]; // Add elements 9-12
    vec0 += array[3]; // Add elements 13-16
    vec0 += array[4]; // Add elements 17-20

    vec1 = SHUFFLE(vec1, vec0, 0x40);
    vec0 += vec1;
    vec1 = SHUFFLE(vec1, vec0, 0x30);
    vec0 += vec1;
    vec0 = SHUFFLE(vec0, vec0, 2);
    _mm_store_ss (result, vec0); // Store the final sum
}

void main(int argc, char *argv[]) {
    int i;

    for (i=0; i < SIZE; i++) { array[i] = (float) i; }

    Add20ArrayElements (array, &result);
}```
C++ Library Extensions

This section contains descriptions of Intel's C++ library extensions that assist users in parallel programming. The following C++ library specialization is included:

- **Introduction to Intels valarray Implementation**: Enables users to leverage a custom valarray header file that uses the Intel® Integrated Performance Primitives (Intel® IPP) for performance benefit.

Intel's valarray Implementation

The Intel® oneAPI DPC++/C++ Compiler provides a high performance implementation of specialized one-dimensional valarray operations for the C++ standard STL valarray container.

The standard C++ valarray template consists of array/vector operations for high performance computing. These operations are designed to exploit high performance hardware features such as parallelism and achieve performance benefits.

Intel's valarray implementation uses the Intel® Integrated Performance Primitives (Intel® IPP), which is part of the product. Select IPP when you install the product.

The valarray implementation consists of a replacement header, `<valarray>`, that provides a specialized, high-performance implementation for the following operators and types:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Valarrays of Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs, acos, acosh, asin, asinh, atan, atan2, atanh, cbrt, cdfnorm, ceil, cos, cosh, erf, erfc, erfinv, exp, expm1, floor, hypot, inv, invcbrt, invsqrt, ln, log, log10, log1p, nearbyint, pow, pow2o3, pow3o2, powx, rint, round, sin, sinh, sqrt, tan, tanh, trunk</td>
<td>float, double</td>
</tr>
<tr>
<td>add, conj, div, mul, mulbyconj, mul, sub</td>
<td>Ipp32fc, Ipp64fc</td>
</tr>
<tr>
<td>addition, subtraction, division, multiplication</td>
<td>float, double</td>
</tr>
<tr>
<td>bitwise or, and, xor</td>
<td>(all unsigned) char, short, int</td>
</tr>
<tr>
<td>min, max, sum</td>
<td>signed or short/signed int, float, double</td>
</tr>
</tbody>
</table>

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex).

Notice revision #20201201

Using Intel's valarray Implementation

Intel's valarray implementation allows you to declare huge arrays for parallel processing. Improved implementation of valarray is tied up with calling the IPP libraries that are part of Intel® Integrated Performance Primitives (Intel® IPP).

This content is specific to C++; it does not apply to DPC++.
Using valarray in Source Code

To use valarrays in your source code, include the valarray header file, `<valarray>`. The `<valarray>` header file is located in the path `<installdir>/perf_header`.

The example code below shows a valarray addition operation (+) specialized through use of Intel’s implementation of valarray:

```cpp
#include <valarray>

void test()
{
    std::valarray<float> vi(N), va(N);
    ...
    vi = vi + va;  //array addition
    ...
}
```

**NOTE**

To use the static merged library containing all CPU-specific optimized versions of the library code, you need to call the ippStaticInit function first, before any IPP calls. This ensures automatic dispatch to the appropriate version of the library code for Intel® processor and the generic version of the library code for non-Intel processors at runtime. If you do not call ippStaticInit first, the merged library will use the generic instance of the code. If you are using the dynamic version of the libraries, you do not need to call ippStaticInit.

Compiling valarray Source Code

To compile your valarray source code, the compiler option, `/Quse-intel-optimized-headers` (for Windows*) or `-use-intel-optimized-headers` (for Linux*), is used to include the required valarray header file and all the necessary IPP library files.

The following examples illustrate how to compile and link a program to include the Intel valarray replacement header file and link with the Intel® IPP libraries. Refer to the Intel® IPP documentation for details.

In the following examples, "merged" libraries refers to using a static library that contains all the CPU-specific variants of the library code.

**Windows* OS examples:**

The following command line performs a one-step compilation for a system based on IA-32 architecture, running Windows OS:

```bash
icx /Quse-intel-optimized-headers source.cpp
```

The following command lines perform separate compile and link steps for a system based on IA-32 architecture, running Windows OS:

**DLL (dynamic):**

```bash
icx /Quse-intel-optimized-headers /c source.cpp
icx source.obj /Quse-intel-optimized-headers
```

**Merged (static):**

```bash
icx /Quse-intel-optimized-headers /Qipp-link:static /c source.cpp
icx source.obj /Quse-intel-optimized-headers /Qipp-link:static
```
Linux* OS examples:
The following command line performs a one-step compilation for a system based on Intel® 64 architecture, running Linux OS:

```
icpx -use-intel-optimized-headers source.cpp
```

The following command lines perform separate compile and link steps for a system based on Intel® 64 architecture, running Linux OS:

so (dynamic):

```
icpx -use-intel-optimized-headers -c source.cpp
icpx source.o -use-intel-optimized-headers -shared-intel
```

Merged (static):

```
icpx -use-intel-optimized-headers -c source.cpp
icpx source.o -use-intel-optimized-headers
```

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

Intel's C++ Asynchronous I/O Extensions for Windows* Operating Systems

This topic only applies to Windows* OS.

Intel's C/C++ asynchronous input/output (Intel's C/C++ AIO) extensions, like library functions or classes, can be used to improve the performance of C/C++ applications by executing I/O operations in asynchronous mode. The extensions initiate I/O operation and immediately resume normal tasks while the I/O operations are executed in parallel.

Intel's C/C++ asynchronous I/O extensions are supported on IA-32 architecture-based (C++ only; does not pertain to DPC++) and Intel® 64 architecture-based Windows platforms.

Intel's C/C++ AIO library functions and template class are implemented in the libicaio.lib library. This library is supplied as part of the Intel® oneAPI DPC++/C++ Compiler package and is installed into the common directory: <install-dir>/lib.

Types of Intel's C/C++ Asynchronous I/O Extensions

Intel's C/C++ asynchronous I/O extensions comprise the following:

- **Asynchronous I/O Library**: A set of POSIX*-based asynchronous I/O library functions, supported on Windows operating systems, for applications written in C/C++ language. The interface file is aio.h.
- **Asynchronous I/O Template Class**: An asych_class template class, supported on Windows* operating systems, for applications written in C++ language. This template class can be used to introduce asynchronous execution of I/O operations with the Standard Template Library's (STL's) streams classes. The interface file is aiostream.h.

See Also

Intel's C++ Asynchronous I/O Library for Windows* OS
Intel's C++ Asynchronous I/O Class for Windows* OS
Intel's C++ Asynchronous I/O Library for Windows* Operating Systems

This topic only applies to Windows* OS.

Intel's C++ asynchronous I/O (AIO) library implementation for the Windows operating system (on IA-32 (C++ only; does not pertain to DPC++) and Intel® 64 platforms) is similar to the POSIX* AIO library implementation for the Linux* operating system.

The differences between Intel's C/C++ AIO Windows OS implementation and the standard POSIX AIO implementation are listed below:

- In `struct aiocb`,
  - The Windows OS compatible type `HANDLE` replaces the POSIX AIO type `unsigned int` for the file descriptor `aio_fildes`.
  - The type `intptr_t` replaces the POSIX AIO types `ssize_t` and `__off_t`.
  - The structure specifying the signal event descriptor, `struct sigevent` is similar to the Linux* operating system implementation of the POSIX AIO library. It differs from the Linux* implementation in the following ways:
    - Signal notification and non-notification for thread call-back is supported
    - Signal notification on completion of the AIO operation is not supported

This is true for programs that were already written for Linux/Unix and ported to Windows OS that wish to setup an AIO completion handler without the name of the handler set in the `aiocb struct`. Because of the way that signals are supported in Windows, this is impossible to implement. For new applications, or to port existing applications, the programmer should set the name of the handler before calling the `aio_read` or `aio_write` routines. For example:

```c
static void aio_CompletionRoutine(sigval_t sigval)
{
    // ... code ...
}
...

my_aio.aio_sigevent.sigev_notify          = SIGEV_THREAD;
my_aio.aio_sigevent.sigev_notify_function = aio_CompletionRoutine;
```

**NOTE**
The POSIX AIO library and the Microsoft* SDK provide similar AIO functions. The main difference between the POSIX AIO functions and the Windows operating system-based AIO functions is that while POSIX allows you to execute AIO operations with any file, the Windows operating system executes AIO operations only with files flagged with `FILE_FLAG_OVERLAPPED`.

Intel's asynchronous I/O library functions listed below are all based on POSIX AIO functions. They are defined in the `aio.h` file.

- `aio_read`
- `aio_write`
- `aio_suspend`
- `aio_error`
- `aio_return`
- `aio_fsync`
- `aio_cancel`
- `lio_listio`

**aio_read**

Performs an asynchronous read operation.
**Syntax**

```c
int aio_read(struct aiocb *aiocbp);
```

**Description**

The `aio_read()` function requests an asynchronous read operation, calling the function,

```c
"ReadFile(hFile, lpBuffer, nNumberOfBytesToRead, lpNumberOfBytesRead, NULL);
```

where,

- `hFile` is given by `aiocbp->aio_fildes`
- `lpBuffer` is given by `aiocbp->aio_buf`
- `nNumberOfBytesToRead` is given by `aiocbp->aio_nbytes`

Use the function `aio_return()` to retrieve the actual bytes read in `lpNumberOfBytesRead`.

Use the extension `aiocb->aio_offset == (intptr_t)-1` to start the read operation after the last read record. This extension avoids extra file positioning and enhances performance.

**Returns**

-0: On success
-1: On error

To get the correct error code, use `errno`. To get the error that occurred during asynchronous read operation, use `aio_error()` function.

**See Also**

Example Code for `aio_read()`

**aio_write**

*Performs an asynchronous write operation.*

**Syntax**

```c
int aio_write(struct aiocb *aiocbp);
```

**Description**

The `aio_write()` function requests an asynchronous write operation, calling the function,

```c
"WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, lpNumberOfBytesWritten, NULL);
```

where,

- `hFile` is given by `aiocbp->aio_fildes`
- `lpBuffer` is given by `aiocbp->aio_buf`
- `nNumberOfBytesToWrite` is given by `aiocbp->aio_nbytes`

Use the function `aio_return()` to retrieve the actual bytes written in `lpNumberOfBytesWritten`.

Use the extension `aiocb->aio_offset == (intptr_t)-1` to start the write operation after the last written record. This extension avoids extra file positioning and enhances performance.

**Returns**

-0: On success
-1: On error

To get the correct error code, use `errno`. To get the error that occurred during asynchronous write operation, use `aio_error()` function.
See Also
Example Code for aio_write()

Example for aio_read and aio_write Functions

The example illustrates the performance gain of the asynchronous I/O usage in comparison with synchronous I/O usage. In the example, 5.6 MB of data is asynchronously written with the main program computation, which is the scalar multiplication of two vectors with some normalization.

C-source file executing a scalar multiplication:

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

double do_compute(double A, double B, int arr_len)
{
    int i;
    double res = 0;
    double *xA = malloc(arr_len * sizeof(double));
    double *xB = malloc(arr_len * sizeof(double));
    if ( !xA || !xB )
        abort();
    for (i = 0; i < arr_len; i++) {
        xA[i] = sin(A);
        xB[i] = cos(B);
        res = res + xA[i]*xA[i];
    }
    free(xA);
    free(xB);
    return res;
}
```

C-main-source file using asynchronous I/O implementation:

```c
#define DIM_X   123
#define DIM_Y   70000
double  aio_dat[DIM_Y /*12MB*/] = {0};
double  aio_dat_tmp[DIM_Y /*12MB*/];
#include <stdio.h>
#include <aio.h>
typedef struct aiocb  aiocb_t;
aiocb_t   my_aio;
aiocb_t  *my_aio_list[1] = {&my_aio};

int main()
{
    double do_compute(double A, double B, int arr_len);
    int i, j;
    HANDLE fd = CreateFile("aio.dat",
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ,
        NULL,
        OPEN_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    /* Do some complex computation */
    for (i = 0; i < DIM_X; i++) {
        /* Do some complex computation */
    }
    close(fd);
}
```
for (j = 0; j < DIM_Y; j++)
daio_dat[j] = do_compute(i, j, DIM_X);

if (i) aio_suspend(my_aio_list, 1, 0);
my_aio.aio_fildes = fd;
my_aio.aio_buf = memcpy(aio_dat_tmp, aio_dat, sizeof(aio_dat_tmp));
my_aio.aio_nbytes = sizeof(aio_dat_tmp);
my_aio.aio_offset = (intptr_t)-1;
my_aio.aio_sigevent.sigev_notify = SIGEV_NONE;

if (aio_write((void*)&my_aio) == -1) {
    printf("ERROR!!! %s\n", "aio_write()==-1");
    abort();
}
aio_suspend(my_aio_list, 1, 0);
return 0;
}

C-main-source file example 2 using asynchronous I/O implementation:

```c
#define DIM_X   123
#define DIM_Y    70
double  aio_dat[DIM_Y] = {0};
double  aio_dat_tmp[DIM_Y];
static volatile int aio_flg = 1;
#include <aio.h>
typedef struct aiocb aiocb_t;
aiocb_t               my_aio;
#define WAIT { while (!aio_flg); aio_flg = 0; }
#define aio_OPEN(_fname)\nCreateFile(_fname,\n    GENERIC_READ | GENERIC_WRITE,\n    FILE_SHARE_READ,\n    NULL,\n    OPEN_ALWAYS,\n    FILE_ATTRIBUTE_NORMAL,\n    NULL)

static void aio_CompletionRoutine(sigval_t sigval)
{
    aio_flg = 1;
}

int main()
{
    double do_compute(double A, double B, int arr_len);
    int     i, j, res;
    char   *fname = "aio_sample2.dat";
    HANDLE  aio_fildes = aio_OPEN(fname);

    my_aio.aio_fildes = aio_fildes;
    my_aio.aio_nbytes = sizeof(aio_dat_tmp);
    my_aio.aio_sigevent.sigev_notify = SIGEV_NONE;
    my_aio.aio_sigevent.sigev_notify_function = aio_CompletionRoutine;
```
/
** writing
*/
my_aio.aio_offset = -1;
printf("Writing\n");
for (i = 0; i < DIM_X; i++) {
    for (j = 0; j < DIM_Y; j++)
        aio_dat[j] = do_compute(i, j, DIM_X);
    WAIT;
    my_aio.aio_buf = memcpy(aio_dat_tmp, aio_dat, sizeof(aio_dat_tmp));
    res = aio_write(&my_aio);
    if (res) {printf("res!=0\n");abort();}
}
// flushing
//
printf("Flushing\n");
WAIT;
res = aio_fsync(O_SYNC, &my_aio);
if (res) {printf("res!=0\n");abort();}
WAIT;
// reading
//
printf("Reading\n");
my_aio.aio_buf = (volatile char*)aio_dat_tmp;
for (i = 0; i < DIM_X; i++) {
    aio_read(&my_aio);
    for (j = 0; j < DIM_Y; j++)
        aio_dat[j] = do_compute(i, j, DIM_X);
    WAIT;
    res = aio_return(&my_aio);
    if (res != sizeof(aio_dat)) {
        printf("aio_read() did read %d bytes, expecting %d bytes\n", res, sizeof(aio_dat));
    }
    for (j = 0; j < DIM_Y; j++)
        if (aio_dat[j] != aio_dat_tmp[j])
            {printf("ERROR: aio_dat[j] != aio_dat_tmp[j]\n I=%d J=%d\n", i, j); abort();}
    my_aio.aio_offset += my_aio.aio_nbytes;
}
CloseHandle(aio_fildes);
printf("\nDone\n");
return 0;
}

See Also
aio_read()
aio_write()
**aio_suspend**

Suspends the calling process until one of the asynchronous I/O operations completes.

**Syntax**

```c
int aio_suspend(const struct aiocb * const cblist[], int n, const struct timespec *timeout);
```

**Arguments**

- `cblist[]` Pointer to a control block on which I/O is initiated
- `n` Length of `cblist` list
- `*timeout` Time interval to suspend the calling process

**Description**

The `aio_suspend()` function is like a wait operation. It suspends the calling process until,

- At least one of the asynchronous I/O requests in the list `cblist` of length `n` has completed
- A signal is delivered
- The time interval indicated in `timeout` is not `NULL` and has passed.

Each item in the `cblist` list must either be `NULL` (when it is ignored), or a pointer to a control block on which I/O was initiated using `aio_read()`, `aio_write()`, or `lio_listio()` functions.

**Returns**

- **0**: On success
- **-1**: On error

To get the correct error code, use `errno`.

**See Also**

Example Code for `aio_suspend()`

**Example for aio_suspend Function**

The following example illustrates a wait operation execution using the `aio_suspend()` function.

```c
int aio_ex_2(HANDLE fd)
{
    static struct aiocb   aio[2];
    static struct aiocb  *aio_list[2] = {&aio[0], &aio[1]};
    int i, ret;

    /* Data initialization */
    IC_AIO_DATA_INIT(aio[0], fd, "rec#1\n", strlen("rec#1\n"), 0)
    IC_AIO_DATA_INIT(aio[1], fd, "rec#2\n", strlen("rec#2\n"), aio[0].aio_nbytes)

    /* Asynch-write */
    if (aio_write(&aio[0]) == -1) return errno;
    if (aio_write(&aio[1]) == -1) return errno;

    /* Do some complex computation */
    printf("do_compute(1000, 1.123)=%.f", do_compute(1000, 1.123));
    /* do the wait operation using sleep() */

    return 0;
}
```
ret = aio_suspend(aio_list, 2, 0);
if (ret == -1) return errno;
return 0;
} /* aio_ex_2 */

Result upon execution:

-bash-3.00$ ./a.out
-bash-3.00$ cat dat
rec#1
rec#2

Remarks:

1. In the example, the IC_AIO_DATA_INIT is defined as follows:

```c
#define IC_AIO_DATA_INIT(_aio, _fd, _dat, _len, _off)\
{memset(&_aio, 0, sizeof(_aio)); \\(_aio.aio_fildes = _fd; \\(_aio.aio_buf = _dat; \\(_aio.aio_nbytes = _len; \\(_aio.aio_offset = _off;)
```

2. The file descriptor fd is obtained as:

```c
HANDLE fd = CreateFile("dat",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ,
    NULL,
    OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL/*|FILE_FLAG_OVERLAPPED*/,
    NULL);
```

See Also

aio_suspend()

**aio_error**

*Returns error status for asynchronous I/O requests.*

**Syntax**

```c
int aio_error(const struct aiocb *aiocbp);
```

**Arguments**

*aiocbp*  
Pointer to control block from where asynchronous I/O request is generated

**Description**

The aio_error() function returns the error status for the asynchronous I/O request in the control block, which is pointed to by aiocbp.

**Returns**

**EINPROGRESS:** When asynchronous I/O request is not completed

**ECANCELED:** When asynchronous I/O request is cancelled

0: On success

**Error value:** On error
To get the correct error value/code, use `errno`. This is the same error value returned when an error occurs during a `ReadFile()`, `WriteFile()`, or a `FlushFileBuffers()` operation.

**See Also**

Example Code for `aio_error()`

**aio_return**

*Returns the final return status for the asynchronous I/O request.*

**Syntax**

```c
ssize_t aio_return(struct aiocb *aiocbp);
```

**Arguments**

*`*aiocbp`*

Pointer to control block from where asynchronous I/O request is generated

**Description**

The `aio_return` function returns the final return status for the asynchronous I/O request with control block pointed to by `aiocbp`.

Call this function only once for any given request, after `aio_error()` returns a value other than `EINPROGRESS`.

**Returns**

**Return value for synchronous ReadFile()/WriteFile()/FlushFileBuffer() requests:** When asynchronous I/O operation is completed

**Undefined return value:** When asynchronous I/O operation is not completed

**Error value:** When an error occurs

To get the correct error code/value, use `errno`.

**See Also**

Example Code for `aio_return()`

**Example for aio_error and aio_return Functions**

The following example illustrates how the `aio_error()` and `aio_return()` functions can be used.

```c
int aio_ex_3(HANDLE fd)
{
    static struct aiocb aio;
    static struct aiocb *aio_list[] = {&aio};
    int    ret;
    char  *dat = "Hello from Ex-3\n";

    /* Data initialization and asynchronously writing */
    IC_AIO_DATA_INIT(aio, fd, dat, strlen(dat), 0);
    if (aio_write(& aio) == -1) return errno;

    ret = aio_error(&aio);
    if ( ret == EINPROGRESS ) {
        fprintf(stderr, "ERRNO=%d STR=%s\n", ret, strerror(ret));
    }
}
```
ret = aio_suspend(aio_list, 1, NULL);
if (ret == -1) return errno;
else if (ret)
    return ret;

ret = aio_error(&aio);
if (ret) return ret;

ret = aio_return(&aio);
printf("ret=%d\n", ret);
return 0;

}/* aio_ex_3 */

Result upon execution:
-bash-3.00$ ./a.out
ERRNO=115 STR=Operation now in progress
ret=16
-bash-3.00$ cat dat
Hello from Ex-3

Remarks:
1. In the example, the IC_AIO_DATA_INIT is defined as follows:

```c
#define IC_AIO_DATA_INIT(_aio, _fd, _dat, _len, _off)
    {memset(&_aio, 0, sizeof(_aio));
     _aio.aio_fildes = _fd;
     _aio.aio_buf    = _dat;
     _aio.aio_nbytes = _len;
     _aio.aio_offset = _off;}
```

2. The file descriptor fd is obtained as:

```c
HANDLE fd = CreateFile("dat",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ,
    NULL,
    OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL/*|FILE_FLAG_OVERLAPPED*/,
    NULL);
```

See Also
aio_error()
aio_return()
aio_fsync

**aio_fsync**

_Synchronizes all outstanding asynchronous I/O operations._

_Syntax_:

```c
int aio_fsync(int op, struct aiocb *aiocbp);
```

_Arguments_

- **op** Type of synchronization request operation
Description

The aio_fsync() function performs a synchronization request operation on all outstanding asynchronous I/O operations associated with aiocbp->aio_fildes.

Returns

0: On successfully performing a synchronization request.
-1: On error; to get the correct error code, use errno.

aio_cancel

Cancels outstanding asynchronous I/O requests for the file descriptor fd.

Syntax

int aio_cancel(HANDLE fd, struct aiocb *aiocbp);

Arguments

fd
File descriptor

*aiocbp
Pointer to control block from where asynchronous I/O request is generated

Description

The aio_cancel() function cancels outstanding asynchronous I/O requests for the file descriptor fd. If aiocbp is NULL, all outstanding asynchronous I/O requests are cancelled. If aiocbp is not NULL, only the requests described by the control block pointed to by aiocbp are cancelled.

Normal asynchronous notification occurs for cancelled requests. The request return status is set to -1, and the request error status is set to ECANCELED. The control block of requests that cannot be cancelled is not changed.

Unspecified results occur if aiocbp is not NULL and the fd differs from the file descriptor with which the asynchronous operation was initiated.

Returns

AIO_CANCELLED: When all specified requests are cancelled successfully.
AIO_NOTCANCELLED: When at least one of the specified requests is still in process of being cancelled; check the status of request using aio_error.
AIO_ALLDONE: When all specified requests were completed before cancel call was placed.
-1: When some error occurs. To get the correct error code, use errno.

See Also

Example Code for aio_cancel()

Example for aio_cancel Function

The following example illustrates how aio_cancel() function can be used.

```c
int aio_ex_4(HANDLE fd)
{
    static struct aiocb aio;
```
static struct aiocb *aio_list[] = {&aio};
int ret;
char *dat = "Hello from Ex-4"

printf("AIO_CANCELED=%d AIO_NOTCANCELED=%d\n", AIO_CANCELED, AIO_NOTCANCELED);

/* Data initialization and asynchronously writing */
IC_AIO_DATA_INIT(aio, fd, dat, strlen(dat), 0);
if (aio_write(&aio) == -1) return errno;

ret = aio_cancel(fd, &aio);
if ( ret == AIO_NOTCANCELED ) {
    fprintf(stderr, "ERRNO=%d STR=%s\n", ret, strerror(ret));
    ret = aio_suspend(aio_list, 1, NULL);
    if (ret == -1) return errno;
}

ret = aio_cancel(fd, &aio);
if ( ret == AIO_CANCELED )
    fprintf(stderr, "ERRNO=%d STR=%s\n", ret, strerror(ret));
else if (ret) return ret;

return 0;
}/* aio_ex_4 */

Result upon execution:
-bash-3.00$ ./a.out
AIO_CANCELED=0 AIO_NOTCANCELED=1
ERRNO=1 STR=Operation not permitted
-bash-3.00$ cat dat
Hello from Ex-4
-bash-3.00$

Remarks:
1. In the example, the IC_AIO_DATA_INIT is defined as follows:

```c
#define IC_AIO_DATA_INIT(_aio, _fd, _dat, _len, _off)\
{\n    memset(&_aio, 0, sizeof(_aio)); \n    _aio.aio_fildes = _fd; \n    _aio.aio_buf = _dat; \n    _aio.aio_nbytes = _len; \n    _aio.aio_offset = _off;}\
```

2. The file descriptor fd is obtained as:

```c
HANDLE fd = CreateFile("dat",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ,
    NULL,
    OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL/*|FILE_FLAG_OVERLAPPED*/,
    NULL);
```

See Also
aio_cancel()
**lio_listio**

*Performs an asynchronous read operation.*

**Syntax**

```c
int lio_listio(int mode, struct aiocb *list[], int nent, struct sigevent *sig);
```

**Arguments**

- `mode` Takes following values declared in `<aio.h>` file:
  - `LIO_WAIT`: Use when you want the function to return only after completing I/O operations (synchronous I/O operations)
  - `LIO_NOWAIT`: Use when you want the function to return as soon as I/O operations are queued (asynchronous I/O requests)

- `*list[]` Array of the `aiocb` pointers specifying the submitted I/O requests; NULL elements in the array are ignored

- `nent` Number of elements in the array

- `*sig` Determines if asynchronous notification is sent after all I/O operations completes; takes following values:
  - 0: Asynchronous notification occurs; a queued signal, with an application-defined value, is generated when an asynchronous I/O request occurs
  - 1: Asynchronous notification does not occur even when asynchronous I/O requests are processed
  - 2: Asynchronous notification occurs; a notification function is called to perform notification

**Description**

The `lio_listio()` function initiates a list of I/O requests with a single function call.

The `mode` argument determines whether the function returns when all the I/O operations are completed, or as soon as the operations are queued.

If the `mode` argument is `LIO_WAIT`, the function waits until all I/O operations are complete. The `sig` argument is ignored in this case.

If the `mode` argument is `LIO_NOWAIT`, the function returns immediately. Asynchronous notification occurs according to the `sig` argument after all the I/O operations complete.

**Returns**

When `mode`=LIO_NOWAIT the `lio_listio()` function returns:

- **0**: I/O operations are successfully queued
- **-1**: Error; I/O operations not queued; to get the proper error code, use `errno`.

When `mode`=LIO_WAIT the `lio_listio()` function returns:

- **0**: I/O operations specified completed successfully
- **-1**: Error; I/O operations not completed; to get the proper error code, use `errno`.
Example for lio_listio Function

The following example illustrates how the lio_listio() function can be used.

```c
int aio_ex_5(HANDLE fd)
{
    static struct aiocb aio[2];
    static struct aiocb *aio_list[2] = {&aio[0], &aio[1]};
    int i, ret;

    /*
    ** Data initialization and Synchronously writing
    */
    IC_AIO_DATA_INIT(aio[0], fd, "rec#1\n", strlen("rec#1\n"), 0)
    IC_AIO_DATA_INIT(aio[1], fd, "rec#2\n", strlen("rec#2\n"),
                      aio[0].aio_nbytes)
    aio[0].aio_lio_opcode = aio[1].aio_lio_opcode = LIO_WRITE;
    ret = lio_listio(LIO_WAIT, aio_list, 2, 0);
    if (ret) return ret;

    return 0;
} /* aio_ex_5 */
```

Result upon execution:

```
-bash-3.00$ ./a.out
-bash-3.00$ cat dat
rec#1
rec#2
-bash-3.00$
```

Remarks:

1. In the example, the IC_AIO_DATA_INIT is defined as follows:

```c
#define IC_AIO_DATA_INIT(_aio, _fd, _dat, _len, _off)\n    {memset(&_aio, 0, sizeof(_aio)); \n    _aio.aio_fildes = _fd; \n    _aio.aio_buf = _dat; \n    _aio.aio_nbytes = _len; \n    _aio.aio_offset = _off;}
```

2. The file descriptor `fd` is obtained as:

```c
HANDLE fd = CreateFile("dat",\n    GENERIC_READ | GENERIC_WRITE,\n    FILE_SHARE_READ,\n    NULL,\n    OPEN_ALWAYS,\n    FILE_ATTRIBUTE_NORMAL/*|FILE_FLAG_OVERLAPPED*/,\n    NULL);
```

3. The aio_lio_opcode refers to the field of each aiocb structure that specifies the operation to be performed. The supported operations are LIO_READ (do a 'read' operation), LIO_WRITE (do a 'write' operation), and LIO_NOP (do no operation); these symbols are defined in `<aio.h>`.

See Also
lio_listio()
Handling Errors Caused by Asynchronous I/O Functions

This topic only applies to Windows* OS.

The `errno` macro is used to obtain the errors that occur during asynchronous request functions such as `aio_read()`, `aio_write()`, `aio_fsync()`, and `lio_listio()` or asynchronous control functions, such as `aio_cancel()`, `aio_error()`, `aio_return()`, and `aio_suspend()`.

The following example illustrates how `errno` can be used.

```c
#include <stdio.h>
#include <stdlib.h>
#include <aio.h>

struct aiocb    my_aio;
struct aiocb   *my_aio_list[1] = {&my_aio};

int main()
{
    int res;
    double  arr[123456];
    timespec_t  my_t = {1, 0};

    /* Data initialization */
    my_aio.aio_fildes = CreateFile("dat",
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ,
        NULL,
        OPEN_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    my_aio.aio_buf    = (volatile char *)arr;
    my_aio.aio_nbytes = sizeof(arr);

    /* Do asynchronous writing with computation overlapping */
    aio_write(&my_aio);
    do_compute(arr, 123456);

    /* Suspend the asynchronous writing for 1 sec */
    res = aio_suspend(my_aio_list, 1, &my_t);
    if ( res ) {
        /* The call was ended by timeout, before the indicated operations had completed. */
        if ( errno == EAGAIN ) {
            res = aio_suspend(my_aio_list, 1, 0);
            if ( res ) {
                printf("aio_suspend returned non-0\n");
            } else {
                printf("aio_suspend returned neither 0 nor EAGAIN\n");
            }
        } else {
            printf("aio_suspend returned neither 0 nor EAGAIN\n");
            return errno;
        }
    }

    CloseHandle(my_aio.aio_fildes);
    printf("\nPass\n");
}```
return 0;
}

In the example, the program executes an asynchronous write operation, using \texttt{aio\_write()}, overlapping with some computation, the \texttt{do\_compute()} function execution. The pending write operation is suspended for one second using \texttt{aio\_suspend()}. On successful execution of the asynchronous write operation, zero is returned. \texttt{EAGAIN} or any other error value is returned when the call is ended by timeout before the indicated operation has completed.

You can check \texttt{EAGAIN} using the \texttt{errno} macro.

**Intel's C++ Asynchronous I/O Class for Windows* Operating Systems**

This topic only applies to Windows* OS.

Intel's C++ asynchronous I/O template class, \texttt{async\_class}, is an implementation for the Windows* operating system on IA-32 (for C++ only) and Intel® 64 architectures.

The \texttt{async\_class} template class allows users to perform I/O operations asynchronously to the main program thread. In particular, the \texttt{async\_class} template class can be used to introduce asynchronous execution of I/O operations with the STL streams classes. Users can quickly switch any of the I/O operations of the STL streams to asynchronous mode with minimal changes to the application code. The template class \texttt{async\_class} is defined in the \texttt{aiostream.h} file.

**See Also**

Details of template class \texttt{async\_class}

**Template Class \texttt{async\_class}**

This topic only applies to Windows* OS.

Intel's C++ asynchronous I/O class implementation contains two main classes within the \texttt{async} namespace: the \texttt{async\_class} template class and the \texttt{thread\_control} base class.

The header/typedef definitions are as follows:

```cpp
namespace async {

    template<class A>
    class async_class:
    public thread_control, public A

}
```

The template class \texttt{async\_class} inherits support for asynchronous execution of I/O operations that are integrated within the base \texttt{thread\_control} class.

All functionality to control asynchronous execution of a queue of STL stream operations is encapsulated in the base class \texttt{thread\_control} and is inherited by template class \texttt{async\_class}.

In most cases it is enough to add the header file \texttt{aiostream.h} to the source file and declare the file object as an instance of the new template class \texttt{async:async\_class}. The initial stream class must be the parameter for the template class. Consequently, the defined output operator \texttt{<<} and input operator \texttt{>>} are executed asynchronously.

**NOTE**

The header file \texttt{aiostream.h} includes all necessary declarations for the STL stream I/O operations to add asynchronous functionality of the \texttt{thread\_control} class. It also contains the necessary declarations of extensions for the standard C++ STL streams I/O operations: output operator \texttt{>>} and input operator \texttt{<<}.
You can call synchronization method `wait()` to wait for completion of any I/O operations with the file object. If the `wait()` method is not called explicitly, it is called implicitly in the object destructor.

**Public Interface of Template Class async_class**

The following methods define the public interface of the template class `async_class`:

- `get_last_operation_id()`
- `wait()`
- `get_status()`
- `get_last_error()`
- `get_error_operation_id()`
- `stop_queue()`
- `resume_queue()`
- `clear_queue()`

**Library Restrictions**

Intel’s C++ asynchronous I/O template class does not control the integrity or validity of the objects during asynchronous operation. Such control should be done by the user.

For application stability in the Visual Studio 2003 environment, link the C++ part of `libacaio.lib` library with multi-threaded `msvcrt` run-time library. Use `/MT` or `/MTd` compiler option.

**See Also**

Example of Using async_class Template Class

**get_last_operation_id**

*Returns ID of the last added operation.*

---

**Syntax**

```cpp
void get_last_operation_id(void)
```

**Description**

This method returns the ID of the last added operation. Use this ID to get the status of operation or to wait for the operation to complete.

**Return Values**

Nothing

**wait**

*Stops execution of current thread.*

---

**Syntax**

```cpp
int wait(void)
```

```cpp
int wait(unsigned int operation_id)
```

**Description**

Method `wait(void)` stops execution of the current thread until all the asynchronous operations are completed.

Method `wait(operation_id)` stops execution of the current thread until the operation identified by `operation_id` is completed.
**Return Values**

-1: On error during queue execution

Call the `get_last_error()` method to check the error code.

---

### get_status

*Returns status of specified operation.*

**Syntax**

```c
void get_status(unsigned int operation_id)
```

**Description**

This method returns the status of an operation, specified by `operation_id`, without stopping current thread execution.

**Return Values**

- `STATUS_WAIT`: Operation is waiting for execution.
- `STATUS_COMPLETED`: Operation finished execution.
- `STATUS_ERROR`: An error occurred during operation execution.
- `STATUS_EXECUTE`: Operation is executing.
- `STATUS_BLOCKED`: Execution of the queue was blocked after some earlier errors.

---

### get_last_error

*Returns the error code of the last failed operation.*

**Syntax**

```c
unsigned int get_last_error()
```

**Description**

This method returns the error code of the last failed operation. If the error occurs during the execution of an asynchronous operation, the asynchronous thread stops executing the queue of asynchronous operations and waits for new user requests.

To obtain the error status, use the `wait()` and `get_status()` methods.

**Return Values**

Error code of last failed operation.

This error code is equal to the value returned by `GetLastError()` function on the Windows* platform. If the error occurs during the execution of an asynchronous operation, the asynchronous thread stops executing the queue of asynchronous operations and waits for new user requests.

---

### get_error_operation_id

*Returns the ID of the last failed operation.*

**Syntax**

```c
unsigned int get_error_operation_id()
```
Description
This method returns the ID of the last failed operation. If the error occurs during the execution of an asynchronous operation, the asynchronous thread stops executing the queue of the asynchronous operations and waits for new user requests.

To obtain the error status of the failed operation, use the `wait()` and `get_status()` methods.

Return Values
ID of last failed operation.

stop_queue
Stops queue execution.

Syntax
int stop_queue()

Description
This method allows you to control the asynchronous operations queue by stopping queue execution.

Return Values
0: On success
-1: On error

resume_queue
Resumes queue execution.

Syntax
int resume_queue()

Description
This method allows you to control the asynchronous operations queue by resuming queue execution.

Return Values
0: On success
-1: On error

clear_queue
Clears stopped or error-interrupted queues.

Syntax
void push_back_operation(class base_operation*)

Description
This method clears the content of stopped queues or queues interrupted by errors.

Return Values
0: On success
-1: On error
Example for Using async_class Template Class

The following example illustrates how Intel's C++ asynchronous I/O template class can be used. Consider the following code that writes arrays of floats to an external file.

```cpp
// Data is array of floats
std::vector<float> v(10000);

// User defines new operator << for std::vector<float> type
std::ofstream& operator << (std::ofstream & str, std::vector<float> & vec)
{
    // User's output actions
    ...
}

// Output file declaration - object of standard ofstream STL class
std::ofstream external_file("output.txt");

// Output operations
external_file << v;
```

The following code illustrates the changes to be made to the above code to execute the output operation asynchronously.

```cpp
// Add new header to support STL asynchronous IO operations
#include <aiostream.h>

std::vector<float> v(10000);

std::ofstream& operator << (std::ofstream & str, std::vector<float> & vec)
{
    // User's output actions
    ...
}

// Declare output file as the instance of new async::async_class template class.
// New inherited from STL ofstream type is declared
async::async_class<std::ofstream> external_file("output.txt");

// Add stop operation, to wait the completion of all asynchronous IO operations
external_file.wait();
```

Performance Recommendations

It is recommended not to use asynchronous mode for small objects. For example, do not use asynchronous mode when the output standard type value in a loop where execution of other loop operations takes less time than output of the same value to the STL stream.

However, if you can find the balance between output of small data and its previous calculation inside the loop, you still have some stable performance improvement.

For example, in the following code, the program reads two matrices from external files, calculates the elements of a third matrix, and prints out the elements inside the loop.

```cpp
#define ARR_LEN 900

std::ifstream fA("A.txt");
fA >> A;
std::ifstream fB("B.txt");
```
fB >> B;
std::ofstream fC(f);

for(int i=0; i< ARR_LEN; i++)
{
    for(int j=0; j< ARR_LEN; j++)
    {
        C[i][j] = 0;
        for(int k=0; k < ARR_LEN; k++)
            C[i][j] += A[i][k]*B[k][j]*sin((float)(k))*cos((float)(-k))*
                        sin((float)(k+1))*cos((float)(-k-1));
        fC << C[i][j] << std::endl;
    }
}

By increasing matrix size, you can also achieve performance improvement during parallel data reading from two files.

### IEEE 754-2008 Binary Floating-Point Conformance Library

The Intel® IEEE 754-2008 Binary Floating-Point Conformance Library provides all operations mandated by the IEEE 754-2008 standard for binary32 and binary64 binary floating-point interchange formats.

Many routines in the libbfp754 Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

### Overview: Intel® IEEE 754-2008 Binary Floating-Point Conformance Library

The Intel® IEEE 754-2008 Binary Floating-Point Conformance Library provides all operations mandated by the IEEE 754-2008 standard for binary32 and binary64 binary floating-point interchange formats. The minimum requirements for correct operation of the library are an Intel® Pentium® 4 processor and an operating system supporting Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions.

The library supports all four rounding-direction attributes mandated by the IEEE 754-2008 standard for binary floating-point arithmetic: roundTiesToEven, roundTowardPositive, roundTowardNegative, roundTowardZero. The additional rounding-direction attribute, roundTiesToAway, is not required by the standard, hence, not fully supported in this library. The default rounding-direction attribute is set as roundTiesToEven.

The library also supports all mandated exceptions (invalid operation, division by zero, overflow, underflow, and inexact) and sets flags accordingly under default exception handling. Alternate exception handling, which is optional in the standard, is not supported.

The bfp754.h header file includes prototypes for the library functions. For a complete list of the functions available, refer to the Function List. The user also needs to specify linker option -lbfp754 and floating-point semantics control option -fp-model source -fp-model except in order to use the library. **Note:** -fp-model is only available for C++; it is not available for DPC++.

Many routines in the libbfp754 Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.
Operations
The IEEE standard 754-2008 defines four types of operations.

1. General-computational operations that produce correctly rounded floating-point or integer results. These operations might signal the floating-point exceptions.
2. Quiet-computational operations that produce floating-point results. These operations do not signal any floating-point exceptions.
3. Signaling-computational operations that produce no floating-point results. These operations might signal floating-point exceptions.
4. Non-computational operations that produce no floating-point results. These operations do not signal floating-point exceptions.

The standard also distinguishes among operations by their floating-point operand formats and result format for general-computational operations:

1. Homogenous general-computational operations whose floating-point operands and floating-point result are in the same format.
2. formatOf general-computational operations whose floating-point operands and floating-point result have different formats.

NOTE
The IEEE 754-2008 standard requires that all formatOf general-computational operations be computed without any loss of precision before converting to the destination format. This may differ from how these operations are implemented on most hardware and software.

For example, when all operands are in binary64 format and the destination format is binary32, most hardware and software implementations would first compute an intermediate result rounded in binary64 and then convert the intermediate result to binary32. This double rounding procedure may produce a result different from what is defined in the standard under certain rounding mode. For example:

\[ x = 0x3ff0000010000000 = 1.000000000000000000000001_2, \]
\[ y = 0x3ca0000000000000 = 1.0_2 \times 2^{-53} \]
\[ x+y = 1.0000000000000000000000100000000000000000000001_2 \]

When the rounding-direction attribute is set to roundTiesToEven, using double rounding procedure, the addition result rounds to \[ 1.00000000000000000000001_2 \] (0x3f800000) in binary64, which would then round to 1 (0x3f800000) in binary32. On the other hand, according to the standard, the addition result should round to \[ 1.000000000000000000000001_2 \] (0x3f800001) in binary32.

Data Types
The following table correlates the names of the formats used in defining operations in the standard with their C99 types used in this library.

<table>
<thead>
<tr>
<th>Format Name</th>
<th>Definition</th>
<th>C99 Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary32</td>
<td>IEEE 754-2008 binary32 interchange format</td>
<td>float</td>
</tr>
<tr>
<td>Format Name</td>
<td>Definition</td>
<td>C99 Type</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>binary64</td>
<td>IEEE 754-2008 binary64 interchange format</td>
<td>double</td>
</tr>
<tr>
<td>int</td>
<td>Integer operand formats</td>
<td>int, unsigned int, long</td>
</tr>
<tr>
<td></td>
<td></td>
<td>long int, unsigned long</td>
</tr>
<tr>
<td>int32</td>
<td>Signed 32-bit integer</td>
<td>int</td>
</tr>
<tr>
<td>uint32</td>
<td>Unsigned 32-bit integer</td>
<td>unsigned int</td>
</tr>
<tr>
<td>int64</td>
<td>Signed 64-bit integer</td>
<td>long long int</td>
</tr>
<tr>
<td>uint64</td>
<td>Unsigned 64-bit integer</td>
<td>unsigned long long int</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean value represented by generic integer type</td>
<td>int</td>
</tr>
<tr>
<td>enum</td>
<td>Enumerated values of floating-point class</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>Enumerated values of floating-point radix</td>
<td>int</td>
</tr>
<tr>
<td>logBFormat</td>
<td>Type for the destination of the logB operation and the scale exponent operand of the scaleB operation</td>
<td>int</td>
</tr>
<tr>
<td>decimalCharacterSequence</td>
<td>Decimal character sequence</td>
<td>char*</td>
</tr>
<tr>
<td>hexCharacterSequence</td>
<td>Hexadecimal-significand character sequence</td>
<td></td>
</tr>
<tr>
<td>exceptionGroup</td>
<td>Set of exceptions as a set of booleans</td>
<td>int</td>
</tr>
<tr>
<td>flags</td>
<td>Set of status flags</td>
<td>int</td>
</tr>
<tr>
<td>binaryRoundingDirection</td>
<td>Rounding direction for binary</td>
<td>int</td>
</tr>
<tr>
<td>modeGroup</td>
<td>Dynamically-specifiable modes</td>
<td>int</td>
</tr>
<tr>
<td>void</td>
<td>No explicit operand or result</td>
<td>void</td>
</tr>
</tbody>
</table>

**See Also**

**Function List**

**Using the Intel® IEEE 754-2008 Binary Floating-Point Conformance Library**

Many routines in the *libbfp754* Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

To use the library, include the header file, `bfp754.h`, in your program.
Here is an example program illustrating the use of the library on Linux.* OS. This content is specific to C++; it does not apply to DPC++.

```c
//binary.c
#include <stdio.h>
#include <bfp754.h>
int main(){
    double a64, b64;
    float c32;
    a64 = 1.000000059604644775390625;
    b64 = 1.1102230246251565404236316680908203125e-16;
    c32 = __binary32_add_binary64_binary64(a64, b64);
    printf("The addition result using the library: %8.8f\n", c32);
    c32 = a64 + b64;
    printf("The addition result without the library: %8.8f\n", c32);
    return 0;
}
```

The command for compiling binary.c is:

```
icx -fp-model source -fp-model except binary.c –lbfp754
```

The output of a.out will look similar to the following:

```
The addition result using the library: 1.00000012
The addition result without the library: 1.00000000
```

### Function List

Many routines in the libbfp754 Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for homogeneous general-computational operations:

<table>
<thead>
<tr>
<th>Routine or Function Group</th>
<th>IEEE standard equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>round_integral_nearest_even</td>
<td>roundToIntegralTiesToEven</td>
</tr>
<tr>
<td>round_integral_nearest_away</td>
<td>roundToIntegralTiesToAway</td>
</tr>
<tr>
<td>round_integral_zero</td>
<td>roundToIntegralTowardZero</td>
</tr>
<tr>
<td>round_integral_positive</td>
<td>roundToIntegralTowardPositive</td>
</tr>
<tr>
<td>round_integral_negative</td>
<td>roundToIntegralTowardNegative</td>
</tr>
<tr>
<td>round_integral_exact</td>
<td>roundToIntegralExact</td>
</tr>
<tr>
<td>next_up</td>
<td>nextUp</td>
</tr>
<tr>
<td>next_down</td>
<td>nextDown</td>
</tr>
<tr>
<td>rem</td>
<td>remainder</td>
</tr>
<tr>
<td>minnum</td>
<td>minNum</td>
</tr>
<tr>
<td>maxnum</td>
<td>maxNum</td>
</tr>
<tr>
<td>minnum_mag</td>
<td>minNumMag</td>
</tr>
<tr>
<td>maxnum_mag</td>
<td>maxNumMag</td>
</tr>
<tr>
<td>scalbn</td>
<td>scaleB</td>
</tr>
<tr>
<td>ilogb</td>
<td>logB</td>
</tr>
</tbody>
</table>
The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for `formatOf` general-computational operations:

<table>
<thead>
<tr>
<th>Routine or Function Group</th>
<th>IEEE standard equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>addition</td>
</tr>
<tr>
<td>sub</td>
<td>subtraction</td>
</tr>
<tr>
<td>mul</td>
<td>multiplication</td>
</tr>
<tr>
<td>div</td>
<td>division</td>
</tr>
<tr>
<td>sqrt</td>
<td>squareRoot</td>
</tr>
<tr>
<td>fma</td>
<td>fusedMultiplyAdd</td>
</tr>
<tr>
<td>from_int32</td>
<td>convert</td>
</tr>
<tr>
<td>from_uint32</td>
<td></td>
</tr>
<tr>
<td>from_int64</td>
<td></td>
</tr>
<tr>
<td>from_uint64</td>
<td></td>
</tr>
<tr>
<td>to_int32_rnint</td>
<td>convertToIntegerTiesToEven</td>
</tr>
<tr>
<td>to_uint32_rnint</td>
<td></td>
</tr>
<tr>
<td>to_int64_rnint</td>
<td></td>
</tr>
<tr>
<td>to_uint64_rnint</td>
<td></td>
</tr>
<tr>
<td>to_int32_int</td>
<td>convertToIntegerTowardZero</td>
</tr>
<tr>
<td>to_uint32_int</td>
<td></td>
</tr>
<tr>
<td>to_int64_int</td>
<td></td>
</tr>
<tr>
<td>to_uint64_int</td>
<td></td>
</tr>
<tr>
<td>to_int32.ceil</td>
<td>convertToIntegerTowardPositive</td>
</tr>
<tr>
<td>to_uint32.ceil</td>
<td></td>
</tr>
<tr>
<td>to_int64.ceil</td>
<td></td>
</tr>
<tr>
<td>to_uint64.ceil</td>
<td></td>
</tr>
<tr>
<td>to_int32.floor</td>
<td>convertToIntegerTowardNegative</td>
</tr>
<tr>
<td>to_uint32.floor</td>
<td></td>
</tr>
<tr>
<td>to_int64.floor</td>
<td></td>
</tr>
<tr>
<td>to_uint64.floor</td>
<td></td>
</tr>
<tr>
<td>to_int32_rninta</td>
<td>convertToIntegerTiesToAway</td>
</tr>
<tr>
<td>to_uint32_rninta</td>
<td></td>
</tr>
<tr>
<td>to_int64_rninta</td>
<td></td>
</tr>
<tr>
<td>to_uint64_rninta</td>
<td></td>
</tr>
<tr>
<td>to_int32_xrnint</td>
<td>convertToIntegerExactTiesToEven</td>
</tr>
<tr>
<td>to_uint32_xrnint</td>
<td></td>
</tr>
<tr>
<td>to_int64_xrnint</td>
<td></td>
</tr>
<tr>
<td>to_uint64_xrnint</td>
<td></td>
</tr>
<tr>
<td>Routine or Function Group</td>
<td>IEEE standard equivalent</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>to_int32_xint</td>
<td>convertToIntegerExactTowardZero</td>
</tr>
<tr>
<td>to_uint32_xint</td>
<td>convertToIntegerExactTowardPositive</td>
</tr>
<tr>
<td>to_int64_xint</td>
<td>convertToIntegerExactTowardNegative</td>
</tr>
<tr>
<td>to_uint64_xint</td>
<td>convertToIntegerExactTiesToAway</td>
</tr>
<tr>
<td>to_int32_xceil</td>
<td>convertToIntegerExactTowardPositive</td>
</tr>
<tr>
<td>to_uint32_xceil</td>
<td>convertToIntegerExactTowardPositive</td>
</tr>
<tr>
<td>to_int64_xceil</td>
<td>convertToIntegerExactTowardNegative</td>
</tr>
<tr>
<td>to_uint64_xceil</td>
<td>convertToIntegerExactTiesToAway</td>
</tr>
<tr>
<td>to_int32_xfloor</td>
<td>convertToIntegerExactTowardNegative</td>
</tr>
<tr>
<td>to_uint32_xfloor</td>
<td>convertToIntegerExactTiesToAway</td>
</tr>
<tr>
<td>to_int64_xfloor</td>
<td>convertToIntegerExactTiesToAway</td>
</tr>
<tr>
<td>to_uint64_xfloor</td>
<td>convertToIntegerExactTiesToAway</td>
</tr>
<tr>
<td>to_int32_xrninta</td>
<td>convertToIntegerExactTiesToAway</td>
</tr>
<tr>
<td>to_uint32_xrninta</td>
<td>convertToIntegerExactTiesToAway</td>
</tr>
<tr>
<td>to_int64_xrninta</td>
<td>convertToIntegerExactTiesToAway</td>
</tr>
<tr>
<td>to_uint64_xrninta</td>
<td>convertToIntegerExactTiesToAway</td>
</tr>
<tr>
<td>binary32_to_binary64</td>
<td>convertFormat</td>
</tr>
<tr>
<td>binary64_to_binary32</td>
<td>convertFormat</td>
</tr>
<tr>
<td>from_string</td>
<td>convertFromDecimalCharacter</td>
</tr>
<tr>
<td>to_string</td>
<td>convertToDecimalCharacter</td>
</tr>
<tr>
<td>from_hexstring</td>
<td>convertFromHexCharacter</td>
</tr>
<tr>
<td>to_hexstring</td>
<td>convertToHexCharacter</td>
</tr>
</tbody>
</table>

The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for quiet-computational operations:

<table>
<thead>
<tr>
<th>Routine or Function Group</th>
<th>IEEE standard equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>copy</td>
</tr>
<tr>
<td>negate</td>
<td>negate</td>
</tr>
<tr>
<td>abs</td>
<td>abs</td>
</tr>
<tr>
<td>copysign</td>
<td>copySign</td>
</tr>
</tbody>
</table>

The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for signaling-computational operations:

<table>
<thead>
<tr>
<th>Routine or Function Group</th>
<th>IEEE standard equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>quiet_equal</td>
<td>compareQuietEqual</td>
</tr>
<tr>
<td>quiet_not_equal</td>
<td>compareQuietNotEqual</td>
</tr>
<tr>
<td>signaling_equal</td>
<td>compareSignalingEqual</td>
</tr>
<tr>
<td>signaling_greater</td>
<td>compareSignalingGreater</td>
</tr>
</tbody>
</table>
The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for non-computational operations:

<table>
<thead>
<tr>
<th>Routine or Function Group</th>
<th>IEEE standard equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>signaling_greater_equal</td>
<td>compareSignalingGreaterEqual</td>
</tr>
<tr>
<td>signaling_less</td>
<td>compareSignalingLess</td>
</tr>
<tr>
<td>signaling_less_equal</td>
<td>compareSignalingLessEqual</td>
</tr>
<tr>
<td>signaling_not_equal</td>
<td>compareSignalingNotEqual</td>
</tr>
<tr>
<td>signaling_not_greater</td>
<td>compareSignalingNotGreater</td>
</tr>
<tr>
<td>signaling_less_unordered</td>
<td>compareSignalingLessUnordered</td>
</tr>
<tr>
<td>signaling_not_less</td>
<td>compareSignalingNotLess</td>
</tr>
<tr>
<td>signaling_greater_unordered</td>
<td>compareSignalingGreaterUnordered</td>
</tr>
<tr>
<td>quiet_greater</td>
<td>compareQuietGreater</td>
</tr>
<tr>
<td>quiet_greater_equal</td>
<td>compareQuietGreaterEqual</td>
</tr>
<tr>
<td>quiet_less</td>
<td>compareQuietLess</td>
</tr>
<tr>
<td>quiet_less_equal</td>
<td>compareQuietLessEqual</td>
</tr>
<tr>
<td>quiet_unordered</td>
<td>compareQuietUnordered</td>
</tr>
<tr>
<td>quiet_not_greater</td>
<td>compareQuietNotGreater</td>
</tr>
<tr>
<td>quiet_less_unordered</td>
<td>compareQuietNotLess</td>
</tr>
<tr>
<td>quiet_greater_unordered</td>
<td>compareQuietGreaterUnordered</td>
</tr>
<tr>
<td>quiet_ordered</td>
<td>compareQuietOrdered</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Routine or Function Group</th>
<th>IEEE standard equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>is754version1985</td>
<td>is754version1985</td>
</tr>
<tr>
<td>is754version2008</td>
<td>is754version2008</td>
</tr>
<tr>
<td>class</td>
<td>class</td>
</tr>
<tr>
<td>isSignMinus</td>
<td>isSignMinus</td>
</tr>
<tr>
<td>isNormal</td>
<td>isNormal</td>
</tr>
<tr>
<td>isFinite</td>
<td>isFinite</td>
</tr>
<tr>
<td>isZero</td>
<td>isZero</td>
</tr>
<tr>
<td>isSubnormal</td>
<td>isSubnormal</td>
</tr>
<tr>
<td>isInfinite</td>
<td>isInfinite</td>
</tr>
<tr>
<td>isNaN</td>
<td>isNaN</td>
</tr>
<tr>
<td>isSignaling</td>
<td>isSignaling</td>
</tr>
<tr>
<td>isCanonical</td>
<td>isCanonical</td>
</tr>
<tr>
<td>radix</td>
<td>radix</td>
</tr>
<tr>
<td>totalOrder</td>
<td>totalOrder</td>
</tr>
<tr>
<td>totalOrderMag</td>
<td>totalOrderMag</td>
</tr>
<tr>
<td>Routine or Function Group</td>
<td>IEEE standard equivalent</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>lowerFlags</td>
<td>lowerFlags</td>
</tr>
<tr>
<td>raiseFlags</td>
<td>raiseFlags</td>
</tr>
<tr>
<td>testFlags</td>
<td>testFlags</td>
</tr>
<tr>
<td>testSavedFlags</td>
<td>testSavedFlags</td>
</tr>
<tr>
<td>restoreFlags</td>
<td>restoreFlags</td>
</tr>
<tr>
<td>saveFlags</td>
<td>saveAllFlags</td>
</tr>
<tr>
<td>getBinaryRoundingDirection</td>
<td>getBinaryRoundingDirection</td>
</tr>
<tr>
<td>setBinaryRoundingDirection</td>
<td>setBinaryRoundingDirection</td>
</tr>
<tr>
<td>saveModes</td>
<td>saveModes</td>
</tr>
<tr>
<td>restoreModes</td>
<td>restoreModes</td>
</tr>
<tr>
<td>defaultMode</td>
<td>defaultModes</td>
</tr>
</tbody>
</table>

**Homogeneous General-Computational Operations Functions**

Many routines in the `libbfp754` Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for homogeneous general-computational operations:

**round_integral_nearest_even**

**Description:** The function rounds floating-point number $x$ to its nearest integral value, with the halfway (tied) case rounding to even.

**Calling interface:**

```
float __binary32_round_integral_nearest_even(float x);
double __binary64_round_integral_nearest_even(double x);
```

**round_integral_nearest_away**

**Description:** The function rounds floating-point number $x$ to its nearest integral value, with the halfway (tied) case rounding away from zero.

**Calling interface:**

```
float __binary32_round_integral_nearest_away(float x);
double __binary64_round_integral_nearest_away(double x);
```

**round_integral_zero**

**Description:** The function rounds floating-point number $x$ to the closest integral value toward zero.

**Calling interface:**

```
float __binary32_round_integral_zero(float x);
double __binary64_round_integral_zero(double x);
```

**round_integral_positive**

**Description:** The function rounds floating-point number $x$ to the closest integral value toward positive infinity.

**Calling interface:**

```
float __binary32_round_integral_positive(float x);
double __binary64_round_integral_positive(double x);

**roundintegral_negative**

**Description:** The function rounds floating-point number $x$ to the closest integral value toward negative infinity.

**Calling interface:**
float __binary32_round_integral_negative(float x);
double __binary64_round_integral_negative(double x);

**roundintegral_exact**

**Description:** The function rounds floating-point number $x$ to the closest integral value according to the rounding-direction applicable.

**Calling interface:**
float __binary32_round_integral_exact(float x);
double __binary64_round_integral_exact(double x);

**next_up**

**Description:** The function returns the least floating-point number in the same format as $x$ that is greater than $x$.

**Calling interface:**
float __binary32_next_up(float x);
double __binary64_next_up(double x);

**next_down**

**Description:** The function returns the largest floating-point number in the same format as $x$ that is less than $x$.

**Calling interface:**
float __binary32_next_down(float x);
double __binary64_next_down(double x);

**rem**

**Description:** The function returns the remainder of $x$ and $y$.

**Calling interface:**
float __binary32_rem(float x, float y);
double __binary64_rem(double x, double y);

**minnum**

**Description:** The function returns the minimal value of $x$ and $y$.

**Calling interface:**
float __binary32_minnum(float x, float y);
double __binary64_minnum(double x, double y);

**maxnum**

**Description:** The function returns the maximal value of $x$ and $y$.

**Calling interface:**
float __binary32_maxnum(float x, float y);
double __binary64_maxnum(double x, double y);

**minnum_mag**

**Description:** The function returns the minimal absolute value of x and y.

**Calling interface:**
float __binary32_minnum_mag(float x, float y);
double __binary64_minnum_mag(double x, double y);

**maxnum_mag**

**Description:** The function returns the maximal absolute value of x and y.

**Calling interface:**
float __binary32_maxnum_mag(float x, float y);
double __binary64_maxnum_mag(double x, double y);

**scalbn**

**Description:** The function computes $x \times 2^n$ for integer value n.

**Calling interface:**
float __binary32_scalbn(float x, int n);
double __binary64_scalbn(double x, int n);

**ilogb**

**Description:** The function returns the exponent part of x as integer.

**Calling interface:**
int __binary32_ilogb(float x);
int __binary64_ilogb(double x);

**General-Computational Operations Functions**

Many routines in the `libbfp754` Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for formatOf general-computational operations:

**add**

**Description:** The function computes the addition of two floating-point numbers; the result is then converted to the destination format.

**Calling interface:**
float __binary32_add_binary32_binary32(float x, float y);
float __binary32_add_binary32_binary64(float x, double y);
float __binary32_add_binary64_binary32(double x, float y);
float __binary32_add_binary64_binary64(double x, double y);
double __binary64_add_binary32_binary32(float x, float y);
double __binary64_add_binary32_binary64(float x, double y);
double __binary64_add_binary64_binary32(double x, float y);
double __binary64_add_binary64_binary64(double x, double y);
sub

Description: The function computes the subtraction of two floating-point numbers; the result is then converted to the destination format.

Calling interface:
float __binary32_sub_binary32_binary32(float x, float y);
float __binary32_sub_binary32_binary64(float x, double y);
float __binary32_sub_binary64_binary32(double x, float y);
float __binary32_sub_binary64_binary64(double x, double y);
double __binary64_sub_binary32_binary32(float x, float y);
double __binary64_sub_binary32_binary64(float x, double y);
double __binary64_sub_binary64_binary32(double x, float y);
double __binary64_sub_binary64_binary64(double x, double y);

mul

Description: The function computes the multiplication of two floating-point numbers; the result is then converted to the destination format.

Calling interface:
float __binary32_mul_binary32_binary32(float x, float y);
float __binary32_mul_binary32_binary64(float x, double y);
float __binary32_mul_binary64_binary32(double x, float y);
float __binary32_mul_binary64_binary64(double x, double y);
double __binary64_mul_binary32_binary32(float x, float y);
double __binary64_mul_binary32_binary64(float x, double y);
double __binary64_mul_binary64_binary32(double x, float y);
double __binary64_mul_binary64_binary64(double x, double y);

div

Description: The function computes the division of two floating-point numbers; the result is then converted to the destination format.

Calling interface:
float __binary32_div_binary32_binary32(float x, float y);
float __binary32_div_binary32_binary64(float x, double y);
float __binary32_div_binary64_binary32(double x, float y);
float __binary32_div_binary64_binary64(double x, double y);
double __binary64_div_binary32_binary32(float x, float y);
double __binary64_div_binary32_binary64(float x, double y);
double __binary64_div_binary64_binary32(double x, float y);
double __binary64_div_binary64_binary64(double x, double y);

sqrt

Description: The function computes the square root of floating-point number; the result is then converted to the destination format.

Calling interface:
float __binary32_sqrt_binary32(float x);
float __binary32_sqrt_binary64(double x);
double __binary32_sqrt_binary32(float x);
double __binary32_sqrt_binary64(double x);
fma

**Description:** The function computes the fused multiply and add of three floating-point numbers \( x, y, \) and \( z \) as \((x \times y) + z\); the result is then converted to the destination format.

**Calling interface:**

```c
float __binary32_fma_binary32_binary32_binary32(float x, float y, float z);
float __binary32_fma_binary32_binary32_binary64(float x, float y, double z);
float __binary32_fma_binary32_binary64_binary32(float x, double y, float z);
float __binary32_fma_binary32_binary64_binary64(float x, double y, double z);
float __binary32_fma_binary64_binary32_binary32(double x, float y, float z);
float __binary32_fma_binary64_binary32_binary64(double x, float y, double z);
float __binary32_fma_binary64_binary64_binary32(double x, double y, float z);
float __binary32_fma_binary64_binary64_binary64(double x, double y, double z);
float __binary64_fma_binary32_binary32_binary32(float x, float y, float z);
float __binary64_fma_binary32_binary32_binary64(float x, float y, double z);
float __binary64_fma_binary32_binary64_binary32(double x, float y, float z);
float __binary64_fma_binary32_binary64_binary64(double x, float y, double z);
float __binary64_fma_binary64_binary32_binary32(double x, double y, float z);
float __binary64_fma_binary64_binary32_binary64(double x, double y, double z);
float __binary64_fma_binary64_binary64_binary32(double x, double y, float z);
float __binary64_fma_binary64_binary64_binary64(double x, double y, double z);
```

**from_int32 / from_uint32 / from_int64 / from_uint64**

**Description:** This function converts integral values in the specified integer format to floating-point number.

**Calling interface:**

```c
float __binary32_from_int32(int n);
float __binary32_from_uint32(unsigned int n);
float __binary32_from_int64(long long int n);
float __binary32_from_uint64(unsigned long long int n);
float __binary64_from_int32(int n);
float __binary64_from_uint32(unsigned int n);
float __binary64_from_int64(long long int n);
float __binary64_from_uint64(unsigned long long int n);
```

**to_int32_rnint / to_uint32_rnint / to_int64_rnint / to_uint64_rnint**

**Description:** This function rounds floating-point number to the nearest integral value in the specified integer format, with halfway cases rounded to even, without signaling the inexact exception.

**Calling interface:**

```c
int __binary32_to_int32_rnint(float x);
int __binary32_to_uint32_rnint(double x);
unsigned int __binary32_to_uint32_rnint(float x);
unsigned int __binary32_to_uint64_rnint(double x);
long long int __binary32_to_int64_rnint(float x);
long long int __binary32_to_int64_rnint(double x);
unsigned long long int __binary32_to_uint64_rnint(float x);
unsigned long long int __binary64_to_uint64_rnint(double x);
```
**to_int32_int / to_uint32_int / to_int64_int / to_uint64_int**

**Description:** This function rounds floating-point number to the nearest integral value in the specified integer format toward zero, without signaling the inexact exception.

**Calling interface:**
```
int __binary32_to_int32_int(float x);
int __binary64_to_int32_int(double x);
unsigned int __binary32_to_uint32_int(float x);
unsigned int __binary64_to_uint32_int(double x);
long long int __binary32_to_int64_int(float x);
long long int __binary64_to_int64_int(double x);
unsigned long long int __binary32_to_uint64_int(float x);
unsigned long long int __binary64_to_uint64_int(double x);
```

**to_int32.ceil / to_uint32.ceil / to_int64.ceil / to_uint64.ceil**

**Description:** This function rounds floating-point number to the nearest integral value in the specified integer format toward positive infinity, without signaling the inexact exception.

**Calling interface:**
```
int __binary32_to_int32_ceil(float x);
int __binary64_to_int32_ceil(double x);
unsigned int __binary32_to_uint32_ceil(float x);
unsigned int __binary64_to_uint32_ceil(double x);
long long int __binary32_to_int64_ceil(float x);
long long int __binary64_to_int64_ceil(double x);
unsigned long long int __binary32_to_uint64_ceil(float x);
unsigned long long int __binary64_to_uint64_ceil(double x);
```

**to_int32.floor / to_uint32.floor / to_int64.floor / to_uint64.floor**

**Description:** This function rounds floating-point number to the nearest integral value in the specified integer format toward negative infinity, without signaling the inexact exception.

**Calling interface:**
```
int __binary32_to_int32_floor(float x);
int __binary64_to_int32_floor(double x);
unsigned int __binary32_to_uint32_floor(float x);
unsigned int __binary64_to_uint32_floor(double x);
long long int __binary32_to_int64_floor(float x);
long long int __binary64_to_int64_floor(double x);
unsigned long long int __binary32_to_uint64_floor(float x);
unsigned long long int __binary64_to_uint64_floor(double x);
```

**to_int32_rninta / to_uint32_rninta / to_int64_rninta / to_uint64_rninta**

**Description:** This function rounds floating-point number to the nearest integral value in the specified integer format, with halfway cases rounded away from zero, without signaling the inexact exception.

**Calling interface:**
```
int __binary32_to_int32_rninta(float x);
int __binary64_to_int32_rninta(double x);
unsigned int __binary32_to_uint32_rninta(float x);
unsigned int __binary64_to_uint32_rninta(double x);
long long int __binary32_to_int64_rninta(float x);
long long int __binary64_to_int64_rninta(double x);
```
long long int __binary64_to_int64_rninta(double x);
unsigned long long int __binary32_to_uint64_rninta(float x);
unsigned long long int __binary64_to_uint64_rninta(double x);

to_int32_xrnint / to_uint32_xrnint / to_int64_xrnint / to_uint64_xrnint

Description: This function rounds floating-point number to the nearest integral value in the specified integer format, with half-way cases rounded to even, signaling if inexact.

Calling interface:
int __binary32_to_int32_xrnint(float x);
int __binary64_to_int32_xrnint(double x);
unsigned int __binary32_to_uint32_xrnint(float x);
unsigned int __binary64_to_uint32_xrnint(double x);
long long int __binary32_to_int64_xrnint(float x);
long long int __binary64_to_int64_xrnint(double x);
unsigned long long int __binary32_to_uint64_xrnint(float x);
unsigned long long int __binary64_to_uint64_xrnint(double x);

to_int32_xint / to_uint32_xint / to_int64_xint / to_uint64_xint

Description: This function rounds floating-point number to the nearest integral value in the specified integer format toward zero, signaling if inexact.

Calling interface:
int __binary32_to_int32_xint(float x);
int __binary64_to_int32_xint(double x);
unsigned int __binary32_to_uint32_xint(float x);
unsigned int __binary64_to_uint32_xint(double x);
long long int __binary32_to_int64_xint(float x);
long long int __binary64_to_int64_xint(double x);
unsigned long long int __binary32_to_uint64_xint(float x);
unsigned long long int __binary64_to_uint64_xint(double x);

to_int32_xceil / to_uint32_xceil / to_int64_xceil / to_uint64_xceil

Description: This function rounds floating-point number to the nearest integral value in the specified integer format toward positive infinity, signaling if inexact.

Calling interface:
int __binary32_to_int32_xceil(float x);
int __binary64_to_int32_xceil(double x);
unsigned int __binary32_to_uint32_xceil(float x);
unsigned int __binary64_to_uint32_xceil(double x);
long long int __binary32_to_int64_xceil(float x);
long long int __binary64_to_int64_xceil(double x);
unsigned long long int __binary32_to_uint64_xceil(float x);
unsigned long long int __binary64_to_uint64_xceil(double x);

to_int32_xfloor / to_uint32_xfloor / to_int64_xfloor / to_uint64_xfloor

Description: This function rounds floating-point number to the nearest integral value in the specified integer format toward negative infinity, signaling if inexact.

Calling interface:
int __binary32_to_int32_xfloor(float x);
int __binary64_to_int32_xfloor(double x);
unsigned int __binary32_to_uint32_xfloor(float x);
unsigned int __binary64_to_uint32_xfloor(double x);
long long int __binary32_to_int64_xfloor(float x);
long long int __binary64_to_int64_xfloor(double x);
unsigned long long int __binary32_to_uint64_xfloor(float x);
unsigned long long int __binary64_to_uint64_xfloor(double x);

to_int32_xrninta / to_uint32_xrninta / to_int64_xrninta / to_uint64_xrninta

Description: This function rounds floating-point number to the nearest integral value in the specified integer format, with halfway cases rounded away from zero, signaling if inexact.

Calling interface:
int __binary32_to_int32_xrninta(float x);
int __binary64_to_int32_xrninta(double x);
unsigned int __binary32_to_uint32_xrninta(float x);
unsigned int __binary64_to_uint32_xrninta(double x);
long long int __binary32_to_int64_xrninta(float x);
long long int __binary64_to_int64_xrninta(double x);
unsigned long long int __binary32_to_uint64_xrninta(float x);
unsigned long long int __binary64_to_uint64_xrninta(double x);

binary32_to_binary64

Description: This function converts floating-point number in binary32 format to binary64 format.

Calling interface:
double __binary32_to_binary64(float x);

binary64_to_binary32

Description: This function rounds floating-point number in binary64 format to binary32 format.

Calling interface:
float __binary64_to_binary32(double x);

from_string

Description: This function converts decimal character sequence to floating-point number.

Calling interface:
float __binary32_from_string(char * s);
double __binary64_from_string(char * s);

to_string

Description: This function converts floating-point number to decimal character sequence.

Calling interface:
void __binary32_to_string(char * s, float x);
void __binary64_to_string(char * s, double x);

from_hexstring

Description: This function converts hexadecimal character sequence to floating-point number.
Calling interface:
float __binary32_from_hexstring(char * s);
double __binary64_from_hexstring(char * s);

to_hexstring
Description: This function converts floating-point number to hexadecimal character sequence.

Calling interface:
void__binary32_to_hexstring(cgar * s, float x);
void__binary64_to_hexstring(char * s, double x);

Quiet-Computational Operations Functions
Many routines in the libbfp754 Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.
The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for quiet-computational operations:

copy
Description: The function copies input floating-point number $x$ to output in the same floating-point format, without any change to the sign.

Calling interface:
float __binary32_copy(float x);
double __binary64_copy(double x);

NOTE
When the input is a signaling NaN, two different outcomes are allowed by the standard. The operation could either signal invalid exception with quieted signaling NaN as output, or deliver signaling NaN as output without signaling any exception.

negate
Description: The function copies input floating-point number $x$ to output in the same floating-point format, reversing the sign.

Calling interface:
float __binary32_negate(float x);
double __binary64_negate(double x);

NOTE
When the input is a signaling NaN, two different outcomes are allowed by the standard. The operation could either signal invalid exception with quieted signaling NaN as output, or deliver signaling NaN as output without signaling any exception.

abs
Description: The function copies input floating-point number $x$ to output in the same floating-point format, setting the sign to positive.

Calling interface:
float __binary32_abs(float x);
double __binary64_abs(double x);

NOTE
When the input is a signaling NaN, two different outcomes are allowed by the standard. The operation could either signal invalid exception with quieted signaling NaN as output, or deliver signaling NaN as output without signaling any exception.

**copysign**

**Description:** The function copies input floating-point number \( x \) to output in the same floating-point format, with the same sign as \( y \).

**Calling interface:**
float __binary32_copysign(float x, float y);
double __binary64_copysign(double x, double y);

NOTE
When the first input is a signaling NaN, two different outcomes are allowed by the standard. The operation could either signal invalid exception with quieted signaling NaN as output, or deliver signaling NaN as output without signaling any exception.

**Signaling-Computational Operations Functions**

Many routines in the *libbfp754* Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for signaling-computational operations:

**quiet_equal**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is equal, returns 0 (false) otherwise. The function signals invalid operation exception when signaling NaN is in the inputs.

**Calling interface:**
int __binary32_quiet_equal_binary32 (float x, float y);
int __binary32_quiet_equal_binary64(float x, double y);
int __binary64_quiet_equal_binary32(double x, float y);
int __binary64_quiet_equal_binary64(double x, double y);

**quiet_not_equal**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is not equal, returns 0 (false) otherwise. The function signals invalid operation exception when signaling NaN is one of the inputs.

**Calling interface:**
int __binary32_quiet_not_equal_binary32 (float x, float y);
int __binary32_quiet_not_equal_binary64(float x, double y);
int __binary64_quiet_not_equal_binary32(double x, float y);
int __binary64_quiet_not_equal_binary64(double x, double y);
**signaling_equal**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is equal, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**
```c
int __binary32_signaling_equal_binary32(float x, float y);
int __binary32_signaling_equal_binary64(float x, double y);
int __binary64_signaling_equal_binary32(double x, float y);
int __binary64_signaling_equal_binary64(double x, double y);
```

**signaling_greater**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is greater, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**
```c
int __binary32_signaling_greater_binary32(float x, float y);
int __binary32_signaling_greater_binary64(float x, double y);
int __binary64_signaling_greater_binary32(double x, float y);
int __binary64_signaling_greater_binary64(double x, double y);
```

**signaling_greater_equal**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is greater or equal, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**
```c
int __binary32_signaling_greater_equal_binary32(float x, float y);
int __binary32_signaling_greater_equal_binary64(float x, double y);
int __binary64_signaling_greater_equal_binary32(double x, float y);
int __binary64_signaling_greater_equal_binary64(double x, double y);
```

**signaling_less**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is less, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**
```c
int __binary32_signaling_less_binary32(float x, float y);
int __binary32_signaling_less_binary64(float x, double y);
int __binary64_signaling_less_binary32(double x, float y);
int __binary64_signaling_less_binary64(double x, double y);
```

**signaling_less_equal**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is less or equal, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**
```c
int __binary32_signaling_less_equal_binary32(float x, float y);
int __binary32_signaling_less_equal_binary64(float x, double y);
int __binary64_signaling_less_equal_binary32(double x, float y);
int __binary64_signaling_less_equal_binary64(double x, double y);
```
**signaling_not_equal**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is not equal, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**

```c
int __binary32_signaling_not_equal_binary32(float x, float y);
int __binary32_signaling_not_equal_binary64(float x, double y);
int __binary64_signaling_not_equal_binary32(double x, float y);
int __binary64_signaling_not_equal_binary64(double x, double y);
```

**signaling_not_greater**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is not greater, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**

```c
int __binary32_signaling_not_greater_binary32(float x, float y);
int __binary32_signaling_not_greater_binary64(float x, double y);
int __binary64_signaling_not_greater_binary32(double x, float y);
int __binary64_signaling_not_greater_binary64(double x, double y);
```

**signaling_less_unordered**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is less or unordered, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**

```c
int __binary32_signaling_less_unordered_binary32(float x, float y);
int __binary32_signaling_less_unordered_binary64(float x, double y);
int __binary64_signaling_less_unordered_binary32(double x, float y);
int __binary64_signaling_less_unordered_binary64(double x, double y);
```

**signaling_not_less**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is not less, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**

```c
int __binary32_signaling_not_less_binary32(float x, float y);
int __binary32_signaling_not_less_binary64(float x, double y);
int __binary64_signaling_not_less_binary32(double x, float y);
int __binary64_signaling_not_less_binary64(double x, double y);
```

**signaling_greater_unordered**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is greater or unordered, returns 0 (false) otherwise. The function signals invalid operation exception when \( \text{NaN} \) is in the inputs.

**Calling interface:**

```c
int __binary32_signaling_greater_unordered_binary32(float x, float y);
int __binary32_signaling_greater_unordered_binary64(float x, double y);
int __binary64_signaling_greater_unordered_binary32(double x, float y);
int __binary64_signaling_greater_unordered_binary64(double x, double y);
```
**quiet_greater**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is greater, returns 0 (false) otherwise. The function signals invalid operation exception when signaling NaN is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_greater_binary32(float x, float y);
int __binary32_quiet_greater_binary64(float x, double y);
int __binary64_quiet_greater_binary32(double x, float y);
int __binary64_quiet_greater_binary64(double x, double y);
```

**quiet_greater_equal**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is greater or equal, returns 0 (false) otherwise. The function signals invalid operation exception when signaling NaN is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_greater_equal_binary32(float x, float y);
int __binary32_quiet_greater_equal_binary64(float x, double y);
int __binary64_quiet_greater_equal_binary32(double x, float y);
int __binary64_quiet_greater_equal_binary64(double x, double y);
```

**quiet_less**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is less, returns 0 (false) otherwise. The function signals invalid operation exception when signaling NaN is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_less_binary32(float x, float y);
int __binary32_quiet_less_binary64(float x, double y);
int __binary64_quiet_less_binary32(double x, float y);
int __binary64_quiet_less_binary64(double x, double y);
```

**quiet_less_equal**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is less or equal, returns 0 (false) otherwise. The function signals invalid operation exception when signaling NaN is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_less_equal_binary32(float x, float y);
int __binary32_quiet_less_equal_binary64(float x, double y);
int __binary64_quiet_less_equal_binary32(double x, float y);
int __binary64_quiet_less_equal_binary64(double x, double y);
```

**quiet_unordered**

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is unordered, returns zero (false) otherwise. The function signals invalid operation exception when signaling NaN is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_unordered_binary32(float x, float y);
int __binary32_quiet_unordered_binary64(float x, double y);
int __binary64_quiet_unordered_binary32(double x, float y);
int __binary64_quiet_unordered_binary64(double x, double y);
```
quiet_not_greater

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is not greater, returns zero (false) otherwise. The function signals invalid operation exception when signaling \( \text{NaN} \) is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_not_greater_binary32(float x, float y);
int __binary32_quiet_not_greater_binary64(float x, double y);
int __binary64_quiet_not_greater_binary32(double x, float y);
int __binary64_quiet_not_greater_binary64(double x, double y);
```

quiet_less_unordered

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is less or unordered, returns 0 (false) otherwise. The function signals invalid operation exception when signaling \( \text{NaN} \) is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_less_unordered_binary32(float x, float y);
int __binary32_quiet_less_unordered_binary64(float x, double y);
int __binary64_quiet_less_unordered_binary32(double x, float y);
int __binary64_quiet_less_unordered_binary64(double x, double y);
```

quiet_not_less

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is not less, returns zero (false) otherwise. The function signals invalid operation exception when signaling \( \text{NaN} \) is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_not_less_binary32(float x, float y);
int __binary32_quiet_not_less_binary64(float x, double y);
int __binary64_quiet_not_less_binary32(double x, float y);
int __binary64_quiet_not_less_binary64(double x, double y);
```

quiet_greater_unordered

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is greater or unordered, returns 0 (false) otherwise. The function signals invalid operation exception when signaling \( \text{NaN} \) is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_greater_unordered_binary32(float x, float y);
int __binary32_quiet_greater_unordered_binary64(float x, double y);
int __binary64_quiet_greater_unordered_binary32(double x, float y);
int __binary64_quiet_greater_unordered_binary64(double x, double y);
```

quiet_ordered

**Description:** The function returns 1 (true) if the relation between the two inputs \( x \) and \( y \) is ordered, returns 0 (false) otherwise. The function signals invalid operation exception when signaling \( \text{NaN} \) is one of the inputs.

**Calling interface:**

```c
int __binary32_quiet_ordered_binary32(float x, float y);
int __binary32_quiet_ordered_binary64(float x, double y);
int __binary64_quiet_ordered_binary32(double x, float y);
```
int __binary64_quiet_ordered_binary64(double x, double y);

Non-Computational Operations Functions

Many routines in the libbfp754 Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The Intel® IEEE 754-2008 Binary Conformance Library supports the following functions for non-computational operations:

is754version1985

**Description:** The function returns 1, if and only if this programming environment conforms to IEEE Std. 754-1985, otherwise returns 0.

**Calling interface:**

```
int __binary_is754version1985(void);
```

**NOTE**

This function in this library always returns 0.

is754version2008

**Description:** The function returns 1, if and only if this programming environment conforms to IEEE Std. 754-2008, otherwise returns 0.

**Calling interface:**

```
int __binary_is754version2008(void);
```

**NOTE**

This function in this library always returns 1.

class

**Description:** The function returns which class of the ten classes (signalingNaN, quietNaN, negativeInfinity, negativeNormal, negativeSubnormal, negativeZero, positiveZero, positiveSubnormal, positiveNormal, positiveInfinity) the input floating-point number x belongs.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>signalingNaN</td>
</tr>
<tr>
<td>1</td>
<td>quietNaN</td>
</tr>
<tr>
<td>2</td>
<td>negativeInfinity</td>
</tr>
<tr>
<td>3</td>
<td>negativeNormal</td>
</tr>
<tr>
<td>4</td>
<td>negativeSubnormal</td>
</tr>
<tr>
<td>5</td>
<td>negativeZero</td>
</tr>
<tr>
<td>6</td>
<td>positiveZero</td>
</tr>
<tr>
<td>7</td>
<td>positiveSubnormal</td>
</tr>
<tr>
<td>Return value</td>
<td>Class</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>8</td>
<td>positiveNormal</td>
</tr>
<tr>
<td>9</td>
<td>positiveInfinity</td>
</tr>
</tbody>
</table>

**Calling interface:**
```
int __binary32_class(float x);
int __binary64_class(double x);
```

**isSignMinus**

**Description:** The function returns 1, if and only if its argument has negative sign.

**Calling interface:**
```
int __binary32_isSignMinus(float x);
int __binary64_isSignMinus(double x);
```

**isNormal**

**Description:** The function returns 1, if and only if its argument is normal (not zero, subnormal, infinite, or NaN).

**Calling interface:**
```
int __binary32_isNormal(float x);
int __binary64_isNormal(double x);
```

**isFinite**

**Description:** The function returns 1, if and only if its argument is finite (not infinite or NaN).

**Calling interface:**
```
```

**isZero**

**Description:** The function returns 1, if and only if its argument is ±0.

**Calling interface:**
```
int __binary32_isZero(float x);
int __binary64_isZero(double x);
```

**isSubnormal**

**Description:** The function returns 1, if and only if its argument is subnormal.

**Calling interface:**
```
int __binary32_isSubnormal(float x);
int __binary64_isSubnormal(double x);
```

**isInfinite**

**Description:** The function returns 1, if and only if its argument is infinite

**Calling interface:**
```
int __binary32_isInfinite(float x);
int __binary64_isInfinite(double x);
```
isNaN
Description: The function returns 1, if and only if its argument is a NaN.

Calling interface:
```c
int __binary32_isNaN(float x);
int __binary64_isNaN(double x);
```

isSignaling
Description: The function returns 1, if and only if its argument is a signaling NaN.

Calling interface:
```c
int __binary32_isSignaling(float x);
int __binary64_isSignaling(double x);
```

isCanonical
Description: The function returns 1, if and only if its argument is a finite number, infinity, or NaN that is canonical.

Calling interface:
```c
int __binary32_isCanonical(float x);
int __binary64_isCanonical(double x);
```

NOTE
This function in this library always returns 1, as only canonical floating-point numbers are expected.

radix
Description: The function returns the radix of the format of the input floating-point number.

Calling interface:
```c
int __binary32_radix(float x);
int __binary64_radix(double x);
```

NOTE
This function in this library always returns 2, as the library is intended for binary floating-point numbers.

totalOrder
Description: The function returns 1 if and only if two floating-point inputs \( x \) and \( y \) is total ordered and 0 otherwise.

Calling interface:
```c
int _binary32_totalOrder(float x, float y);
int _binary64_totalOrder(double x, double y);
```

totalOrderMag
Description: \( \text{totalOrderMag}(x, y) \) is the same as \( \text{totalOrder}(|x|, |y|) \).

Calling interface:
```c
int _binary32_totalOrderMag(float x, float y);
int _binary64_totalOrderMag(double x, double y);
```
**lowerFlags**

**Description:** The function lowers the flags of the exception group specified by the input.

<table>
<thead>
<tr>
<th>Value</th>
<th>Exception name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>_BFP754_INVALID</td>
</tr>
<tr>
<td>2</td>
<td>_BFP754_DIVBYZERO</td>
</tr>
<tr>
<td>4</td>
<td>_BFP754_OVERFLOW</td>
</tr>
<tr>
<td>8</td>
<td>_BFP754_UNDERFLOW</td>
</tr>
<tr>
<td>16</td>
<td>_BFP754_INEXACT</td>
</tr>
</tbody>
</table>

**Calling interface:**

void __binary_lowerFlags(int x);

**raiseFlags**

**Description:** The function raises the flags of the exception group specified by the input.

**Calling interface:**

void __binary_raiseFlags(int x);

**testFlags**

**Description:** The function returns 1, if and only if any flag of the exception group specified by the input is raised, and 0 otherwise.

**Calling interface:**

int __binary_testFlags(int x);

**testSavedFlags**

**Description:** The function returns 1, if and only if any flag of the exception group specified by the input y is raised in x, and 0 otherwise.

**Calling interface:**

int __binary_testSavedFlags(int x, int y);

**restoreFlags**

**Description:** The function restores the flags to their states represented in x.

**Calling interface:**

void __binary_restoreFlags(int x);

**saveFlags**

**Description:** The function returns a representation of the state of all status flags.

**Calling interface:**

int __binary_saveFlags(void);

**getBinaryRoundingDirection**

**Description:** The function returns an integer representing the rounding direction in use.
<table>
<thead>
<tr>
<th>Value</th>
<th>Exception name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>__BFP754_ROUND_TO_NEAREST_EVEN</td>
</tr>
<tr>
<td>1</td>
<td>__BFP754_ROUND_TOWARD_POSITIVE</td>
</tr>
<tr>
<td>2</td>
<td>__BFP754_ROUND_TOWARD_NEGATIVE</td>
</tr>
<tr>
<td>3</td>
<td>__BFP754_ROUND_TOWARD_ZERO</td>
</tr>
</tbody>
</table>

**Calling interface:**

```c
int __binary_getBinaryRoundingDirection(void);
```

**setBinaryRoundingDirection**

**Description:** The function sets the rounding direction based on input integer.

**Calling interface:**

```c
void __binary_setBinaryRoundingDirection(int x);
```

**saveModes**

**Description:** The function saves the values of all dynamic-specifiable modes.

**Calling interface:**

```c
int __binary_saveModes(void);
```

**NOTE**

*saveModes* behaves in the same way as *getBinaryRoundingDirection* does, as the rounding mode is the only dynamic-specifiable mode supported.

**restoreModes**

**Description:** The function restores the values of all dynamic-specifiable modes to the input.

**Calling interface:**

```c
int __binary_restoreModes(void);
```

**NOTE**

*restoreModes* behaves in the same way as *setBinaryRoundingDirection* does, as the rounding mode is the only dynamic-specifiable mode supported.

**defaultMode**

**Description:** The function sets the values of all dynamic-specifiable modes to default.

**Calling interface:**

```c
void __binary_defaultMode(void);
```

**NOTE**

*defaultMode* sets the rounding-direction attribute to *roundTiesToEven*, as the rounding mode is the only dynamic-specifiable mode supported.
Intel's Numeric String Conversion Library

Intel's Numeric String Conversion Library, libistrconv, provides a collection of routines for converting between ASCII strings and C data types, which are optimized for performance.

Product and Performance Information

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.
Notice revision #20201201

Overview: Intel's Numeric String Conversion Library

Intel's Numeric String Conversion Library, libistrconv, provides a collection of routines for converting between ASCII strings and C data types, which are optimized for performance. The istrconv.h header file declares prototypes for the library functions.

You can link the libistrconv library as a static or shared library on Linux* platforms. On Windows* platforms, you must link libistrconv as a static library only.

Using Intel's Numeric String Conversion Library

To use the libistrconv library, include the header file, istrconv.h, in your program.

Consider the following example conv.c file that illustrates how to use the library to convert between string and floating-point data type.

```c
// conv.c
#include <stdio.h>
#include <istrconv.h>
#define LENGTH 20

int main() {
    const char pi[] = "3.14159265358979323";
    char s[LENGTH];
    int prec;
    float fx;
    double dx;
    printf("PI: %s\n", pi);
    printf("single-precision\n");
    fx = __IML_string_to_float(pi, NULL);
    prec = 6;
    __IML_float_to_string(s, LENGTH, prec, fx);
    printf("prec: %2d, val: %s\n", prec, s);
    printf("double-precision\n");
    dx = __IML_string_to_double(pi, NULL);
    prec = 15;
    __IML_double_to_string(s, LENGTH, prec, dx);
    printf("prec: %2d, val: %s\n", prec, s);
    return 0;
}
```

To compile the conv.c file on Linux* platforms, use one of the following commands:

For C++:

icx
For DPC++:

dpcpp conv.c -libistrconv

To compile the conv.c file on Windows* platforms, use the following command:

For C++:

icx

For DPC++:

dpcpp conv.c libistrconv.lib

After you compile this example and run the program, you should get the following results:

PI: 3.14159265358979323

single-precision
prec: 6, val: 3.14159

double-precision
prec: 15, val: 3.14159265358979

**Integer Conversion Functions Optimized with SSE4.2 Instructions**

The following integer conversion functions are optimized for better performance with SSE4.2 string processing instructions:

__IML_int_to_string
__IML_uint_to_string
__IML_int64_to_string
__IML_uint64_to_string
__IML_i_to_str
__IML_u_to_str
__IML_ll_to_str
__IML_ull_to_str
__IML_string_to_int
__IML_string_to_uint
__IML_string_to_int64
__IML_string_to_uint64
__IML_str_to_i
__IML_str_to_u
__IML_str_to_ll
__IML_str_to_ull

The SSE4.2 optimized versions of these functions can be deployed in the following situations:

- Used automatically on post-SSE4.2 processors through Intel run-time processor dispatching
- Called directly by defining the "__SSE4_2__" macro to the C preprocessor where <istrconv.h> is included.

The generic versions of these functions can be deployed in the following situations:

- Used automatically on pre-SSE4.2 processors through Intel run-time processor dispatching
- Called directly by adding _generic suffix to the function names
The SSE4.2 optimized versions of these functions moves strings from memory to XMM registers and vice versa directly to maximize performance. The functions would not overwrite the memory beyond the boundary; however, this may introduce memory access violation when the memory location immediately trailing the strings is not allocated or accessible. Users with concerns about potential memory access violation should use the generic versions instead.

**Function List**

Intel's Numeric String Conversion library (libistrconv) functions are listed in this topic.

**Routines to convert floating-point numbers to ASCII strings**

Intel's Numeric String Conversion Library supports the following functions to convert floating-point number x to string s in various formats, where l represents the length of the formatted string allowing for full conversion (not including the null terminator).

__IML_float_to_string__, __IML_double_to_string__

**Description:** These functions are similar to `snprintf(s, n, "%.p", p, x)` in stdio.h, where `p` specifies the maximum number of significant digits in either fixed-point or exponential notation format. If `n` is zero, nothing is written and `s` may be a null pointer. Output characters beyond the `(n-1)th` character are discarded and a null character is appended at the end. `l` is returned on success; otherwise the result is undefined.

**Calling interface:**

```c
int __IML_float_to_string(char * s, size_t n, int p, float x);
int __IML_double_to_string(char * s, size_t n, int p, double x);
```

__IML_float_to_string_f__, __IML_double_to_string_f__

**Description:** These functions are similar to `snprintf(s, n, "%.f", p, x)` in stdio.h, where `p` specifies the number of digits after the decimal point in the fixed-point notation format. If `n` is zero, nothing is written and `s` may be a null pointer. Output characters beyond the `(n-1)th` character are discarded and a null character is appended at the end. `l` is returned on success; otherwise the result is undefined.

**Calling interface:**

```c
int __IML_float_to_string_f(char * s, size_t n, int p, float x);
int __IML_double_to_string_f(char * s, size_t n, int p, double x);
```

__IML_float_to_string_e__, __IML_double_to_string_e__

**Description:** These functions are similar to `snprintf(s, n, "%.e", p, x)` in stdio.h, where `p` specifies the number of digits after the decimal point in the exponential notation format. If `n` is zero, nothing is written and `s` may be a null pointer. Output characters beyond the `(n-1)th` character are discarded and a null character is appended at the end. `l` is returned on success; otherwise, the result is undefined.

**Calling interface:**

```c
int __IML_float_to_string_e(char * s, size_t n, int p, float x);
int __IML_double_to_string_e(char * s, size_t n, int p, double x);
```

__IML_f_to_str__, __IML_d_to_str__

**Description:** These functions are similar to `snprintf(s, n, "%.g", p, x)` in stdio.h, where `p` specifies the maximum number of significant digits in either fixed-point or exponential notation format. If `l < n`, all output characters are stored in `s` with a null terminator at the end. Otherwise, output characters beyond the `n`th character are discarded and no null character is appended at the end. If `n` is zero, nothing is written and `s` may be a null pointer. `l` is returned on success; otherwise the result is undefined.
Calling interface:

```c
int __IML_f_to_str(char * s, size_t n, int p, float x);
int __IML_d_to_str(char * s, size_t n, int p, double x);
```

**Description:** These functions are similar to `snprintf(s, n, "%.f", p, x)` in `stdio.h`, where `p` specifies the number of digits after the decimal point in the fixed-point notation format. If `l < n`, all output characters are stored in `s` with a null terminator at the end. Otherwise, output characters beyond the `n`th character are discarded and no null character is appended at the end. If `n` is zero, nothing is written and `s` may be a null pointer. `l` is returned on success; otherwise the result is undefined.

Calling interface:

```c
int __IML_f_to_str_f(char * s, size_t n, int p, float x);
int __IML_d_to_str_f(char * s, size_t n, int p, double x);
```

**Description:** These functions are similar to `snprintf(s, n, "%.f", p, x)` in `stdio.h`, where `p` specifies the number of digits after the decimal point in the exponential notation format. If `l < n`, all output characters are stored in `s` with a null terminator at the end. Otherwise, output characters beyond the `n`th character are discarded and no null character is appended at the end. If `n` is zero, nothing is written and `s` may be a null pointer. `l` is returned on success; otherwise the result is undefined.

**Routines to convert integers to ASCII strings**

Intel's Numeric String Conversion Library supports the following functions to convert integer `x` to string `s`, where `l` represents the length of the formatted string allowing for full conversion (not including the null terminator).

```c
__IML_int_to_string, __IML_uint_to_string, __IML_int64_to_string, __IML_uint64_to_string
```

**Description:** These functions are similar to `snprintf(s, n, "%[d|u|lld|llu]", x)` in `stdio.h`. If `n` is zero, nothing is written and `s` may be a null pointer. Output characters beyond the `(n-1)`th character are discarded and a null character is appended at the end. `l` is returned on success; otherwise the result is undefined.

Calling interface:

```c
int __IML_int_to_string(char * s, size_t n, int x);
int __IML_uint_to_string(char * s, size_t n, unsigned int x);
int __IML_int64_to_string(char * s, size_t n, long long x);
int __IML_uint64_to_string(char * s, size_t n, unsigned long long x);
```

**Description:** These functions are similar to `snprintf(s, n, "%[o|lll]", x)` in `stdio.h`. If `n` is zero, nothing is written and `s` may be a null pointer. Output characters beyond the `(n-1)`th character are discarded and a null character is appended at the end. `l` is returned on success; otherwise the result is undefined.

Calling interface:
int __IML_int_to_oct_string(char * s, size_t n, int x);
int __IML_uint_to_oct_string(char * s, size_t n, unsigned int x);
int __IML_int64_to_oct_string(char * s, size_t n, long long x);
int __IML_uint64_to_oct_string(char * s, size_t n, unsigned long long x);

__IML_int_to_hex_string, __IML_uint_to_hex_string, __IML_int64_to_hex_string,
__IML_uint64_to_hex_string

Description: These functions are similar to snprintf(s, n, "%[x|llx]", x) in stdio.h. If n is zero, nothing is written and s may be a null pointer. Output characters beyond the (n-1)th character are discarded and a null character is appended at the end. l is returned on success; otherwise the result is undefined.

Calling interface:
int __IML_int_to_hex_string(char * s, size_t n, int x);
int __IML_uint_to_hex_string(char * s, size_t n, unsigned int x);
int __IML_int64_to_hex_string(char * s, size_t n, long long x);
int __IML_uint64_to_hex_string(char * s, size_t n, unsigned long long x);

__IML_i_to_str, __IML_u_to_str, __IML_ll_to_str, __IML_ull_to_str

Description: These functions are similar to snprintf(s, n, "%[d|u|lld|llu]", x) in stdio.h. If l < n, all output characters are stored in s with a null terminator at the end. Otherwise, output characters beyond the nth character are discarded and no null character is appended at the end. If n is zero, nothing is written, and s may be a null pointer. l is returned on success, otherwise the result is undefined.

Calling interface:
int __IML_i_to_str(char * s, size_t n, int x);
int __IML_u_to_str(char * s, size_t n, unsigned int x);
int __IML_ll_to_str(char * s, size_t n, long long x);
int __IML_ull_to_str(char * s, size_t n, unsigned long long x);

__IML_i_to_oct_str, __IML_u_to_oct_str, __IML_ll_to_oct_str, __IML_ull_to_oct_str

Description: These functions are similar to snprintf(s, n, "%[o|llo]", x) in stdio.h. If l < n, all output characters are stored in s with a null terminator at the end. Otherwise, output characters beyond the nth character are discarded and no null character is appended at the end. If n is zero, nothing is written, and s may be a null pointer. l is returned on success, otherwise the result is undefined.

Calling interface:
int __IML_i_to_oct_str(char * s, size_t n, int x);
int __IML_u_to_oct_str(char * s, size_t n, unsigned int x);
int __IML_ll_to_oct_str(char * s, size_t n, long long x);
int __IML_ull_to_oct_str(char * s, size_t n, unsigned long long x);

__IML_i_to_hex_str, __IML_u_to_hex_str, __IML_ll_to_hex_str, __IML_ull_to_hex_str

Description: These functions are similar to snprintf(s, n, "%[x|llx]", x) in stdio.h. If l < n, all output characters are stored in s with a null terminator at the end. Otherwise, output characters beyond the nth character are discarded and no null character is appended at the end. If n is zero, nothing is written, and s may be a null pointer. l is returned on success, otherwise the result is undefined.
Calling interface:

```c
int __IML_i_to_hex_str(char * s, size_t n, int x);
int __IML_u_to_hex_str(char * s, size_t n, unsigned int x);
int __IML_ll_to_hex_str(char * s, size_t n, long long x);
int __IML_ull_to_hex_str(char * s, size_t n, unsigned long long x);
```

Routines to convert ASCII strings to floating-point numbers

Intel’s Numeric String Conversion Library supports the following functions to convert the initial portion of decimal string `s` to floating-point number `x`. If no conversion could be performed, zero is returned. If the correct value is outside the range of the return type, plus (+) or minus (-) `HUGE_VALF`, `HUGE_VAL`, or `HUGE_VALL` is returned, and the value of macro `ERANGE` is stored in `errno`.

```c
__IML_string_to_float, __IML_string_to_double, __IML_string_to_long_double
```

**Description:** These functions are similar to `strtof(nptr, endptr)`, `strtod(nptr, endptr)`, and `strtold(nptr, endptr)` in `stdlib.h`, where `endptr` points to the object that stores the final part of `nptr` when `endptr` is not a null pointer.

Calling interface:

```c
float __IML_string_to_float(const char * nptr, char ** endptr);
double __IML_string_to_double(const char * nptr, char ** endptr);
long double __IML_string_to_long_double(const char * nptr, char ** endptr);
```

```c
__IML_str_to_f, __IML_str_to_d, __IML_str_to_ld
```

**Description:** These functions convert the initial `n` decimal digits of the `significand` string multiplied by 10 raised to power of `exponent` to floating-point number as return. `endptr` points to the object that stores the final part of significand, provided that `endptr` is not a null pointer.

Calling interface:

```c
float __IML_str_to_f(const char * significand, size_t n, int exponent, char ** endptr);
double __IML_str_to_d(const char * significand, size_t n, int exponent, char ** endptr);
long double __IML_str_to_ld(const char * significand, size_t n, int exponent, char ** endptr);
```

Routines to convert ASCII strings to integers

Intel’s Numeric String Conversion Library supports the following functions to convert the initial portion of string `s` to integer `x`. If no conversion could be performed, zero is returned. If the correct value is outside the range of the return type, `INT_MIN`, `INT_MAX`, `UINT_MAX`, `LLONG_MIN`, `LLONG_MAX`, `ULLONG_MAX` is returned, and the value of macro `ERANGE` is stored in `errno`.

```c
__IML_string_to_int, __IML_string_to_uint, __IML_string_to_int64, __IML_string_to_uint64
```

**Description:** These functions are similar to `((unsigned) int)strtol(nptr, endptr, 10)` and `strtoull(nptr, endptr, 10)` functions in `stdlib.h`, where `endptr` points to the object that stores the final part of `nptr` when `endptr` is not a null pointer.

Calling interface:

```c
int __IML_string_to_int(const char * nptr, char ** endptr);
unsigned int __IML_string_to_uint(const char * nptr, char ** endptr);
```
Description: These functions are similar to [(unsigned int)strto[u]l(nptr, endptr, 8) and strto[u]ll(nptr, endptr, 8)] functions in stdlib.h, where endptr points to the object that stores the final part of nptr when endptr is not a null pointer.

Calling interface:

int __IML_oct_string_to_int(const char * nptr, char ** endptr);
unsigned int __IML_oct_string_to_uint(const char * nptr, char ** endptr);
long long __IML_oct_string_to_int64(const char * nptr, char ** endptr);
unsigned long long __IML_oct_string_to_uint64(const char * nptr, char ** endptr);

Description: These functions convert the initial n decimal digits (including an optional + or - sign) pointed to by nptr to integral values. When endptr is not a null pointer it points to the object that stores the final part of nptr. These functions treat any leading whitespace as invalid.

Calling interface:

int __IML_str_to_i(const char * nptr, size_t n, char ** endptr);
unsigned int __IML_str_to_u(const char * nptr, size_t n, char ** endptr);
long long __IML_str_to_ll(const char * nptr, size_t n, char ** endptr);
unsigned long long __IML_str_to_ull(const char * nptr, size_t n, char ** endptr);

Description: These functions convert the initial n octal digits (including an optional + or - sign) pointed to by nptr to integral values. When endptr is not a null pointer it points to the object that stores the final part of nptr. These functions treat any leading whitespace as invalid.

Calling interface:

int __IML_oct_str_to_i(const char * nptr, size_t n, char ** endptr);
unsigned int __IML_oct_str_to_u(const char * nptr, size_t n, char ** endptr);
long long __IML_oct_str_to_ll(const char * nptr, size_t n, char ** endptr);
unsigned long long __IML_oct_str_to_ull(const char * nptr, size_t n, char ** endptr);

__IML_hex_str_to_i, __IML_hex_str_to_u, __IML_hex_str_to_ll, __IML_hex_str_to_ull

**Description:** These functions convert the initial $n$ hexadecimal digits (including an optional + or - sign) pointed to by `nptr` to integral values. When `endptr` is not a null pointer it points to the object that stores the final part of `nptr`. These functions treat any leading whitespace as invalid.

**Calling interface:**

```c
int __IML_hex_str_to_i(const char * nptr, size_t n, char ** endptr);
```

```c
unsigned int __IML_hex_str_to_u(const char * nptr, size_t n, char ** endptr);
```

```c
long long __IML_hex_str_to_ll(const char * nptr, size_t n, char ** endptr);
```

```c
unsigned long long __IML_hex_str_to_ull(const char * nptr, size_t n, char ** endptr);
```

---

**Macros**

The Intel® oneAPI DPC++/C++ Compiler supports the ISO Standard predefined macros, as well as additional predefined macros.

**ISO Standard Predefined Macros**

The ISO/ANSI standard for the C language requires that certain predefined macros be supplied with conforming compilers.

The compiler includes predefined macros in addition to those required by the standard. The default predefined macros differ among Windows*, Linux* operating systems. Differences also exist on Linux as a result of the `-std` compiler option.

The following table lists the macros that the Intel® oneAPI DPC++/C++ Compiler supplies in accordance with this standard:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DATE</strong></td>
<td>The date of compilation as an 11-character string literal in the form mm dd yyyy. If the day is less than 10 characters, a space is added before the day value.</td>
</tr>
<tr>
<td><strong>FILE</strong></td>
<td>A string literal representing the name of the file being compiled.</td>
</tr>
<tr>
<td><strong>LINE</strong></td>
<td>The current line number as a decimal constant.</td>
</tr>
<tr>
<td><strong>STDC_HOSTED</strong></td>
<td>Defined and value is 1 only when compiling a C translation unit with /Qstd=c99.</td>
</tr>
<tr>
<td><strong>STDC_VERSION</strong></td>
<td>Defined and value is 199901L only when compiling a C translation unit with /Qstd=c99.</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>The time of compilation as a string literal in the form hh:mm:ss.</td>
</tr>
</tbody>
</table>

**See Also**

Additional Predefined Macros

**Additional Predefined Macros**

The compiler supports the predefined macros listed in the table below. The compiler also includes predefined macros specified by the ISO/ANSI standard.
Unless otherwise stated, the macros are supported on systems based on IA-32 (for C++ only) and Intel® 64 architectures.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AVX</strong></td>
<td>On Linux*, defined as '1' when option -march=corei7-avx, or higher processor targeting options are specified.</td>
</tr>
<tr>
<td><strong>AVX2</strong></td>
<td>On Linux, defined as '1' when option -march=core-avx2, or higher processor targeting options are specified.</td>
</tr>
<tr>
<td><strong>AVX512BW</strong></td>
<td>Defined as '1' for processors that support Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Byte and Word Instructions (BWI).</td>
</tr>
<tr>
<td><strong>AVX512CD</strong></td>
<td>Defined as '1' for processors that support Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Conflict Detection Instructions (CDI).</td>
</tr>
<tr>
<td><strong>AVX512DQ</strong></td>
<td>Defined as '1' for processors that support Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Doubleword and Quadword Instructions (DQI).</td>
</tr>
<tr>
<td><strong>AVX512ER</strong></td>
<td>Defined as '1' for processors that support Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Exponential and Reciprocal Instructions.</td>
</tr>
<tr>
<td><strong>AVX512F</strong></td>
<td>Defined as '1' for processors that support Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Foundation instructions.</td>
</tr>
<tr>
<td><strong>AVX512PF</strong></td>
<td>Defined as '1' for processors that support Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Pre Fetch Instructions (PFI).</td>
</tr>
<tr>
<td><strong>AVX512VL</strong></td>
<td>Defined as '1' for processors that support Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Vector Length Extensions.</td>
</tr>
<tr>
<td><strong>BASE_FILE</strong></td>
<td>Name of source file</td>
</tr>
<tr>
<td><strong>COUNTER</strong></td>
<td>Defined as '0'.</td>
</tr>
<tr>
<td><strong>cplusplus</strong></td>
<td>Defined as '1' (for the Intel® oneAPI DPC++/C++ Compiler).</td>
</tr>
<tr>
<td><strong>ELF</strong></td>
<td>Defined as '1' at the start of compilation.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>(Linux) __EXCEPTIONS</td>
<td>Defined as '1' when option fno-exceptions is not used.</td>
</tr>
<tr>
<td>(Linux) <strong>gnu_linux</strong></td>
<td>Defined as '1' at the start of compilation.</td>
</tr>
<tr>
<td>(Linux) <strong>GNUC</strong></td>
<td>The major version number of GCC installed on the system.</td>
</tr>
<tr>
<td>(Linux) <strong>GNUC_MINOR</strong></td>
<td>The minor version number of GCC or g++* installed on the system.</td>
</tr>
<tr>
<td>(Linux) <strong>GNUC_PATCHLEVEL</strong></td>
<td>The patch level version number of GCC or G++ installed on the system.</td>
</tr>
<tr>
<td>(Linux) <strong>GNUG</strong></td>
<td>The major version number of G++ installed on the system.</td>
</tr>
<tr>
<td>(Linux) <strong>i386</strong></td>
<td>Defined as '1' for compilations targeting IA-32 architecture (C++ only).</td>
</tr>
<tr>
<td>(Linux) __INTEGRAL_MAX_BITS</td>
<td>64</td>
</tr>
<tr>
<td>(Windows) __INTEL_LLVM_COMPILER</td>
<td>The version of the compiler in the form VVVVMMUUMU , where VVVV is the major release version, MM is the minor release version, and UU is the update number. For example, the base release of 2021.1 is represented by the value 20210100. This symbol is also recognized by CMake*.</td>
</tr>
<tr>
<td>(Windows) __INTEL_MS_COMPAT_LEVEL</td>
<td>Defined as '1'.</td>
</tr>
<tr>
<td>(Windows) __LIBSYCL_MAJOR_VERSION</td>
<td>Used to set the DPC++ runtime library major version.</td>
</tr>
<tr>
<td>(Windows) __LIBSYCL_MINOR_VERSION</td>
<td>Used to set the DPC++ runtime library minor version.</td>
</tr>
<tr>
<td>(Windows) __LIBSYCL_PATCH_VERSION</td>
<td>Used to set the DPC++ runtime library patch version.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>linux</strong></td>
<td>Defined as '1' at the start of compilation.</td>
</tr>
<tr>
<td>__linux</td>
<td></td>
</tr>
<tr>
<td>linux</td>
<td>(Linux)</td>
</tr>
<tr>
<td><strong>LONG_DOUBLE_SIZE</strong></td>
<td>On Linux, defined as 80.</td>
</tr>
<tr>
<td>(Windows*, Linux)</td>
<td>On Windows, defined as 64; defined as 80 when option /Qlong-double is specified.</td>
</tr>
<tr>
<td><strong>LONG_MAX</strong></td>
<td>9223372036854775807L</td>
</tr>
<tr>
<td>(Linux)</td>
<td></td>
</tr>
<tr>
<td><strong>LP64</strong></td>
<td>Defined as '1'.</td>
</tr>
<tr>
<td>(Linux)</td>
<td></td>
</tr>
<tr>
<td>__LP64</td>
<td></td>
</tr>
<tr>
<td>_M_IX86</td>
<td>700</td>
</tr>
<tr>
<td>(Windows)</td>
<td>Defined as '1' while building code targeting Intel® 64 architecture.</td>
</tr>
<tr>
<td>_M_X64</td>
<td>Defined as '1'.</td>
</tr>
<tr>
<td>(Windows)</td>
<td></td>
</tr>
<tr>
<td><em>MMX</em>_</td>
<td>On Linux, it is available only on systems based on Intel® 64 architecture.</td>
</tr>
<tr>
<td>(Linux)</td>
<td></td>
</tr>
<tr>
<td>_MSC_EXTENSIONS</td>
<td>This macro is defined when Microsoft extensions are enabled.</td>
</tr>
<tr>
<td>(Windows)</td>
<td></td>
</tr>
<tr>
<td>_MSC_FULL_VER</td>
<td>The Visual C++* version being used.</td>
</tr>
<tr>
<td>(Windows)</td>
<td></td>
</tr>
<tr>
<td>_MSC_VER</td>
<td>The Visual C++ version being used.</td>
</tr>
<tr>
<td>(Windows)</td>
<td></td>
</tr>
<tr>
<td>_MT</td>
<td>On Windows, defined as '1' when a multithreaded delay-locked loop (DLL) or library is used (when option /MD[d] or /MT[d] is specified).</td>
</tr>
<tr>
<td>(Windows)</td>
<td>Defined as '1'.</td>
</tr>
<tr>
<td><strong>NO_MATH_INLINE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>NO_STRING_INLINE</strong></td>
<td></td>
</tr>
<tr>
<td>(Linux)</td>
<td></td>
</tr>
<tr>
<td>_OPENMP</td>
<td>201611 when you specify option [Q]openmp.</td>
</tr>
<tr>
<td>(Windows, Linux)</td>
<td></td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>OPTIMIZE</strong></td>
<td>Defined as '1'.</td>
</tr>
<tr>
<td>__pentium4</td>
<td>Defined as '1'.</td>
</tr>
<tr>
<td><strong>pentium4</strong></td>
<td>Defined as '1'.</td>
</tr>
<tr>
<td><strong>PIC</strong></td>
<td>On Linux, defined as '1' when option fPIC is specified.</td>
</tr>
<tr>
<td><strong>pic</strong></td>
<td>On Linux, defined as '1' when option fPIC is specified.</td>
</tr>
<tr>
<td><strong>PTRDIFF_TYPE</strong></td>
<td>On Linux, defined as int on IA-32 architecture (C++ only); defined as long on Intel® 64 architecture.</td>
</tr>
<tr>
<td><strong>QMSPP</strong></td>
<td>Defined as '1'.</td>
</tr>
<tr>
<td><strong>REGISTER_PREFIX</strong></td>
<td>The specification assumes that the Data Parallel C++ (DPC++) implementation addresses space deduction. However, the deduction is performed in the middle end, where it is hard to provide user friendly diagnostics. When you write to raw pointers obtained from constant_ptr, there are no available diagnostics. You can enable diagnostics by enabling the RESTRICT_WRITE_ACCESS_TO_CONSTANT_PTR macro, which allows constant_ptr to use constant pointers as underlying pointer types. After enabling the macro, conversions from constant_ptr to raw pointers return constant pointers, and writing to const pointers is diagnosed by the front-end. This behavior does not follow the SYCL* specification, since constant_ptr conversions to the underlying pointer type will return pointers without any additional qualifiers. The macro is disabled by default.</td>
</tr>
<tr>
<td><strong>SIGNED_CHARS</strong></td>
<td>Defined as '1'.</td>
</tr>
<tr>
<td>_SIZE_T_DEFINED</td>
<td>Defined, no value.</td>
</tr>
<tr>
<td><strong>SIZE_TYPE</strong></td>
<td>On Linux, defined as unsigned on IA-32 architecture (C++ only); defined as unsigned long on Intel® 64 architecture.</td>
</tr>
<tr>
<td><strong>SSE</strong></td>
<td>On Linux, defined as '1' for processors that support SSE instructions.</td>
</tr>
<tr>
<td></td>
<td>On Windows, defined as '1'.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>SSE2</strong></td>
<td>On Linux, defined as '1' for processors that support Intel® SSE2 instructions.</td>
</tr>
<tr>
<td><strong>SSE3</strong></td>
<td>On Linux, defined as '1' for processors that support Intel® SSE3 instructions.</td>
</tr>
<tr>
<td><strong>SSE4_1</strong></td>
<td>On Linux, defined as '1' for processors that support Intel® SSE4 instructions.</td>
</tr>
<tr>
<td><strong>SSE4_2</strong></td>
<td>On Linux, defined as '1' for processors that support SSSE4 instructions.</td>
</tr>
<tr>
<td><strong>SSSE3</strong></td>
<td>On Linux, defined as '1' for processors that support SSSE3 instructions.</td>
</tr>
</tbody>
</table>
| __SYCL_COMPILER_VERSION (Windows*, Linux*) | The build date of the SYCL* library, presented in the format YYYYMMDD. 
**NOTE** This is only available after the SYCL library headers are included in the source code. |
| SYCL_DISABLE_CPP_VERSION_CHECK_WARNING (Windows, Linux) | Disables the warning displayed if the user tries to compile using an unsupported version of C++. |
| SYCL_LANGUAGE_VERSION | The SYCL_LANGUAGE_VERSION is defined only when compiling SYCL code. |
| SYCL_USE_NATIVE_FP_ATOMICS | Enable functions to generate native floating-point atomics on the target device by enabling the SYCL_USE_NATIVE_FP_ATOMICS macro. If the target device does not support floating-point atomics, emulated atomics are used instead. The macro is disabled by default. |
| unix | Defined as '1'. |
| __unix__ | Defined as '1'. |
| (Linux) | Defined, no value. |
| __USER_LABEL_PREFIX__ (Linux) | Defined, no value. |
| _VA_LIST_DEFINED (Windows) | Defined, no value. |
| __VERSION__ (Linux) | The compiler version string |
| __w64 (Windows) | Defined as '1'. |
| __WCHAR_T | Defined as '1'. |
### Use Predefined Macros for Intel® Compilers

This topic describes macros that affect the various Intel® Compilers.

When you install both the Intel® oneAPI Base Toolkit (Base Kit) and the Intel® oneAPI HPC Toolkit (HPC Kit), you will notice that there are three compilers installed:

- Intel® DPC++ Compiler
- Intel® C++ Compiler
- Intel® C++ Compiler Classic

#### Predefined Macros for Each Compiler

You can use the following predefined macros to invoke a specific compiler or version of a compiler.
<table>
<thead>
<tr>
<th>Compiler</th>
<th>Predefined Macros to Differentiate from Other Compiler</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® DPC++ Compiler</td>
<td>• SYCL_LANGUAGE_VERSION</td>
<td>SYCL_LANGUAGE_VERSION is defined in SYCL* spec and should be defined by all SYCL compilers.</td>
</tr>
<tr>
<td></td>
<td>• __INTEL_LLVM_COMPILER</td>
<td>__INTEL_LLVM_COMPILER is used to select the compiler.</td>
</tr>
<tr>
<td></td>
<td>• __VERSION</td>
<td>__VERSION is used to select the compiler.</td>
</tr>
<tr>
<td>Intel® C++ Compiler</td>
<td>• __INTEL_LLVM_COMPILER</td>
<td>__INTEL_LLVM_COMPILER is used to select the compiler.</td>
</tr>
<tr>
<td></td>
<td>• __VERSION</td>
<td>__VERSION is used to select the compiler version.</td>
</tr>
</tbody>
</table>

**Predefined Macros for Intel® DPC++ Compiler**

See the example below using `#if defined(SYCL_LANGUAGE_VERSION) && defined(__INTEL_LLVM_COMPILER)` to define a code block specific to the Intel® DPC++ Compiler:

```cpp
// dpcpp only
#if defined(SYCL_LANGUAGE_VERSION) && defined(__INTEL_LLVM_COMPILER)
    // code specific for DPC++ compiler below
    // ... ...

    // example only
    std::cout << "SYCL_LANGUAGE_VERSION: " << SYCL_LANGUAGE_VERSION << std::endl;
    std::cout << "__INTEL_LLVM_COMPILER: " << __INTEL_LLVM_COMPILER << std::endl;
    std::cout << "__VERSION__: " << __VERSION__ << std::endl;
#endif
```

Example output using the Intel® oneAPI Toolkit Gold release with an Intel DPC++ Compiler patch release of 2021.1.2:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYCL_LANGUAGE_VERSION: 202001</td>
<td>SYCL_LANGUAGE_VERSION: 202001</td>
</tr>
<tr>
<td>__INTEL_LLVM_COMPILER: 202110</td>
<td>__INTEL_LLVM_COMPILER: 202110</td>
</tr>
<tr>
<td><strong>VERSION</strong>: Intel(R) Clang Based C++, clang 12.0.0</td>
<td><strong>VERSION</strong>: Intel(R) Clang Based C++, gcc 4.2.1 mode</td>
</tr>
</tbody>
</table>

**Predefined Macros for Intel® C++ Compiler**

See the example below using `#if !defined(SYCL_LANGUAGE_VERSION) && defined(__INTEL_LLVM_COMPILER)` to define a code block specific to the Intel® C++ Compiler:

```cpp
// icx only
#if !defined(SYCL_LANGUAGE_VERSION) && defined(__INTEL_LLVM_COMPILER)
    // code specific for Intel C++ Compiler below
    // ... ...

    // example only
    std::cout << "__INTEL_LLVM_COMPILER: " << __INTEL_LLVM_COMPILER << std::endl;
    std::cout << "__VERSION__: " << __VERSION__ << std::endl;
#endif
```
Example output using the Intel® oneAPI Toolkit Gold release with an Intel C++ Compiler patch release of 2021.1.2:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__INTEL_LLVM_COMPILER: 202110</code></td>
<td><code>__INTEL_LLVM_COMPILER: 202110</code></td>
</tr>
<tr>
<td><code>__VERSION__: Intel(R) Clang Based C++, clang 12.0.0</code></td>
<td><code>__VERSION__: Intel(R) Clang Based C++, gcc 4.2.1 mode</code></td>
</tr>
</tbody>
</table>

## Pragmas

Pragmas are directives that provide instructions to the compiler for use in specific cases. For example, you can use the `novector` pragma to specify that a loop should never be vectorized. The keyword `#pragma` is standard in the C++ language, but individual pragmas are machine-specific or operating system-specific, and vary by compiler.

Some pragmas provide the same functionality as compiler options. Pragmas override behavior specified by compiler options.

Some pragmas are available for both Intel® and non-Intel microprocessors but they may perform additional optimizations for Intel® microprocessors than they perform for non-Intel microprocessors. Refer to the individual pragma name for detailed description.

The Intel® oneAPI DPC++/C++ Compiler pragmas are categorized as follows:

- **Intel-specific Pragmas** - pragmas developed or modified by Intel to work specifically with the Intel oneAPI DPC++/C++ Compiler
- **Intel Supported Pragmas** - pragmas developed by external sources that are supported by the Intel oneAPI DPC++/C++ Compiler for compatibility reasons

### Using Pragmas

You enter pragmas into your C++ source code using the following syntax:

```
#pragma <pragma name>
```

### Individual P pragma Descriptions

Each pragma description has the following details:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Description</td>
<td>Contains a brief description of what the pragma does.</td>
</tr>
<tr>
<td>Syntax</td>
<td>Contains the pragma syntax.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Contains a list of the arguments (parameters).</td>
</tr>
<tr>
<td>Description</td>
<td>Contains a detailed description of what the pragma does.</td>
</tr>
<tr>
<td>Example</td>
<td>Contains typical usage example/s.</td>
</tr>
<tr>
<td>See Also</td>
<td>Contains links or paths to other pragmas or related topics.</td>
</tr>
</tbody>
</table>

### Intel-SpecificPragma Reference

The pragmas that are specific to the Intel® oneAPI DPC++/C++ Compiler are described below. Click on each pragma name for a more detailed description.
Some pragmas are available for both Intel® microprocessors and non-Intel microprocessors, but may perform additional optimizations for Intel® microprocessors than for non-Intel microprocessors.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_section</td>
<td>Allocates one or more variables in the specified section. Controls section attribute specification for variables.</td>
</tr>
<tr>
<td>distribute_point</td>
<td>Instructs the compiler to prefer loop distribution at the location indicated.</td>
</tr>
<tr>
<td>inline</td>
<td>Specifies inlining of all calls in a statement. This also describes pragmas forceinline and noinline.</td>
</tr>
<tr>
<td>intel_omp_task</td>
<td>For Intel legacy tasking, specifies a unit of work, potentially executed by a different thread.</td>
</tr>
<tr>
<td>intel_omp_taskq</td>
<td>For Intel legacy tasking, specifies an environment for the while loop in which to queue the units of work specified by the enclosed task pragma.</td>
</tr>
<tr>
<td>ivdep</td>
<td>Instructs the compiler to ignore assumed vector dependencies.</td>
</tr>
<tr>
<td>loop_count</td>
<td>Specifies the iterations for a for loop.</td>
</tr>
<tr>
<td>nofusion</td>
<td>Prevents a loop from fusing with adjacent loops.</td>
</tr>
<tr>
<td>nvector</td>
<td>Specifies that a particular loop should never be vectorized.</td>
</tr>
<tr>
<td>optimization_level</td>
<td>Controls optimization for one function or all functions after its first occurrence.</td>
</tr>
<tr>
<td>optimization_parameter</td>
<td>Passes certain information about a function to the optimizer.</td>
</tr>
<tr>
<td>omp simd early_exit</td>
<td>Extends #pragma omp simd, allowing vectorization of multiple exit loops.</td>
</tr>
<tr>
<td>optimize</td>
<td>Enables or disables optimizations for code after this pragma until another optimize pragma or end of the translation unit.</td>
</tr>
<tr>
<td>parallel/noparallel</td>
<td>Resolves dependencies to facilitate auto-parallelization of the immediately following loop (parallel) or prevents auto-parallelization of the immediately following loop (noparallel).</td>
</tr>
<tr>
<td>simd</td>
<td>Enforces vectorization of loops.</td>
</tr>
<tr>
<td>simdoff</td>
<td>Specifies a block of code in the SIMD loop or SIMD-enabled function that should be executed serially, in a logical order of SIMD lanes.</td>
</tr>
<tr>
<td>target variant dispatch</td>
<td>Conditionally calls a procedure offload variant if the specified device is available; otherwise, executes the procedure on the host.</td>
</tr>
<tr>
<td>unroll/nounroll</td>
<td>Tells the compiler to unroll or not to unroll a counted loop.</td>
</tr>
<tr>
<td>unroll_and_jam/nounroll_and_jam</td>
<td>Enables or disables loop unrolling and jamming. These pragmas can only be applied to iterative for loops.</td>
</tr>
<tr>
<td>vector</td>
<td>Tells the compiler that the loop should be vectorized according to the argument keywords.</td>
</tr>
</tbody>
</table>

**alloc_section**

*Allocates one or more variables in the specified section. Controls section attribute specification for variables.*
Syntax

#pragma alloc_section(var1, var2, ..., "r;attribute-list")

Arguments

var

A variable that can be used to define a symbol in the section.

"r;attribute-list"

A comma-separated list of attributes; defined values are: 'short' and 'long'.

Description

The alloc_section pragma places the listed variables, var1, var2, etc., in the specified section. This pragma controls section attribute specification for variables. The compiler decides whether the variable, as defined by var1, var2, etc., should go to a "data", "bss", or "rdata" section.

The section name must be enclosed in double quotation marks. It should be previously introduced into the program using #pragma section. The list of comma-separated variable names follows the section name after a separating comma.

All listed variables must be defined before this pragma, in the same translation unit and in the same scope. The variables have static storage; their linkage does not matter in C modules, but in C++ modules they are defined with the extern "C" linkage specification.

Example: Using #pragma alloc_section

```cpp
#pragma alloc_section(var1, "r;short")
int var1 = 20;
#pragma alloc_section(var2, "r;short")
extern int var2;
```

block_loop/noblock_loop

Enables or disables loop blocking for the immediately following nested loops. block_loop enables loop blocking for the nested loops. noblock_loop disables loop blocking for the nested loops.

Syntax

#pragma block_loop [clause[,clause]...]
#pragma noblock_loop

Arguments

clause

Can be any of the following:

factor(expr)

expr is a positive scalar constant integer expression representing the blocking factor for the specified loops. This clause is optional. If the factor clause is not present, the blocking factor will be determined based on processor type and memory access patterns and will be applied to the specified levels in the nested loop following the pragma.
At most only one factor clause can appear in a block_loop pragma.

level (level_expr[, level_expr]...)  

level_expr is specified in the form const1 or const1:const2 where const1 is a positive integer constant m <= 8 representing the loop at level m, where the immediate following loop is level 1. The const2 is a positive integer constant n <= 8 representing the loop at level n, where n > m. const1:const2 represents the nested loops from level const1 through const2.

The clauses can be specified in any order. If you do not specify any clause, the compiler chooses the best blocking factor to apply to all levels of the immediately following nested loop.

Description

The block_loop pragma lets you exert greater control over optimizations on a specific loop inside a nested loop.

Using a technique called loop blocking, the block_loop pragma separates large iteration counted loops into smaller iteration groups. Execution of these smaller groups can increase the efficiency of cache space use and augment performance.

If there is no level and factor clause, the blocking factor will be determined based on the processor's type and memory access patterns and it will apply to all the levels in the nested loops following this pragma.

You can use the noblock_loop pragma to tune the performance by disabling loop blocking for nested loops.

The loop-carried dependence is ignored during the processing of block_loop pragmas.

```
#pragma block_loop factor(256) level(1)    /* applies blocking factor 256 to               */
#pragma block_loop factor(512) level(2)    /* the top level loop in the following               */
 nested loop and blocking factor 512 to the 2nd level (1st nested) loop */
#pragma block_loop factor(256) level(2)    /* levels can be specified in any order */
#pragma block_loop factor(512) level(1)    /* adjacent loops can be specified as a range */
#pragma block_loop factor(256)              /* the blocking factor applies to all levels */
 of loop nest */
#pragma block_loop /* the blocking factor will be determined based on processor type and memory access patterns and will be applied to all the levels in the nested loop following the directive */
#pragma noblock_loop /* None of the levels in the nested loop following this directive will have a blocking factor applied */
```

Consider the following:

```
#pragma block_loop factor(256) level(1:2)
for (j = 1 ; j<n ; j++){
  f = 0 ;
```
for (i = 1; i < n; i++) {
    f = f + a[i] * b[i];
}  
c[j] = c[j] + f;

The above code produces the following result after loop blocking:

for (jj = 1; jj < n/256 + 1; jj++) {
    for (ii = 1; ii < n/256 + 1; ii++) {
        for (j = (jj-1)*256 + 1; min(jj*256, n); j++) {
            f = 0;
            for (i = (ii-1)*256 + 1; i < min(ii*256, n); i++) {
                f = f + a[i] * b[i];
            }
            c[j] = c[j] + f;
        }
    }
}

code_align
Specifies the byte alignment for a loop

Syntax
#pragma code_align(n)

Arguments

n
Optional. A positive integer initialization expression indicating the number of bytes for the minimum alignment boundary. Its value must be a power of 2, between 1 and 4096, such as 1, 2, 4, 8, and so on.

If you specify 1 for n, no alignment is performed. If you do not specify n, the default alignment is 16 bytes.

Description
This pragma must precede the loop to be aligned.

If the code is compiled with the Qalign-loops:m option, and a code_align(n) pragma precedes a loop, the loop is aligned on a max (m, n) byte boundary. If a procedure has the code_align(k) attribute and a code_align(n) pragma precedes a loop, then both the procedure and the loop are aligned on a max (k, n) byte boundary.

distribute_point
Instructs the compiler to prefer loop distribution at the location indicated.

Syntax
#pragma distribute_point

Arguments

None
Description

The `distribute_point` pragma is used to suggest to the compiler to split large loops into smaller ones; this is particularly useful in cases where optimizations like vectorization cannot take place due to excessive register usage.

The following rules apply to this pragma:

- When the pragma is placed inside a loop, the compiler distributes the loop at that point. All loop-carried dependencies are ignored.
- When inside the loop, pragmas cannot be placed within an `if` statement.
- When the pragma is placed outside the loop, the compiler distributes the loop based on an internal heuristic. The compiler determines where to distribute the loops and observes data dependency. If the pragmas are placed inside the loop, the compiler supports multiple instances of the pragma.

Example: Using the `distribute_point` pragma outside the loop

```c
#define NUM 1024
void loop_distribution pragma1(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] ) {
    int i;
    // Before distribution or splitting the loop
    #pragma distribute_point
    for (i=0; i< NUM; i++) {
        a[i] = a[i] + i;
        b[i] = b[i] + i;
        c[i] = c[i] + i;
        x[i] = x[i] + i;
        y[i] = y[i] + i;
        z[i] = z[i] + i;
    }
}
```

Example: Using the `distribute_point` pragma inside the loop

```c
#define NUM 1024
void loop_distribution pragma2(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] ) {
    int i;
    // After distribution or splitting the loop.
    for (i=0; i< NUM; i++) {
        a[i] = a[i] +i;
        b[i] = b[i] +i;
        c[i] = c[i] +i;
        #pragma distribute_point
        x[i] = x[i] +i;
        y[i] = y[i] +i;
        z[i] = z[i] +i;
    }
}
Example: Using the `distribute_point` pragma inside and outside the loop

```c++
void dist1(int a[], int b[], int c[], int d[]) {
    #pragma distribute_point
    // Compiler will automatically decide where to
distribute. Data dependency is observed.
    for (int i=1; i<1000; i++) {
        b[i] = a[i] + 1;
        c[i] = a[i] + b[i];
        d[i] = c[i] + 1;
    }
}

void dist2(int a[], int b[], int c[], int d[]) {
    for (int i=1; i<1000; i++) {
        b[i] = a[i] + 1;
        #pragma distribute_point
        // Distribution will start here,
        // ignoring all loop-carried dependency.
        c[i] = a[i] + b[i];
        d[i] = c[i] + 1;
    }
}
```

**inline, noinline, forceinline**

Specifies inlining of all calls in a statement. This also describes pragmas `forceinline` and `noinline`.

**Syntax**

```c++
#pragma inline [recursive]
#pragma forceinline [recursive]
#pragma noinline
```

**Arguments**

- **recursive**
  
  Indicates that the pragma applies to all of the calls that are called by these calls, recursively, down the call chain.

**Description**

These are statement-specific inlining pragmas. Each can be placed before a C/C++ statement, and it will then apply to all of the calls within a statement and all calls within statements nested within that statement.

The `forceinline` pragma indicates that the calls in question should be inlined whenever the compiler is capable of doing so.

The `inline` pragma is a hint to the compiler that the user prefers that the calls in question be inlined, but expects the compiler not to inline them if its heuristics determine that the inlining would be overly aggressive and might slow down the compilation of the source code excessively, create too large of an executable, or degrade performance.

The `noinline` pragma indicates that the calls in question should not be inlined.

These statement-specific pragmas take precedence over the corresponding function-specific pragmas.
Example: Using the `forceinline recursive` pragma

```c
#include <stdio.h>
static void fun(float a[100][100], float b[100][100]) {
    int i, j;
    for (i = 0; i < 100; i++) {
        for (j = 0; j < 100; j++) {
            a[i][j] = 2 * i;
            b[i][j] = 4 * j;
        }
    }
}

static void sun(float a[100][100], float b[100][100]) {
    int i, j;
    for (i = 0; i < 100; i++) {
        for (j = 0; j < 100; j++) {
            a[i][j] = 2 * i;
            b[i][j] = 4 * j;
            fun(a, b);
        }
    }
}
static float a[100][100];
static float b[100][100];
extern int main() {
    int i, j;
    for (i = 0; i < 100; i++) {
        for (j = 0; j < 100; j++) {
            a[i][j] = i + j;
            b[i][j] = i - j;
        }
    }
    for (i = 0; i < 99; i++) {
        fun(a, b);
        #pragma forceinline recursive
        for (j = 0; j < 99; j++) {
            sun(a, b);
        }
    }
    fprintf(stderr, "%d %d\n", a[99][9], b[99][99]);
}
```

The `forceinline recursive` pragma applies to the call 'sun(a,b)' as well as the call 'fun(a,b)' called inside 'sun(a,b)'.

**intel_omp_task**

*For Intel legacy tasking, specifies a unit of work, potentially executed by a different thread.*

**Syntax**

```c
#pragma intel_omp_task [clause[,clause]...]```
structured-block

Arguments

clause

Can be any of the following:

private (variable-list)

Creates a private, default-constructed version for each object in variable-list for the task taskq. It also implies captureprivate on each enclosed task. The original object referenced by each variable retains its value but must not be modified within the dynamic extent of the task construct.

captureprivate (variable-list)

Creates a private, copy-constructed version for each object in variable-list for the task taskq at the time the task is queued. The original object referenced by each variable retains its value but must not be modified within the dynamic extent of the task construct.

Description

The `intel_omp_task` pragma specifies a unit of work, potentially executed by a different thread.

NOTE

This pragma affects parallelization done using the `-qopenmp` option. Options that use OpenMP are available for both Intel® and non-Intel microprocessors, but these options may perform additional optimizations on Intel® microprocessors than they perform on non-Intel microprocessors. The list of major, user-visible OpenMP constructs and features that may perform differently on Intel® vs. non-Intel microprocessors includes: locks (internal and user visible), the SINGLE construct, barriers (explicit and implicit), parallel loop scheduling, reductions, memory allocation, and thread affinity and binding.

`intel_omp_taskq`

For Intel legacy tasking, specifies an environment for the while loop in which to queue the units of work specified by the enclosed `task` pragma.

Syntax

```c
#pragma intel_omp_taskq[clause[,[,]clause]...]
```

structured-block

Arguments

clause

Can be any of the following:

private (variable-list)

Creates a private, default-constructed version for each object in variable-list for the task taskq. It also implies captureprivate on each enclosed task. The original object
firstprivate (variable-list)

Creates a private, copy-constructed version for each object in variable-list for the taskq. It also implies captureprivate on each enclosed task. The original object referenced by each variable must not be modified within the dynamic extent of the construct and has an indeterminate value upon exit from the construct.

lastprivate (variable-list)

Creates a private, default-constructed version for each object in variable-list for the taskq. It also implies captureprivate on each enclosed task. The original object referenced by each variable has an indeterminate value upon entry to the construct, must not be modified within the dynamic extent of the construct, and is copy-assigned the value of the object from the last enclosed task after that task completes execution.

reduction (operator : variable-list)

Performs a reduction operation with the given operator in enclosed task constructs for each object in variable-list. operator and variable-list are defined the same as in the OpenMP* Specifications.

ordered

Organizes ordered constructs in enclosed task constructs in original sequential execution order. The taskq pragma, to which the ordered is bound, must have an ordered clause present.

nowait

Removes the implied barrier at the end of the taskq. Threads may exit the taskq construct before completing all the task constructs queued within it.

Description

The intel_omp_taskq pragma specifies the environment within which the enclosed units of work (tasks) are to be executed. From among all the threads that encounter a intel_omp_taskq pragma, one is chosen to execute it initially.

Conceptually, the intel_omp_taskq pragma causes an empty queue to be created by the chosen thread, and then the code inside the taskq block is executed as single-threaded. All the other threads wait for work to be queued on the conceptual queue.
The `intel_omp_taskq` pragma specifies a unit of work, potentially executed by a different thread. When a task pragma is encountered lexically within a taskq block, the code inside the task block is conceptually queued on the queue associated with the taskq. The conceptual queue is disbanded when all work queued on it finishes, and when the end of the taskq block is reached.

**NOTE**
This pragma affects parallelization done using the `Qopenmp` (Windows*) or `qopenmp` (Linux*) option. Options that use OpenMP* are available for both Intel® and non-Intel microprocessors, but these options may perform additional optimizations on Intel® microprocessors than they perform on non-Intel microprocessors. The list of major, user-visible OpenMP* constructs and features that may perform differently on Intel® vs. non-Intel microprocessors includes: locks (internal and user visible), the SINGLE construct, barriers (explicit and implicit), parallel loop scheduling, reductions, memory allocation, and thread affinity and binding.

### ivdep
*Instructs the compiler to ignore assumed vector dependencies.*

**Syntax**
```c
#pragma ivdep
```

**Arguments**
None

**Description**
The `ivdep` pragma instructs the compiler to ignore assumed vector dependencies. To ensure correct code, the compiler treats an assumed dependence as a proven dependence, which prevents vectorization. This pragma overrides that decision. Use this pragma only when you know that the assumed loop dependencies are safe to ignore.

In addition to the `ivdep` pragma, the `vector` pragma can be used to override the efficiency heuristics of the vectorizer.

**NOTE**
The proven dependencies that prevent vectorization are not ignored, only assumed dependencies are ignored.

**Examples**

```
void ignore_vec_dep(int *a, int k, int c, int m) {
    #pragma ivdep
    for (int i = 0; i < m; i++)
        a[i] = a[i + k] * c;
}
```

The loop in this example will not vectorize without the `ivdep` pragma, since the value of `k` is not known; vectorization would be illegal if `k < 0`. 
The pragma binds only the for loop contained in current function. This includes a for loop contained in a sub-function called by the current function.

**Example**

```c
#pragma ivdep
for (i=1; i<n; i++) {
    e[ix[2][i]] = e[ix[2][i]]+1.0;
    e[ix[3][i]] = e[ix[3][i]]+2.0;
}
```

This loop requires the parallel option in addition to the ivdep pragma to indicate there is no loop-carried dependencies:

**Example**

```c
#pragma ivdep
for (j=0; j<n; j++) { a[b[j]] = a[b[j]] + 1; }
```

This loop requires the parallel option in addition to the ivdep pragma to ensure there is no loop-carried dependency for the store into a().

**See Also**

Function Annotations and the SIMD Directive for Vectorization
	novector pragma

vector pragma

**loop_count**

Specifies the iterations for a for loop.

**Syntax**

```c
#pragma loop_count(n)
#pragma loop_count=n
```

or

```c
#pragma loop_count(n1[, n2]...)
#pragma loop_count=n1[, n2]...
```

or

```c
#pragma loop_count min(n), max(n), avg(n)
#pragma loop_count min=n, max=n, avg=n
```

**Arguments**

- *(n) or =n*
  
  A non-negative integer value. The compiler will attempt to iterate the next loop the number of times specified in n; however, the number of iterations is not guaranteed.

- *(n1[, n2]...) or = n1[, n2]...*
  
  Non-negative integer values. The compiler will attempt to iterate the next loop the number of time specified by n1 or n2, or some other unspecified...
number of times. This behavior allows the compiler some flexibility in attempting to unroll the loop. The number of iterations is not guaranteed.

Non-negative integer values. Specify one or more in any order without duplication. The compiler insures the next loop iterates for the specified maximum, minimum, or average number ($nI$) of times. The specified number of iterations is guaranteed for min and max.

**Description**

The `loop_count` pragma specifies the minimum, maximum, or average number of iterations for a `for` loop. In addition, a list of commonly occurring values can be specified to help the compiler generate multiple versions and perform complete unrolling.

You can specify more than one pragma for a single loop; however, do not duplicate the pragma.

**Example**

The following example illustrates how to use the `loop_count` pragma to iterate through the loop a minimum of three, a maximum of ten, and average of five times.

```
#include <stdio.h>
int i;
int mysum(int start, int end, int a) {
    int iret=0;
    #pragma loop_count min(3), max(10), avg(5)
    for (i=start;i<=end;i++)
        iret += a;
    return iret;
}

int main() {
    int t;
    t = mysum(1, 10, 3);
    printf("t1=%d\n",t);
    t = mysum(2, 6, 2);
    printf("t2=%d\n",t);
    t = mysum(5, 12, 1);
    printf("t3=%d\n",t);
}
```

**nofusion**

Prevents a loop from fusing with adjacent loops.

**Syntax**

```
#pragma nofusion
```

**Arguments**

None
Description

The **nofusion** pragma lets you fine tune your program on a loop-by-loop basis. This pragma should be placed immediately before the loop that should not be fused.

### Example

```c
#define SIZE 1024

int sub () {
    int B[SIZE], A[SIZE];
    int i, j, k=0;
    for(j=0; j<SIZE; j++)

    #pragma nofusion
    for (i=0; i<SIZE; i++)
        k += A[i] + 1;
    return k;
}
```

### novector

Specifies that a particular loop should never be vectorized.

#### Syntax

```c
#pragma novector
```

#### Arguments

None

#### Description

The **novector** pragma specifies that a particular loop should never be vectorized, even if it is legal to do so. When avoiding vectorization of a loop is desirable (when vectorization results in a performance regression rather than improvement), the **novector** pragma can be used in the source text to disable vectorization of a loop. This behavior is in contrast to the **vector always** pragma.

### Example: Using the **novector** pragma

```c
void foo(int lb, int ub) {
    #pragma novector
    for(j=lb; j<ub; j++) { a[j]=a[j]+b[j]; }
}
```

When the trip count (ub - lb) is too low to make vectorization worthwhile, you can use the **novector** pragma to tell the compiler not to vectorize, even if the loop is considered vectorizable.

#### See Also

Function Annotations and the SIMD Directive for Vectorization

**vector pragma**
omp simd early_exit

Extends #pragma omp simd, allowing vectorization of multiple exit loops.

Syntax

#pragma omp simd early_exit

Description

Extends #pragma omp simd allowing vectorization of multiple exit loops. When this clause is specified:

- Each operation before last lexical early exit of the loop may be executed as if early exit were not triggered within the SIMD chunk.
- After the last lexical early exit of the loop, all operations are executed as if the last iteration of the loop was found.
- Each list item specified in the linear clause is computed based on the last iteration number upon exiting the loop.
- The last value for linear clauses and conditional lastprivates clauses are preserved with respect to scalar execution.
- The last value for reductions clauses are computed as if the last iteration in the last SIMD chunk was executed up on exiting the loop.
- The shared memory state may not be preserved with regard to scalar execution.
- Exceptions are not allowed.

Examples

The following example demonstrates how to use this pragma.

In the following example, the pragma specifies that the vector execution of the for loop is safe even though the loop may exit before the loop upper bound condition $j < ub$ becomes false. Suppose $j_1$ is the smallest $j$, between $lb$ and $ub$, such that $j$ satisfies $b[j] \leq 0$. If $j_1$ and $j_1+1, j_1+2, ...$ are within the same (last) SIMD chunk, read of $b[j_1], b[j_1+1], b[j_1+2], ...$ and the subsequent evaluation of $\leq 0$ will happen unconditionally, unlike the scalar execution of the same loop. Safety of such vector evaluation is programmer's responsibility. If necessary, simdlen() clause can be used to control the SIMD chunk size.

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>void foo(int lb, int ub) {</td>
</tr>
<tr>
<td>float a = 0;</td>
</tr>
<tr>
<td>#pragma omp simd early_exit reduction(+:a)</td>
</tr>
<tr>
<td>for(j=lb; j&lt;ub; j++) {</td>
</tr>
<tr>
<td>if (b[j] &lt;= 0)</td>
</tr>
<tr>
<td>break;</td>
</tr>
<tr>
<td>a += b[j];</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

optimize

Enables or disables optimizations for code after this pragma until another optimize pragma or end of the translation unit.

Syntax

#pragma optimize("", on|off)
Arguments
The compiler ignores first argument values. Valid second arguments for optimize are:

- **off** Disables optimization
- **on** Enables optimization

Description
The optimize pragma is used to enable or disable optimizations.

Specifying `#pragma optimize("", off)` disables optimization until either the compiler finds a matching `#pragma optimize("", on)` statement or until the compiler reaches the end of the translation unit.

Examples

**Example: Disabling optimization for a single function using the optimize pragma**

```c
#pragma optimize("", off)
alpha() { ... }
#pragma optimize("", on)
omega() { ... }
```

In this example, optimizations are disabled for the `alpha()` function but not for the `omega()` function.

**Example: Disabling optimization for all functions using the optimize pragma**

```c
#pragma optimize("", off)
alpha() { ... }
omega() { ... }
```

In this example, optimizations are disabled for both the `alpha()` and `omega()` functions.

optimization_level
Controls optimization for one function or all functions after its first occurrence.

Syntax

```
#pragma [intel|GCC] optimization_level n
```

Arguments

- **intel|GCC** Indicates the interpretation to use
- **n** An integer value specifying an optimization level; valid values are:
  - 0: same optimizations as option `-00` (Linux*)
  - `od` (Windows*)
  - 1: same optimizations as option `-01`
  - 2: same optimizations as option `-02`
  - 3: same optimizations as option `-03`
Description

The optimization_level pragma is used to restrict optimization for a specific function while optimizing the remaining application using a different, higher optimization level. For example, if you specify option level O3 for the application and specify #pragma optimization_level 1, the marked function will be optimized at option level O1, while the remaining application will be optimized at the higher level.

In general, this pragma optimizes the function at the level specified as n; however, certain compiler optimizations, like Inter-procedural Optimization (IPO), are not enabled or disabled during translation unit compilation. For example, if you enable IPO and a specific optimization level, IPO is enabled even for the function targeted by this pragma; however, IPO might not be fully implemented regardless of the optimization level specified at the command line. The reverse is also true.

Scope of optimization restriction

On Linux* systems, the scope of the optimization restriction can be affected by arguments passed to the -pragma-optimization-level compiler option as explained in the following table.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma intel optimization_level n</td>
<td>Applies the pragma only to the next function, using the specified optimization level, regardless of the argument passed to the -pragma-optimization-level option.</td>
</tr>
<tr>
<td>#pragma GCC optimization_level n or #pragma GCC optimization_level reset</td>
<td>Applies the pragma to all subsequent functions, using the specified optimization level, regardless of the argument passed to the -pragma-optimization-level option. Specifying reset reverses the effect of the most recent #pragma GCC optimization_level statement, by returning to the optimization level previously specified.</td>
</tr>
<tr>
<td>#pragma optimization_level n</td>
<td>Applies either the Intel® oneAPI DPC++/C++ Compiler implementation or the GCC* interpretation. Interpretation depends on the argument passed to the -pragma-optimization-level option.</td>
</tr>
</tbody>
</table>

NOTE

On Windows* systems, the pragma always uses the intel interpretation; the pragma is applied only to the next function.

Examples

Place the pragma immediately before the function being affected.

**Example: intel interpretation of the optimization_level pragma**

```cpp
#pragma intel optimization_level 1
gamma() { ... }
```

**Example: GCC* interpretation of the optimization_level pragma**

```cpp
#pragma GCC optimization_level 1
gamma() { ... }
```
**optimization_parameter**

*Passes certain information about a function to the optimizer.*

**Syntax**

Linux*:

```bash
#pragma intel optimization_parameter target_arch=<CPU>
#pragma intel optimization_parameter inline-max-total-size=n
#pragma intel optimization_parameter inline-max-per-routine=n
```

Windows*:

```bash
#pragma [intel] optimization_parameter target_arch=<CPU>
#pragma [intel] optimization_parameter inline-max-total-size=n
#pragma [intel] optimization_parameter inline-max-per-routine=n
```

**Arguments**

`inline-max-per-routine=n`  
Specifies the maximum number of times the inliner may inline into the routine. `n` is one of the following:

- A non-negative integer constant that specifies the maximum number of times the inliner may inline into the function. If you specify zero, no inlining is done into the function.
- The keyword `unlimited`, which means that there is no limit to the number of times the inliner may inline into the function.

`inline-max-total-size=n`  
Specifies how much larger a function can normally grow when inline expansion is performed. `n` is one of the following:

- A non-negative integer constant that specifies the permitted increase in the function's size when inline expansion is performed. If you specify zero, no inlining is done into the function.
- The keyword `unlimited`, which means that there is no limit to the size a function may grow when inline expansion is performed.

**Description**

Place `#pragma intel optimization_parameter target_arch=<CPU>` at the head of a function to get the compiler to target that function for a specified instruction set. The pragma applies only to the function before which it is placed.

The pragmas `intel optimization_parameter inline-max-total-size=n` and `intel optimization_parameter inline-max-per-routine=n` specify information used during inlining into a function.

**Examples**
Example: Targeting code for Intel® Advanced Vector Extensions (Intel® AVX) processors

For C++:

```
icx -mAVX foo.c    // on Linux*
dpcpp -mAVX foo.c   // on Linux*
```

The following code targets just the function `bar` for Intel® AVX processors, regardless of the command line options used.

### Example: Targeting a function for Intel® Advanced Vector Extensions (Intel® AVX) processors

```
#pragma intel optimization_parameter target_arch=AVX
void bar() { ... }
```

parallel/noparallel

Resolves dependencies to facilitate auto-parallelization of the immediately following loop (parallel) or prevents auto-parallelization of the immediately following loop (noparallel).

### Syntax

```
#pragma parallel [clause[ [,]clause]...]
#pragma noparallel
```

### Arguments

**clause**

Can be any of the following:

- **always**
  - [assert]
  
  Overrides compiler heuristics that estimate whether parallelizing a loop would increase performance. Using this clause on a loop that the compiler finds to be parallelizable tells the compiler to parallelize the loop even if doing so might not improve performance.
  
  If **assert** is added, the compiler will generate an error-level assertion test to display a message saying that the compiler efficiency heuristics indicate that the loop cannot be vectorized.
  
- **firstprivate**
  - (var [ :expr ] ... )text
  
  Provides a superset of the functionality provided by the **private** clause. Variables that appear in a **firstprivate** list are subject to **private** clause semantics. In addition, its initial
value is broadcast to all private instances upon entering the parallel loop.

**lastprivate**

(lastprivate **(var [:expr] ...)**

Provides a superset of the functionality provided by the **private** clause. Variables that appear in a lastprivate list are subject to private clause semantics. In addition, when the parallel region is exited, each variable has the value that results from the sequentially last iteration of the loop up exiting the parallel loop.

**num_threads**

(num_threads (**(n)**)

Parallelizes the loop across **n** threads, where **n** is an integer.

**private**

(private **(var [:expr] ...)**

Specifies a list of scalar and array variables (**var**) to privatize. An array or pointer variable can take an optional argument (**expr**) which is an int32 or int64 expression denoting the number of array elements to privatize.

Like the **private** clause, both the **firstprivate** and the **lastprivate** clauses specify a list of scalar and array variables (**var**) to privatize. An array or pointer variable can take an optional argument (**expr**) which is an int32 or int64 expression denoting the number of array elements to privatize.

The same **var** is not allowed to appear in both the **private** and the **lastprivate** clauses for the same loop.

The same **var** is not allowed to appear in both the **private** and the **firstprivate** clauses for the same loop.

When **expr** is absent, the rules on **var** are the same as with OpenMP. The rules to be observed are as follows:

- **var** must not be part of another variable (as an array or structure element)
- **var** must not have a **const**-qualified type unless it is of class type with a mutable member
- **var** must not have an incomplete type or a reference type
- if **var** is of class type (or array thereof), then it requires an accessible, unambiguous default constructor for the class type. Furthermore, if this **var** is in a lastprivate clause, then it also requires an accessible, unambiguous copy assignment operator for the class type.
When `expr` is present, the same rules apply, but `var` must be an array or a pointer variable.

- If `var` is an array, then only its first `expr` elements are privatized. Without `expr`, the entire array is privatized.
- If `var` is a pointer, then the first `expr` elements are privatized (element size given by the pointer’s target type). Without `expr`, only the pointer variable itself is privatized.
- Program behavior is undefined if `expr` evaluates to a non-positive value, or if it exceeds the array size.

**Description**

The `parallel` pragma instructs the compiler to ignore potential dependencies that it assumes could exist and which would prevent correct parallelization in the immediately following loop. However, if dependencies are proven, they are not ignored.

The `noparallel` pragma prevents autoparallelization of the immediately following loop.

**Caution**

Use this pragma with care. If a loop has cross-iteration dependencies, annotating it with this pragma can lead to incorrect program behavior.

Only use the `parallel` pragma if it is known that parallelizing the annotated loop will improve its performance.

**Example: Using the `parallel` pragma**

```cpp
void example(double *A, double *B, double *C, double *D) {
    int i;
    #pragma parallel
    for (i=0; i<10000; i++) {
        A[i] += B[i] + C[i];
        C[i] += A[i] + D[i];
    }
}
```

**prefetch/noprefetch**

Invites the compiler to issue or disable requests to prefetch data from memory. This pragma applies only to Intel® Advanced Vector Extensions 512 (Intel® AVX-512).

**Syntax**

```cpp
#pragma prefetch
#pragma prefetch *:hint[:distance]
#pragma prefetch [var1 [: hint1 [:distance1]] [, var2 [: hint2 [: distance2]]]...]
#pragma noprefetch [var1 [, var2]...]
```
Arguments

var

An optional memory reference (data to be prefetched)

hint

An optional hint to the compiler to specify the type of prefetch. Possible values:

- 1: For integer data that will be reused
- 2: For integer and floating point data that will be reused from L2 cache
- 3: For data that will be reused from L3 cache
- 4: For data that will not be reused

To use this argument, you must also specify var.

distance

An optional integer argument with a value greater than 0. It indicates the number of loop iterations ahead of which a prefetch is issued, before the corresponding load or store instruction. To use this argument, you must also specify var and hint.

Description

This pragma hints to the compiler to generate data prefetches for some memory references. These hints affect the heuristics used in the compiler. Prefetching data can minimize the effects of memory latency.

If you specify the prefetch pragma with no arguments, all arrays accessed in the immediately following loop are prefetched.

If the loop includes the expression A(j), placing #pragma prefetch A in front of the loop instructs the compiler to insert prefetches for A(j + d) within the loop. Here, d is the number of iterations ahead of which to prefetch the data, and is determined by the compiler.

If you specify #pragma prefetch *, then hint and distance prefetches all array accesses in the loop.

To use these pragmas, the compiler general optimization level must be set at option O2 or higher.

The noprefetch pragma hints to the compiler not to generate data prefetches for some memory references. This affects the heuristics used in the compiler.

Examples

Example: Using the prefetch pragma

```c
#pragma prefetch htab_p:1:30
#pragma prefetch htab_p:0:6

// Issue vprefetch1 for htab_p with a distance of 30 vectorized iterations ahead
// Issue vprefetch0 for htab_p with a distance of 6 vectorized iterations ahead
// If pragmas are not present, compiler chooses both distance values

for (j=0; j<2*N; j++) { htab_p[i*m1 + j] = -1; }
```

Example: Using noprefetch and prefetch pragmas together

```c
#pragma noprefetch b
#pragma prefetch a

for(i=0; i<m; i++) { a[i]=b[i]+1; }
```
### Example: Using `noprefetch` and `prefetch` pragmas together

```c
for (i=i0; i!=i1; i+=is) {
    float sum = b[i];
    int ip = srow[i];
    int c = col[ip];

    #pragma noprefetch col
    #pragma prefetch value:1:80
    #pragma prefetch x:1:40

    for(; ip<srow[i+1]; c=col[++ip])
        sum -= value[ip] * x[c];
    y[i] = sum;
}
```

### simd

*Enforces vectorization of loops.*

**Syntax**

```c
#pragma simd [clause[ [,] clause]...]
```

**Arguments**

`clause`

Can be any of the following:

- `vectorlength (n1[, n2]...)`
  - Where `n` is a vector length (VL). It must be an integer that is a power of 2; the value must be 2, 4, 8, or 16. If you specify more than one `n`, the vectorizer will choose the VL from the values specified.

  Causes each iteration in the vector loop to execute the computation equivalent to `n` iterations of scalar loop execution. Multiple `vectorlength` clauses are merged as a union.

- `vectorlengthfor (data type)`
  - Where `data type` must be one of built-in integer types (8-, 16-, 32-, or 64-bit), pointer types (treated as pointer-sized integer), floating point types (32- or 64-bit), or complex types (64- or 128-bit). Otherwise, behavior is undefined.

  Causes each iteration in the vector loop to execute the computation equivalent to `n` iterations of scalar loop execution where `n` is computed from `size_of_vector_register/sizeof(data type)`.

---

Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference

572
For example, `vectorlengthfor(float)` results in $n=4$ for Intel® Streaming SIMD Extensions (Intel® SSE2) to Intel SSE4.2 targets (packed float operations available on 128bit XMM registers) and $n=8$ for an Intel® Advanced Vector Extensions (Intel® AVX) target (packed float operations available on 256bit YMM registers).

`vectorlengthfor(int)` results in $n=4$ for Intel SSE2 to Intel AVX targets.

`vectorlength()` and `vectorlengthfor()` clauses are mutually exclusive. In other words, the `vectorlengthfor()` clause may not be used with the `vectorlength()` clause, and vice versa.

Behavior for multiple `vectorlengthfor` clauses is undefined.

```plaintext
private ([var1[, var2][,...])
```

Where `var` is a scalar variable. 
Causes each variable to be private to each iteration of a loop. Unless the variable appears in `firstprivate` clause, the initial value of the variable for the particular iteration is undefined. Unless the variable appears in `lastprivate` clause, the value of the variable upon exit of the loop is undefined. Multiple `private` clauses are merged as a union.

```plaintext
firstprivate ([var1[, var2][,...])
```

Provides a superset of the functionality provided by the `private` clause. Variables that appear in a `firstprivate` list are subject to `private` clause semantics. In addition, its initial value is broadcast to all private instances for each iteration upon entering the SIMD loop.

A variable in a `firstprivate` clause can appear in a `lastprivate` clause.

**NOTE**

Execution of the SIMD loop with `firstprivate/lastprivate` clauses may be different from serial execution of the same code even if the loop fails to vectorize.

A variable in a `private` clause cannot appear in a `linear`, `reduction`, `firstprivate`, or `lastprivate` clause.
A variable in a `firstprivate` clause cannot appear in a `linear`, `reduction`, or `private` clause.

`lastprivate (var1[, var2]...)` Provides a superset of the functionality provided by the `private` clause. Variables that appear in a `lastprivate` list are subject to `private` clause semantics. In addition, when the SIMD loop is exited, each variable has the value that resulted from the sequentially last iteration of the SIMD loop (which may be undefined if the last iteration does not assign to the variable).

A variable in a `lastprivate` clause can appear in a `firstprivate` clause.

A variable in a `lastprivate` clause cannot appear in a `linear`, `reduction`, or `private` clause.

`linear (var1:step1 [, var2:step2]...)` Where `var` is a scalar variable and `step` is a compile-time positive, integer constant expression.

For each iteration of a scalar loop, `var1` is incremented by `step1`, `var2` is incremented by `step2`, and so on. Therefore, every iteration of the vector loop increments the variables by `VL*step1`, `VL*step2`, ..., to `VL*stepN`, respectively. If more than one step is specified for a `var`, a compile-time error occurs. Multiple linear clauses are merged as a union.

A variable in a `linear` clause cannot appear in a `reduction`, `private`, `firstprivate`, or `lastprivate` clause.

`reduction (oper:var1 [,var2]...)` Where `oper` is a reduction operator and `var` is a scalar variable.

Applies the vector reduction indicated by `oper` to `var1`, `var2`, ..., `varN`. The `simd` pragma may have multiple reduction clauses with the same or different operators. If more than one reduction operator is associated with a `var`, a compile-time error occurs.

A variable in a `reduction` clause cannot appear in a `linear`, `private`, `firstprivate`, or `lastprivate` clause.
[no]assert  Directs the compiler to assert or not to assert when the vectorization fails. The default is noassert. If this clause is specified more than once, a compile-time error occurs.

[no]vecremainder  Instructs the compiler to vectorize or not to vectorize the remainder loop when the original loop is vectorized. See the description of the vector pragma for more information.

Description

The simd pragma is used to guide the compiler to vectorize more loops. Vectorization using the simd pragma complements (but does not replace) the fully automatic approach.

Without explicit vectorlength() and vectorlengthfor() clauses, the compiler will choose a vectorlength using its own cost model. Misclassification of variables into private, firstprivate, lastprivate, linear, and reduction, or lack of appropriate classification of variables may cause unintended consequences such as runtime failures and/or incorrect result.

You can only specify a particular variable in at most one instance of a private, linear, or reduction clause.

If the compiler is unable to vectorize a loop, a warning will be emitted (use the assert clause to make it an error).

If the vectorizer has to stop vectorizing a loop for some reason, the fast floating-point model is used for the SIMD loop.

The vectorization performed on this loop by the simd pragma overrides any setting you may specify for options -fp-model (Linux*) and /fp (Windows*) for this loop.

**NOTE**
- fp-model is only available for C++; it is not available for DPC++.

Note that the simd pragma may not affect all auto-vectorizable loops. Some of these loops do not have a way to describe the SIMD vector semantics.

The following restrictions apply to the simd pragma:

- The countable loop for the simd pragma has to conform to the for-loop style of an OpenMP worksharing loop construct. Additionally, the loop control variable must be a signed integer type.
- The vector values must be signed 8-, 16-, 32-, or 64-bit integers, single or double-precision floating point numbers, or single or double-precision complex numbers.
- A SIMD loop may contain another loop (for, while, do-while) in it. Goto out of such inner loops are not supported. Break and continue are supported.

**NOTE** Inlining can create such an inner loop, which may not be obvious at the source level.

- A SIMD loop performs memory references unconditionally. Therefore, all address computations must result in valid memory addresses, even though such locations may not be accessed if the loop is executed sequentially.
User-mandated vectorization, also called SIMD vectorization can assert or not assert an error if a #pragma simd annotated loop fails to vectorize. By default, the simd pragma is set to noassert, and the compiler will issue a warning if the loop fails to vectorize. To direct the compiler to assert an error when the #pragma simd annotated loop fails to vectorize, add the assert clause to the simd pragma. If a simd pragma annotated loop is not vectorized by the compiler, the loop holds its serial semantics.

**Example: Using the simd pragma**

```cpp
void add_floats(float *a, float *b, float *c, float *d, float *e, int n){
    int i;
    #pragma simd
    for (i=0; i<n; i++){
        a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
    }
}
```

In the example above, the function add_floats() uses too many unknown pointers for the compiler’s automatic runtime independence check optimization to kick-in. The programmer can enforce the vectorization of this loop by using the simd pragma to avoid the overhead of runtime check.

**See Also**

Function Annotations and the SIMD Directive for Vectorization

fp-model, fp  compiler option

vec, Qvec  compiler option

vector pragma

**simdoff**

Specifies a block of code in the SIMD loop or SIMD-enabled function that should be executed serially, in a logical order of SIMD lanes.

**Syntax**

```cpp
#pragma simdoff
structured-block
```

**Arguments**

None.

**Description**

The simdoff block will use a single SIMD lane to execute operations in the order of the loop iterations, or logical lanes of a SIMD-enabled function. This preserves ordering of operations in the block with respect to each other, and correlates with iteration space of the enclosing SIMD construct. The ordered simd block is executed in order, with respect to each SIMD lane or each loop iteration. The operations within the ordered simd or simdoff block can be re-ordered by optimizations, as long as the original execution semantics are preserved.

simdoff blocks allow the isolation and resolution of situations prohibited from SIMD execution. This includes cross-iteration data dependencies, function calls with side effects, such as OpenMP, oneTBB and native thread synchronization primitives.

simdoff sections are useful for resolving cross-iteration data dependencies in otherwise data-parallel computations. For example, the section may handle histogram updates as shown below:
```c
#pragma simd
for (int i = 0; i < N; i++)
{
    float amount = compute_amount(i);
    int   cluster = compute_cluster(i);
#pragma simdoff
    {  
        totals[cluster] += amount; // Requires ordering to process multiple updates for the same cluster
    }
}
```

### target variant dispatch

**Conditionally calls a procedure offload variant if the specified device is available; otherwise, executes the procedure on the host.**

#### Syntax

```
#pragma omp target variant dispatch {device(integer-expression) | nowait | subdevice([integer-constant ,] integer-expression [ : integer-expression [ : integer-expression] ] ) } | use_device_pointer (ptr-list))
```

#### Arguments

- **device**
  - Tells the compiler to call the variant only if device \( n \) is available.

- **subdevice**
  - Tells the compiler to call the variant only if the specified tiles or compute slices are available.

- **nowait**
  - Tells the compiler that calls to the procedure can occur asynchronously. If nowait is not specified, calls occur synchronously.

- **use_device_ptr**
  - Tells the compiler to use the corresponding device pointer instead of the host pointer when the variant procedure is called.

If both **device** and **subdevice** are specified, the variant is called only if the specified tiles or compute slices are available on device \( n \). Otherwise, the base version of the procedure is called on the host.

#### Description

The `omp target variant dispatch` pragma causes the compiler to emit conditional dispatch code around the associated procedure call that follows the pragma. If the specified device is available, the variant version is called.

The name of the procedure associated with the `omp target variant dispatch` pragma must have appeared in an `omp declare variant` pragma in the specification part of the calling scope. The interface of the variant procedure must be accessible in the base procedure where `omp target variant dispatch` appears.
NOTE
Use pragma `omp target variant dispatch` when calling Intel® oneAPI Math Kernel Library (oneMKL).

In other cases, we recommend you use the OpenMP* pragma `omp dispatch`. For more information about pragma `omp dispatch`, see the OpenMP* documentation.

**unroll/nounroll**

_Tells the compiler to unroll or not to unroll a counted loop._

**Syntax**

```cpp
#pragma unroll
#pragma unroll(n)
#pragma nounroll
```

**Arguments**

- **n**
  
  The unrolling factor representing the number of times to unroll a loop; it must be an integer constant from 0 through 255.

**Description**

The `unroll[n]` pragma tells the compiler how many times to unroll a counted loop.

The `unroll` pragma must precede the `for` statement for each `for` loop it affects. If `n` is specified, the optimizer unrolls the loop `n` times. If `n` is omitted or if it is outside the allowed range, the optimizer assigns the number of times to unroll the loop.

This pragma is supported only when option `O3` is set. The `unroll` pragma overrides any setting of loop unrolling from the command line.

The pragma can be applied for the innermost loop nest as well as for the outer loop nest. If applied to outer loop nests, the current implementation supports complete outer loop unrolling. The loops inside the loop nest are either not unrolled at all or completely unrolled. The compiler generates correct code by comparing `n` and the loop count.

When unrolling a loop increases register pressure and code size it may be necessary to prevent unrolling of a loop. In such cases, use the `nounroll` pragma. The `nounroll` pragma instructs the compiler not to unroll a specified loop.

**Examples**

**Example: Using the `unroll` pragma for innermost loop unrolling**

```cpp
void unroll(int a[], int b[], int c[], int d[]) {
    #pragma unroll(4)
    for (int i = 1; i < 100; i++) {
        b[i] = a[i] + 1;
        d[i] = c[i] + 1;
    }
}
```
Example: Using the `unroll` pragma for outer loop unrolling

```c
int m = 0;
int dir[4] = {1,2,3,4};
int data[10];
#pragma unroll (4)  // outer loop unrolling
for (int i = 0; i < 4; i++) {
    for (int j = dir[i]; data[j]==N ; j+=dir[i])
        m++;
}
```

When you place the `unroll` pragma before the first `for` loop, it causes the compiler to unroll the outer loop completely. If an `unroll` pragma is placed before the inner `for` loop as well as before the outer `for` loop, the compiler ignores the inner `for` loop `unroll` pragma. If the `unroll` pragma is placed only for the innermost loop, the compiler unrolls the innermost loop according to some factor.

### unroll_and_jam/nounroll_and_jam

*Enables or disables loop unrolling and jamming. These pragmas can only be applied to iterative for loops.*

**Syntax**

```c
#pragma unroll_and_jam
#pragma unroll_and_jam (n)
#pragma nounroll_and_jam
```

**Arguments**

- `n`
  
  The unrolling factor representing the number of times to unroll a loop; it must be an integer constant from 0 through 255

**Description**

The `unroll_and_jam` pragma partially unrolls one or more loops higher in the nest than the innermost loop and fuses/jams the resulting loops back together. This transformation allows more reuses in the loop.

This pragma is not effective on innermost loops. Ensure that the immediately following loop is not the innermost loop after compiler-initiated interchanges are completed.

Specifying this pragma is a hint to the compiler that the unroll and jam sequence is legal and profitable. The compiler enables this transformation whenever possible.

The `unroll_and_jam` pragma must precede the `for` statement for each `for` loop it affects. If `n` is specified, the optimizer unrolls the loop `n` times. If `n` is omitted or if it is outside the allowed range, the optimizer assigns the number of times to unroll the loop. The compiler generates correct code by comparing `n` and the loop count.

This pragma is supported only when compiler option O3 is set. The `unroll_and_jam` pragma overrides any setting of loop unrolling from the command line.

When unrolling a loop increases register pressure and code size it may be necessary to prevent unrolling of a nested loop or an imperfect nested loop. In such cases, use the `nounroll_and_jam` pragma. The `nounroll_and_jam` pragma hints to the compiler not to unroll a specified loop.
Example: Using the unroll_and_jam pragma

```c
int a[10][10];
int b[10][10];
int c[10][10];
int d[10][10];
void unroll(int n) {
    int i,j,k;
    #pragma unroll_and_jam (6)
    for (i = 1; i < n; i++) {
        #pragma unroll_and_jam (6)
        for (j = 1; j < n; j++) {
            for (k = 1; k < n; k++){
                a[i][j] += b[i][k]*c[k][j];
            }
        }
    }
}
```

**vector**

* Tells the compiler that the loop should be vectorized according to the argument keywords.

**Syntax**

```c
#pragma vector {always[assert]|dynamic_align|nodynamic_align|temporal|nontemporal|no|vecremainder|vectorlength(n1[, n2]...)}
#pragma vector nontemporal[(var1[, var2, ...])]
```

**Arguments**

- **always**
  - Instructs the compiler to override any efficiency heuristic during the decision to vectorize or not, and vectorize non-unit strides or very unaligned memory accesses; controls the vectorization of the subsequent loop in the program; optionally takes the keyword `assert`.

- **dynamic_align**
  - Instructs the compiler to perform dynamic alignment optimization for the loop.

- **nodynamic_align**
  - Disables dynamic alignment optimization for the loop.

- **nontemporal**
  - Instructs the compiler to use non-temporal (that is, streaming) stores on systems based on all supported architectures, unless otherwise specified; optionally takes a comma-separated list of variables.

  When this pragma is specified, it is your responsibility to also insert any fences as required to ensure correct memory ordering within a thread or across threads.

  One typical way to do this is to insert a `_mm_sfence` intrinsic call just after the loops (such as the initialization loop) where the compiler may insert streaming store instructions.
temporal

Instructs the compiler to use temporal (that is, non-streaming) stores on systems based on all supported architectures, unless otherwise specified.

vecremainder

Instructs the compiler to vectorize the remainder loop when the original loop is vectorized.

novecremainder

Instructs the compiler not to vectorize the remainder loop when the original loop is vectorized.

vectorlength \((n_1, n_2)\ldots\)

Instructs the vectorizer which vector length/factor to use when generating the main vector loop.

**Description**

The `vector` pragma indicates that the loop should be vectorized, if it is legal to do so, ignoring normal heuristic decisions about profitability. The `vector` pragma takes several argument keywords to specify the kind of loop vectorization required. The compiler does not apply the `vector` pragma to nested loops, each nested loop needs a preceding pragma statement. Place the pragma before the loop control statement.

**Using the `always` keyword**

When the `always` argument keyword is used, the pragma controls the vectorization of the subsequent loop in the program. If `assert` is added, the compiler will generate an error-level assertion test to display a message saying that the compiler efficiency heuristics indicate that the loop cannot be vectorized.

**Using the `dynamic_align` and `nodynamic_align` keywords**

Dynamic alignment is an optimization the compiler can perform to improve alignment of memory references inside the loop. It involves peeling iterations from the vector loop into a scalar loop (which may, in turn, also be vectorized) before the vector loop so that the vector loop aligns with a particular memory reference. Specifying `dynamic_align` enables the optimization to be performed, but the compiler will still use efficiency heuristics to determine whether the optimization will be applied to the loop. Specifying `nodynamic_align` disables the optimization. By default, the compiler does not perform optimization.

**Using the `nontemporal` and `temporal` keywords**

The `nontemporal` and `temporal` argument keywords are used to control how the "stores" of register contents to storage are performed (streaming versus non-streaming) on systems based on IA-32 and Intel® 64 architectures.

By default, the compiler automatically determines whether a streaming store should be used for each variable.

Streaming stores may cause significant performance improvements over non-streaming stores for large numbers on certain processors. However, the misuse of streaming stores can significantly degrade performance.

**Using the `[no]vecremainder` keyword**

If keyword `vecremainder` is specified, the compiler tries to vectorize the remainder loop when the main loop is vectorized. Even if the `always` keyword is specified, the remainder loop vectorization is still a subject of compiler efficiency heuristics.

If keyword `novecremainder` is specified, the compiler vectorizes the main loop, but it does not vectorize the remainder loop.

**Using the `vectorlength` keyword**

\(n\) is an integer power of 2; the value must be 2, 4, 6, 8, 16, 32, or 64. If more than one value is specified, the vectorizer will choose one of the specified vector lengths based on a cost model decision.
NOTE
The pragma vector should be used with care.
Overriding the efficiency heuristics of the compiler should only be done if the programmer is absolutely sure that vectorization will improve performance.

See Also
Function Annotations and the SIMD Directive for Vectorization

Intel-supportedPragma Reference

The Intel® oneAPI DPC++/C++ Compiler supports the following pragmas to ensure compatibility with other compilers.

Pragmas Compatible with the Microsoft* Compiler

The following pragmas are compatible with the Microsoft Compiler. For more information about these pragmas, go to the Microsoft Developer Network (http://msdn.microsoft.com).

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_text</td>
<td>Names the code section where the specified function definitions are to reside.</td>
</tr>
<tr>
<td>auto_inline</td>
<td>Excludes any function defined within the range where off is specified from being considered as candidates for automatic inline expansion.</td>
</tr>
<tr>
<td>bss_seg</td>
<td>Indicates to the compiler the segment where uninitialized variables are stored in the .obj file.</td>
</tr>
<tr>
<td>check_stack</td>
<td>The on argument indicates that stack checking should be enabled for functions that follow and the off argument indicates that stack checking should be disabled for functions that follow.</td>
</tr>
<tr>
<td>code_seg</td>
<td>Specifies a code section where functions are to be allocated.</td>
</tr>
<tr>
<td>comment</td>
<td>Places a comment record into an object file or executable file.</td>
</tr>
<tr>
<td>component</td>
<td>Controls collecting of browse information or dependency information from within source files.</td>
</tr>
<tr>
<td>conform</td>
<td>Specifies the run-time behavior of the /Zc:forScope compiler option.</td>
</tr>
<tr>
<td>const_seg</td>
<td>Specifies the segment where functions are stored in the .obj file.</td>
</tr>
<tr>
<td>data_seg</td>
<td>Specifies the default section for initialized data.</td>
</tr>
<tr>
<td>deprecated</td>
<td>Indicates that a function, type, or any other identifier may not be supported in a future release or indicates that a function, type, or any other identifier should not be used any more.</td>
</tr>
<tr>
<td>fenv_access</td>
<td>Informs an implementation that a program may test status flags or run under a non-default control mode.</td>
</tr>
<tr>
<td>float_control</td>
<td>Specifies floating-point behavior for a function.</td>
</tr>
<tr>
<td>fp_contract</td>
<td>Allows or disallows the implementation to contract expressions.</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>loop</td>
<td>Controls how the loop code will be considered or excluded from consideration by the auto-vectorizer.</td>
</tr>
<tr>
<td>init_seg</td>
<td>Specifies the section to contain C++ initialization code for the translation unit.</td>
</tr>
<tr>
<td>message</td>
<td>Displays the specified string literal to the standard output device (stdout).</td>
</tr>
<tr>
<td>optimize</td>
<td>Specifies optimizations to be performed on functions below the pragma or until the next optimize pragma; implemented to partly support the Microsoft implementation of same pragma; for the Intel oneAPI DPC++/C++ Compiler implementation, see the optimize reference page.</td>
</tr>
<tr>
<td>pointers_to_members</td>
<td>Specifies whether a pointer to a class member can be declared before its associated class definition and is used to control the pointer size and the code required to interpret the pointer.</td>
</tr>
<tr>
<td>pop_macro</td>
<td>Sets the value of the specified macro to the value on the top of the stack.</td>
</tr>
<tr>
<td>push_macro</td>
<td>Saves the value of the specified macro on the top of the stack.</td>
</tr>
<tr>
<td>region/endregion</td>
<td>Specifies a code segment in the Microsoft Visual Studio* Code Editor that expands and contracts by using the outlining feature.</td>
</tr>
<tr>
<td>section</td>
<td>Creates a section in an .obj file. Once a section is defined, it remains valid for the remainder of the compilation.</td>
</tr>
<tr>
<td>vtordisp</td>
<td>The on argument enables the generation of hidden vtordisp members and the off disables them.</td>
</tr>
<tr>
<td></td>
<td>push argument pushes the current vtordisp setting to the internal compiler stack. pop argument removes the top record from the compiler stack and restores the removed value of vtordisp.</td>
</tr>
<tr>
<td>warning</td>
<td>Allows selective modification of the behavior of compiler warning messages.</td>
</tr>
<tr>
<td>weak</td>
<td>Declares symbol you enter to be weak.</td>
</tr>
</tbody>
</table>

**OpenMP* Standard Pragmas**

The Intel oneAPI DPC++/C++ Compiler currently supports OpenMP* 5.0 Version TR4, and some OpenMP Version 5.1 pragmas. Supported pragmas are listed below. For more information about these pragmas, reference the OpenMP* Version 5.1 specification.

Intel-specific clauses are noted in the affected pragma description.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp allocate</td>
<td>Specifies memory allocators to use for object allocation and deallocation</td>
</tr>
<tr>
<td>omp atomic</td>
<td>Specifies a computation that must be executed atomically.</td>
</tr>
<tr>
<td>omp barrier</td>
<td>Specifies a point in the code where each thread must wait until all threads in the team arrive.</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>omp cancel</td>
<td>Requests cancellation of the innermost enclosing region of the type specified, and causes the encountering task to proceed to the end of the cancelled construct.</td>
</tr>
<tr>
<td>omp cancellation point</td>
<td>Defines a point at which implicit or explicit tasks check to see if cancellation has been requested for the innermost enclosing region of the type specified. This construct does not implement a synchronization between threads or tasks.</td>
</tr>
<tr>
<td>omp critical</td>
<td>Specifies a code block that is restricted to access by only one thread at a time.</td>
</tr>
<tr>
<td>omp declare reduction</td>
<td>Declares User-Defined Reduction (UDR) functions (reduction identifiers) that can be used as reduction operators in a reduction clause.</td>
</tr>
<tr>
<td>omp declare simd</td>
<td>Creates a version of a function that can process multiple arguments using Single Instruction Multiple Data (SIMD) instructions from a single invocation from a SIMD loop.</td>
</tr>
<tr>
<td>omp declare target</td>
<td>Specifies functions and variables that are created or mapped to a device.</td>
</tr>
<tr>
<td>omp declare variant</td>
<td>Identifies a variant of a base procedure and specifies the context in which this variant is used.</td>
</tr>
<tr>
<td>omp dispatch</td>
<td>Determines if a procedure variant is called for a given procedure.</td>
</tr>
<tr>
<td>omp distribute</td>
<td>Specifies that the iterations of one or more loops should be distributed among the initial threads of all thread teams in a league.</td>
</tr>
<tr>
<td>omp distribute parallel for</td>
<td>Specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams.</td>
</tr>
<tr>
<td>omp distribute parallel for simd</td>
<td>Specifies a loop that will be executed in parallel by multiple threads that are members of multiple teams. It will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp distribute simd</td>
<td>Specifies a loop that will be distributed across the primary threads of the teams region. It will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp flush</td>
<td>Identifies a point at which a thread's temporary view of memory becomes consistent with the memory.</td>
</tr>
<tr>
<td>omp for</td>
<td>Specifies a work-sharing loop. Iterations of the loop are executed in parallel by the threads in the team.</td>
</tr>
<tr>
<td>omp for simd</td>
<td>Specifies that the iterations of the loop will be distributed across threads in the team. Iterations executed by each thread can also be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp interop</td>
<td>Identifies a foreign runtime context and identifies runtime characteristics of that context, enabling interoperability with it.</td>
</tr>
<tr>
<td>omp loop</td>
<td>Specifies that the iterations of the associated loops can execute in any order or concurrently.</td>
</tr>
<tr>
<td>omp masked</td>
<td>Specifies a block of code to be executed by the primary thread (thread 0) of the team.</td>
</tr>
<tr>
<td>omp master (deprecated; see omp masked)</td>
<td>Specifies a code block that must be executed only once by the primary thread of the team.</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>omp ordered</td>
<td>Specifies a block of code that the threads in a team must execute in the natural order of the loop iterations, or as a stand-alone directive, specifies cross-iteration dependences in a doacross loop-nest.</td>
</tr>
<tr>
<td>omp ordered simd</td>
<td>Specifies a block of code in the SIMD loop or SIMD-enabled function that should be executed serially, in a logical order of SIMD lanes.</td>
</tr>
<tr>
<td>omp ordered simd monotonic</td>
<td>Specifies a block of code in which the value of the new list item on each iteration of the associated SIMD loop(s) corresponds to the value of the original list item before entering the associated loop, plus the number of the iterations for which the conditional update happens prior to the current iteration, times linear-step. The value corresponding to the sequentially last iteration of the associated loop(s) is assigned to the original list item.</td>
</tr>
<tr>
<td>omp ordered simd overlap</td>
<td>Specifies a block of code that has to be executed scalar for overlapping inx values and parallel for different inx values within SIMD loop.</td>
</tr>
<tr>
<td>omp parallel</td>
<td>Specifies that a structured block should be run in parallel by a team of threads.</td>
</tr>
<tr>
<td>omp parallel for</td>
<td>Provides an abbreviated way to specify a parallel region containing only a FOR construct.</td>
</tr>
<tr>
<td>omp parallel for simd</td>
<td>Specifies a parallel construct that contains one for simd construct and no other statement.</td>
</tr>
<tr>
<td>omp parallel sections</td>
<td>Specifies a parallel construct that contains only a sections construct.</td>
</tr>
<tr>
<td>omp requires</td>
<td>Lists the features that an implementation must support so that the program compiles and runs correctly.</td>
</tr>
<tr>
<td>omp scope</td>
<td>Defines a structured block that is executed by all threads in a team but where additional OpenMP* operations can be specified.</td>
</tr>
<tr>
<td>omp sections</td>
<td>Defines a set of structured blocks that will be distributed among the threads in the team.</td>
</tr>
<tr>
<td>omp simd</td>
<td>Transforms the loop into a loop that will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp single</td>
<td>Specifies that a block of code is to be executed by only one thread in the team.</td>
</tr>
<tr>
<td>omp target</td>
<td>Creates a device data environment and executes the construct on that device.</td>
</tr>
<tr>
<td>omp target data</td>
<td>Specifies that variables are mapped to a device data environment for the extent of the region.</td>
</tr>
<tr>
<td>omp target enter data</td>
<td>Specifies that variables are mapped to a device data environment.</td>
</tr>
<tr>
<td>omp target exit data</td>
<td>Specifies that variables are unmapped from a device data environment.</td>
</tr>
<tr>
<td>omp target parallel loop</td>
<td>Provides an abbreviated way to specify a target region that contains only a parallel loop construct.</td>
</tr>
<tr>
<td>omp target teams</td>
<td>Creates a device data environment and executes the construct on the same device. It also creates a league of thread teams with the primary thread in each team executing the structured block.</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>omp target teams distribute</td>
<td>Creates a device data environment and then executes the construct on that device. It also specifies that loop iterations will be distributed among the primary threads of all thread teams in a league created by a teams construct.</td>
</tr>
<tr>
<td>omp target teams distribute parallel for</td>
<td>Creates a device data environment and then executes the construct on that device. It also specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams created by a teams construct.</td>
</tr>
<tr>
<td>omp target teams distribute parallel for simd</td>
<td>Creates a device data environment and then executes the construct on that device. It also specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams created by a teams construct. The loop will be distributed across the teams, which will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp target teams distribute simd</td>
<td>Creates a device data environment and then executes the construct on that device. It also specifies that loop iterations will be distributed among the primary threads of all thread teams in a league created by a teams construct. It will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp target teams loop</td>
<td>Provides an abbreviated way to specify a target region that contains only a teams loop construct.</td>
</tr>
<tr>
<td>omp target update</td>
<td>Makes the items listed in the device data environment consistent between the device and host, in accordance with the motion clauses on the pragma.</td>
</tr>
<tr>
<td>omp task</td>
<td>Specifies a code block whose execution may be deferred.</td>
</tr>
<tr>
<td>omp taskgroup</td>
<td>Causes the program to wait until the completion of all enclosed and descendant tasks.</td>
</tr>
<tr>
<td>omp taskwait</td>
<td>Specifies a wait on the completion of child tasks generated since the beginning of the current task.</td>
</tr>
<tr>
<td>omp taskyield</td>
<td>Specifies that the current task can be suspended at this point in favor of execution of a different task.</td>
</tr>
<tr>
<td>omp teams</td>
<td>Creates a league of thread teams inside a target region to execute a structured block in the initial thread of each team.</td>
</tr>
<tr>
<td>omp teams distribute</td>
<td>Creates a league of thread teams and specifies that loop iterations will be distributed among the primary threads of all thread teams in the league.</td>
</tr>
<tr>
<td>omp teams distribute parallel for</td>
<td>Creates a league of thread teams and specifies that the associated loop can be executed in parallel by multiple threads that are members of multiple teams.</td>
</tr>
<tr>
<td>omp teams distribute parallel for simd</td>
<td>Creates a league of thread teams and specifies that the associated loop can be executed concurrently using SIMD instructions in parallel by multiple threads that are members of multiple teams.</td>
</tr>
<tr>
<td>omp teams distribute simd</td>
<td>Creates a league of thread teams and specifies that the associated loop will be distributed across the primary threads of the teams and executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp teams loop</td>
<td>Provides an abbreviated way to specify a teams construct that contains only a loop construct.</td>
</tr>
</tbody>
</table>
Pragma threadprivate

Specifies a list of globally-visible variables that will be allocated private to each thread.

Pragmas Compatible with Other Compilers

The following pragmas are compatible with other compilers. For more information about these pragmas, see the documentation for the specified compiler.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_directory</td>
<td>HP-compatible pragma. It appends the string argument to the list of places to search for #include files.</td>
</tr>
<tr>
<td>poison</td>
<td>GCC-compatible pragma. It labels the identifiers you want removed from your program; an error results when compiling a &quot;poisoned&quot; identifier; #pragma POISON is also supported.</td>
</tr>
<tr>
<td>options</td>
<td>GCC-compatible pragma; It sets the alignment of fields in structures.</td>
</tr>
<tr>
<td>weak</td>
<td>GCC-compatible pragma, it declares the symbol you enter to be weak.</td>
</tr>
</tbody>
</table>

See Also

Intel-specific Pragmas
optimize compiler option
Zc compiler option

Error Handling

This section contains information about warnings and errors.

Warnings and Errors

This topic describes compiler warnings and errors. The compiler sends these messages, along with the erroneous source line, to stderr.

Warnings

Warning messages report legal but questionable use of C or C++. The compiler displays warnings by default. You can suppress warning messages by specifying an appropriate compiler option. Warnings do not stop translation or linking. Warnings do not interfere with any output files.

The following is a representative warning message:

unknown pragma ignored [-Wunknown-pragmas]

Some warnings that start with -W can be disabled using the negative form of the option -Wno-; for example, option -Wno-unknown-pragmas disables option -Wunknown-pragmas.

Errors

Error messages report syntactic or semantic misuse of C or C++. The compiler always displays error messages. Errors suppress object code for the module containing the error and prevent linking, but they allow parsing to continue to detect other possible errors.
The following are some representative error messages:

- expected ';' at end of declaration
- unexpected type name 'b': expected expression

For a summary of warning and error options, see:

This section contains information about features that can affect compilation, such as environment variables, and using configuration files.

**Supported Environment Variables**

You can customize your system environment by specifying paths where the compiler searches for certain files such as libraries, include files, configuration files, and certain settings.

**Compiler Compile-Time Environment Variables**

The following table shows the compile-time environment variables that affect the compiler:

<table>
<thead>
<tr>
<th>Compile-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL (Windows) <em>CL</em> (Windows)</td>
<td>Define the files and options you use most often with the CL variable. Note: You cannot set the CL environment variable to a string that contains an equal sign. You can use the pound sign instead. In the following example, the pound sign (#) is used as a substitute for an equal sign in the assigned string: <code>SET CL=/Dtest#100</code></td>
</tr>
<tr>
<td>IA32ROOT (IA-32 architecture and Intel® 64 architecture)</td>
<td>Points to the directories containing the include and library files for a non-standard installation structure. <strong>NOTE</strong> IA-32 is only available for C++; it is not available for DPC++.</td>
</tr>
<tr>
<td>ICXCFG</td>
<td>Specifies the configuration file for customizing compilations when invoking the compiler using <code>icx</code>. <strong>NOTE</strong> This environment variable is only available for C++; it is not available for DPC++.</td>
</tr>
<tr>
<td>ICPXCFG</td>
<td>Specifies the configuration file for customizing compilations when invoking the compiler using <code>icpx</code>.</td>
</tr>
<tr>
<td>Compile-Time Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>ICXCFG</strong></td>
<td>Specifies a configuration file, which the compiler should use instead of the default configuration file.</td>
</tr>
<tr>
<td><strong>__INTEL_PRE_CFLAGS AGS</strong></td>
<td>Specifies a set of compiler options to add to the compile line.</td>
</tr>
<tr>
<td><strong>__INTEL_POST_CFLAGS AGS</strong></td>
<td>This is an extension to the facility already provided in the compiler configuration file icx.cfg.</td>
</tr>
</tbody>
</table>

### NOTE

- By default, a configuration file named `icx.cfg` (Windows, Linux), or `icpx.cfg` (Linux) is used. This file is in the same directory as the compiler executable. To use another configuration file in another location, you can use the `ICXCFG` (Windows, Linux), `ICPXCFG` (Linux) environment variable to assign the directory and file name for the configuration file.

You can insert command line options in the prefix position using `__INTEL_PRE_CFLAGS`, or in the suffix position using `__INTEL_POST_CFLAGS`. The command line is built as follows:

**Syntax:**

```
icx <PRE flags> <flags from configuration file> <flags from the compiler invocation> <POST flags>
```

### NOTE

- The driver issues a warning that the compiler is overriding an option because of an environment variable, but only when you include the option `/W5` (Windows) or `-w3` (Linux).

<table>
<thead>
<tr>
<th><strong>PATH</strong></th>
<th>Specifies the directories the system searches for binary executable files.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TMP</strong></td>
<td>Specifies the location for temporary files. If none of these are specified, or writeable, or found, the compiler stores temporary files in <code>/tmp</code> (Linux) or the current directory (Windows).</td>
</tr>
<tr>
<td><strong>TMPDIR</strong></td>
<td>The compiler searches for these variables in the following order: <code>TMP</code>, <code>TMPDIR</code>, and <code>TEMP</code>.</td>
</tr>
<tr>
<td><strong>TEMP</strong></td>
<td>On Windows, these environment variables cannot be set from Visual Studio.</td>
</tr>
</tbody>
</table>

### NOTE

- On Windows, these environment variables cannot be set from Visual Studio.
<table>
<thead>
<tr>
<th>Compile-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LD_LIBRARY_PATH (Linux)</strong></td>
<td>Specifies the location for shared objects (.so files).</td>
</tr>
<tr>
<td><strong>INCLUDE (Windows)</strong></td>
<td>Specifies the directories for the source header files (include files).</td>
</tr>
<tr>
<td><strong>LIB (Windows)</strong></td>
<td>Specifies the directories for all libraries used by the compiler and linker.</td>
</tr>
<tr>
<td><strong>GNU Environment Variables and Extensions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CPATH (Linux)</strong></td>
<td>Specifies the path to include directory for C/C++ compilations.</td>
</tr>
<tr>
<td><strong>C_INCLUDE_PATH (Linux)</strong></td>
<td>Specifies path to include directory for C compilations.</td>
</tr>
<tr>
<td><strong>CPLUS_INCLUDE_PATH (Linux)</strong></td>
<td>Specifies path to include directory for C++ compilations.</td>
</tr>
<tr>
<td><strong>DEPENDENCIES_OUTPUT (Linux)</strong></td>
<td>Specifies how to output dependencies for make based on the non-system header files processed by the compiler. System header files are ignored in the dependency output.</td>
</tr>
<tr>
<td><strong>GCC_EXEC_PREFIX (Linux)</strong></td>
<td>Specifies alternative names for the linker (<code>ld</code>) and assembler (<code>as</code>).</td>
</tr>
<tr>
<td><strong>GCCROOT (Linux)</strong></td>
<td>Specifies the location of the GCC* binaries.</td>
</tr>
</tbody>
</table>

*NOTE* This environment variable is only available for C++; it is not available for DPC++. |

| **GCC_ROOT (Linux)**             | Specifies the location of the GCC headers. |

*NOTE* This environment variable is only available for C++; it is not available for DPC++. |

| **GXX_INCLDE_ROOT (Linux)**      | Specifies the location of the GCC binaries. |

*NOTE* This environment variable is only available for C++; it is not available for DPC++. |

| **LIBRARY_PATH (Linux)**         | Specifies the path for libraries to be used during the link phase. |
| **SUNPRO_DEPENDENCIES_CIES (Linux)** | This variable is the same as **DEPENDENCIES_OUTPUT**, except that system header files are not ignored. |
NOTE INTEL_ROOT is an environment variable that is reserved for the Intel® Compiler. Its use is not supported.

Compiler Run-Time Environment Variables

NOTE The compiler run-time environment variables are only available for C++; they are not available for DPC++.

The following table summarizes compiler environment variables that are recognized at run time.

<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em><em>GNU extensions (recognized by the Intel OpenMP</em> compatibility library)</em>*</td>
<td></td>
</tr>
<tr>
<td>GOMP_CPU_AFFINITY (Linux)</td>
<td>GNU extension recognized by the Intel OpenMP compatibility library. Specifies a list of OS processor IDs. You must set this environment variable before the first parallel region or before certain API calls including <code>omp_get_max_threads()</code>, <code>omp_get_num_procs()</code> and any affinity API calls. For detailed information on this environment variable, see Thread Affinity Interface.</td>
</tr>
<tr>
<td><strong>Default:</strong> Affinity is disabled</td>
<td></td>
</tr>
<tr>
<td>GOMP_STACKSIZE (Linux)</td>
<td>GNU extension recognized by the Intel OpenMP compatibility library. Same as <code>OMP_STACKSIZE.KMP_STACKSIZE overrides GOMP_STACKSIZE</code>, which overrides <code>OMP_STACKSIZE</code>.</td>
</tr>
<tr>
<td><strong>Default:</strong> See the description for <code>OMP_STACKSIZE</code>.</td>
<td></td>
</tr>
<tr>
<td><strong>OpenMP Environment Variables (OMP_) and Extensions (KMP_)</strong></td>
<td></td>
</tr>
<tr>
<td>OMP_CANCELLATION</td>
<td>Activates cancellation of the innermost enclosing region of the type specified. If set to <code>TRUE</code>, the effects of the cancel construct and of cancellation points are enabled and cancellation is activated. If set to <code>FALSE</code>, cancellation is disabled and the cancel construct and cancellation points are effectively ignored.</td>
</tr>
<tr>
<td><strong>NOTE</strong></td>
<td>Internal barrier code will work differently depending on whether the cancellation is enabled. Barrier code should repeatedly check the global flag to figure out if the cancellation had been triggered. If a thread observes the cancellation it should leave the barrier prematurely with the return value 1 (may wake up other threads). Otherwise, it should leave the barrier with the return value 0.</td>
</tr>
<tr>
<td>Run-Time Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>OMP_CANCELLATION</td>
<td>Enables (TRUE) or disables (FALSE) cancellation of the innermost enclosing region of the type specified.</td>
</tr>
</tbody>
</table>
|                              | **Default:** FALSE  
|                              | **Example:** OMP_CANCELLATION=TRUE |
| OMP_DISPLAY_ENV              | Enables (TRUE) or disables (FALSE) the printing to stderr of the OpenMP version number and the values associated with the OpenMP environment variable. |
|                              | Possible values are: TRUE, FALSE, or VERBOSE. |
|                              | **Default:** FALSE  
|                              | **Example:** OMP_DISPLAY_ENV=TRUE |
| OMP_DEFAULT_DEVICE           | Sets the device that will be used in a target region. The OpenMP routine omp_set_default_device or a device clause in a target pragma can override this variable. |
|                              | If no device with the specified device number exists, the code is executed on the host. If this environment variable is not set, device number 0 is used. |
| OMP_DYNAMIC                  | Enables (TRUE) or disables (FALSE) the dynamic adjustment of the number of threads. |
|                              | **Default:** FALSE  
|                              | **Example:** OMP_DYNAMIC=TRUE |
| OMP_MAX_ACTIVE_LEVELS        | The maximum number of levels of parallel nesting for the program. |
|                              | Possible values: Non-negative integer. |
|                              | **Default:** 1 |
| OMP_NESTED                   | Deprecated; use OMP_MAX_ACTIVE_LEVELS instead. |
| OMP_NUM_THREADS              | Sets the maximum number of threads to use for OpenMP parallel regions if no other value is specified in the application. |
|                              | The value can be a single integer, in which case it specifies the number of threads for all parallel regions. The value can also be a comma-separated list of integers, in which case each integer specifies the number of threads for a parallel region at a nesting level. |
|                              | The first position in the list represents the outermost parallel nesting level, the second position represents the next-inner parallel nesting level, and so on. At any level, the integer can be left out of
### Run-Time Environment Variable

<table>
<thead>
<tr>
<th>OMP_PLACES</th>
</tr>
</thead>
</table>

**Description**

the list. If the first integer in a list is left out, it implies the normal default value for threads is used at the outer-most level. If the integer is left out of any other level, the number of threads for that level is inherited from the previous level.

This environment variable applies to the options `Qopenmp` (Windows) or `qopenmp` (Linux).

**Default:** The number of processors visible to the operating system on which the program is executed.

**Syntax:** `OMP_NUM_THREADS=value[,value]`*

Specifies an explicit ordered list of places, either as an abstract name describing a set of places or as an explicit list of places described by nonnegative numbers. An exclusion operator "!" can also be used to exclude the number or place immediately following the operator.

For **explicit lists**, the meaning of the numbers and how the numbering is done for a list of nonnegative numbers are implementation defined. Generally, the numbers represent the smallest unit of execution exposed by the execution environment, typically a hardware thread.

Intervals can be specified using the `<lower-bound> : <length> : <stride>` notation to represent the following list of numbers:

"<lower-bound>, <lower-bound> + <stride>,...,<lower-bound> + (<length>-1)*<stride>."

When `<stride>` is omitted, a unit stride is assumed. Intervals can specify numbers within a place as well as sequences of places.

```bash
# EXPLICIT LIST EXAMPLE
setenv OMP_PLACES "{0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}"
setenv OMP_PLACES "{0:4},{4:4},{8:4},{12:4}"
setenv OMP_PLACES "{0:4}:4:4"
```

The **abstract names** listed below should be understood by the execution and run-time environment:

- **threads:** Each place corresponds to a single hardware thread on the target machine.
- **cores:** Each place corresponds to a single core (having one or more hardware threads) on the target machine.
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• ll_caches:</td>
<td>Each place corresponds to a set of cores that share the last level cache on the device.</td>
</tr>
<tr>
<td>• numa_domains:</td>
<td>Each place corresponds to a set of cores for which their closest memory on the device is 1) the same memory and 2) at a similar distance from the cores.</td>
</tr>
<tr>
<td>• sockets:</td>
<td>Each place corresponds to a single socket (consisting of one or more cores) on the target machine.</td>
</tr>
</tbody>
</table>

When requesting fewer places or more resources than available on the system, the determination of which resources of type `abstract_name` are to be included in the place list is implementation-defined. The precise definitions of the abstract names are implementation defined. An implementation may also add abstract names as appropriate for the target platform. The abstract name may be appended by a positive number in parentheses to denote the length of the place list to be created, that is `abstract_name(num-places)`.

```
# ABSTRACT NAMES EXAMPLE
setenv OMP_PLACES threads
setenv OMP_PLACES threads(4)
```

**NOTE**

If any numerical values cannot be mapped to a processor on the target platform the behavior is implementation-defined. The behavior is also implementation-defined when the `OMP_PLACES` environment variable is defined using an abstract name.

**OMP_PROC_BIND (Windows, Linux)**

Sets the thread affinity policy to be used for parallel regions at the corresponding nested level. Enables (TRUE) or disables (FALSE) the binding of threads to processor contexts. If enabled, this is the same as specifying `KMP_AFFINITY=scatter`. If disabled, this is the same as specifying `KMP_AFFINITY=none`.

**Acceptable values:** TRUE, FALSE, or a comma separated list, each element of which is one of the following values: PRIMARY, MASTER (deprecated), CLOSE, SPREAD.

**Default:** FALSE

If set to FALSE, the execution environment may move OpenMP threads between OpenMP places, thread affinity is disabled, and proc_bind clauses on
Run-Time Environment Variable | Description
---|---

parallel constructs are ignored. Otherwise, the execution environment should not move OpenMP threads between OpenMP places, thread affinity is enabled, and the initial thread is bound to the first place in the OpenMP place list.

If set to PRIMARY, all threads are bound to the same place as the primary thread. If set to CLOSE, threads are bound to successive places, close to where the primary thread is bound. If set to SPREAD, the primary thread’s partition is subdivided and threads are bound to single place successive sub-partitions.

**NOTE**

KMP_AFFINITY takes precedence over GOMP_CPU_AFFINITY and OMP_PROC_BIND.
GOMP_CPU_AFFINITY takes precedence over OMP_PROC_BIND.

**OMP_SCHEDULE**

Sets the run-time schedule type and an optional chunk size.

**Default:** static, no chunk size specified

**Example syntax:**

OMP_SCHEDULE="[modifier:]kind[,chunk_size]" where

- modifier is one of monotonic or nonmonotonic
- kind is one of static, dynamic, guided, or auto
- chunk_size is a positive integer

**NOTE**

Some environment variables are available for both Intel® microprocessors and non-Intel microprocessors, but may perform additional optimizations for Intel® microprocessors than for non-Intel microprocessors.

**OMP_STACKSIZE**

Sets the number of bytes to allocate for each OpenMP thread to use as the private stack for the thread. Recommended size is 16M.

Use the optional suffixes to specify byte units: B (bytes), K (Kilobytes), M (Megabytes), G (Gigabytes), or T (Terabytes) to specify the units. If you specify a value without a suffix, the byte unit is assumed to be K (Kilobytes).
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OMP_THREAD_LIMIT</strong></td>
<td>Limits the number of simultaneously-executing threads in an OpenMP program.</td>
</tr>
<tr>
<td></td>
<td>If this limit is reached and another native operating system thread encounters OpenMP API calls or constructs, the program can abort with an error message. If this limit is reached when an OpenMP parallel region begins, a one-time warning message might be generated indicating that the number of threads in the team was reduced, but the program will continue.</td>
</tr>
<tr>
<td></td>
<td>This environment variable is only used for programs compiled with the following option: Qopenmp (Windows) or openmp (Linux).</td>
</tr>
<tr>
<td></td>
<td>The omp_get_thread_limit() routine returns the value of the limit.</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> No enforced limit</td>
</tr>
<tr>
<td></td>
<td><strong>Related environment variable:</strong> KMP_ALL_THREADS (overrides OMP_THREAD_LIMIT).</td>
</tr>
<tr>
<td></td>
<td><strong>Example syntax:</strong> OMP_THREAD_LIMIT=value</td>
</tr>
<tr>
<td><strong>OMP_WAIT_POLICY</strong></td>
<td>Decides whether threads spin (active) or yield (passive) while they are waiting.</td>
</tr>
<tr>
<td></td>
<td>OMP_WAIT_POLICY=ACTIVE is an alias for KMP_LIBRARY=turnaround, and OMP_WAIT_POLICY=PASSIVE is an alias for KMP_LIBRARY=throughput.</td>
</tr>
<tr>
<td></td>
<td><strong>Default:</strong> Passive</td>
</tr>
<tr>
<td></td>
<td><strong>Syntax:</strong> OMP_WAIT_POLICY=value</td>
</tr>
</tbody>
</table>

This variable does not affect the native operating system threads created by the user program, or the thread executing the sequential part of an OpenMP program.

The kmp_(set, get)_stacksize_s() routines set/retrieve the value. The kmp_set_stacksize_s() routine must be called from sequential part, before first parallel region is created. Otherwise, calling kmp_set_stacksize_s() has no effect. **Default (IA-32 architecture):** 2M

**Default (Intel® 64 architecture):** 4M

**Related environment variables:**
KMP_STACKSIZE (overrides OMP_STACKSIZE).

**Syntax:** OMP_STACKSIZE=value
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| OMP_DISPLAY_AFFINITY | Instructs the runtime to display formatted affinity information for all OpenMP threads in the parallel region upon entering the first parallel region and when any change occurs in the information accessible by the format specifiers listed in the OMP_AFFINITY_FORMAT entry.  
Possible values: TRUE or FALSE  
Default: FALSE |
| OMP_AFFINITY_FORMAT | Defines the format when displaying OpenMP thread affinity information. Possible values are any string with the following format field available:  
- %t or %{team_num}: Value returned by omp_get_team_num()  
- %T or %{num_teams}: Value returned by omp_get_num_teams()  
- %L or %{nesting_level}: Value returned by omp_get_level()  
- %n or %{thread_num}: Value returned by omp_get_thread_num()  
- %a or %{ancestor_tnum}: Value returned by omp_get_ancestor_thread_num(omp_get_level() – 1)  
- %H or %{host}: Name of host device  
- %P or %{process_id}: Process ID  
- %i or %{native_thread_id}: Native thread ID on the platform  
- %A or %{thread_affinity}: List of processor ID on which a thread may execute  
Default: 'OMP: pid %P tid %i thread %n bound to OS proc set {%A}' |
| OMP_MAX_TASK_PRIORITY | Controls the use of task priorities by setting the initial value.  
Possible values: Non-negative integer.  
Default: 0 |
| OMP_TOOL | Controls whether the OpenMP runtime will try to register a first party tool that uses OMPT interface.  
Possible values: ENABLED or DISABLED.  
Default: ENABLED |

**NOTE** Only the host OpenMP runtime is supported.
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| OMP_TOOL_LIBRARIES | Sets a list of first-party tool locations that use the OMPT interface. The list enumerates names of dynamically-loadable libraries with OS-specific path separator.  
**Default**: Empty  
**NOTE** Only the host OpenMP runtime is supported. |
| OMP_TOOL_VERBOSE_INIT | Controls whether the OpenMP runtime will verbosely log the registration of a tool that uses the OMPT interface.  
Possible values:  
- **DISABLED**: Do not log the registration.  
- **STDOUT**: Log the registration to stdout.  
- **STDERR**: Log the registration to stderr.  
- **File_Name**: Log the registration to the location specified by File_Name.  
**Default**: **DISABLED**  
**NOTE** Only the host OpenMP runtime is supported. |
| OMP_DEBUG | Controls whether the OpenMP runtime collects information that an OMPD library may need to support a tool.  
**Possible values**: **ENABLED** or **DISABLED**.  
**Default**: **DISABLED**  
**NOTE** Only the host OpenMP runtime is supported. |
| OMP_ALLOCATOR | Specifies the default allocator for allocation calls, directives, and clauses that do not specify an allocator.  
**Default**: **omp_default_mem_alloc**  
**Syntax**: `<PredefinedMemAllocator> | <PredefinedMemSpace> | <PredefinedMemSpace>:<Traits>`  
Currently supported values for `<PredefinedMemAllocator>` and `<PredefinedMemSpace>` :  
- **omp_default_mem_alloc and omp_default_mem_space**  
Additional values are supported if `libmemkind` is available and there is system support for it: |
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OMP_NUM_TEAMS</strong></td>
<td>Sets the maximum number of teams created by a teams construct by setting nteams-var ICV. Possible values: Positive integer. <strong>Default:</strong> 1</td>
</tr>
<tr>
<td><strong>OMP_TEAMS_THREAD_LIMIT</strong></td>
<td>Sets the maximum number of OpenMP threads to use in each team created by a teams construct. Possible values: Positive integer. <strong>Default:</strong> &lt;NumberOfProcessors&gt; / &lt;nteams-var ICV&gt;</td>
</tr>
<tr>
<td><strong>KMP_AFFINITY</strong> <em>(Windows, Linux)</em></td>
<td>Enables run-time library to bind threads to physical processing units. You must set this environment variable before the first parallel region, or certain API calls including <code>omp_get_max_threads()</code>, <code>omp_get_num_procs()</code> and any affinity API calls. For detailed information on this environment variable, see <em>Thread Affinity Interface</em>. <strong>Default:</strong> noverbose,warnings,respect,granularity=core,none Default (Windows with multiple processor groups): noverbose,warnings,norespect,granularity=group,compact,0,0</td>
</tr>
<tr>
<td><strong>KMP_ALL_THREADS</strong></td>
<td>Limits the number of simultaneously-executing threads in an OpenMP program. If this limit is reached and another native operating system thread encounters OpenMP API calls or constructs, then the program may abort with an error message. If this limit is reached at the time an OpenMP parallel region begins, a one-time warning message may be generated indicating that the number of threads in the team was reduced, but the program will continue execution.</td>
</tr>
</tbody>
</table>

Refer to the [OpenMP specification](https://openmp.org/wp/spec-5.2/) for more information.

**NOTE** On Windows with multiple processor groups, the norespect affinity modifier is assumed when the process affinity mask equals a single processor group (which is default on Windows). Otherwise, the respect affinity modifier is used.
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| KMP_BLOCKTIME                 | **This environment variable is only used for programs compiled with the Qopenmp (Windows) or qopenmp (Linux) option.**
<p>|                               | <strong>Default:</strong> No enforced limit. |
|                               | Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping. |
|                               | Use the optional character suffixes: s (seconds), m (minutes), h (hours), or d (days) to specify the units. |
|                               | Specify <strong>infinite</strong> for an unlimited wait time. |
|                               | <strong>Default:</strong> 200 milliseconds |
|                               | <strong>Related Environment Variable:</strong> KMP_LIBRARY environment variable. |
| KMP_CPUINFO_FILE              | Specifies an alternate file name for a file containing the machine topology description. The file must be in the same format as /proc/cpuinfo. |
|                               | <strong>Default:</strong> None |
| KMP_DETERMINISTIC_REDUCTION   | Enables (TRUE) or disables (FALSE) the use of a specific ordering of the reduction operations for implementing the reduction clause for an OpenMP parallel region. This has the effect that, for a given number of threads, in a given parallel region, for a given data set and reduction operation, a floating point reduction done for an OpenMP reduction clause has a consistent floating point result from run to run, since round-off errors are identical. |
|                               | <strong>NOTE</strong> When compiling, you must set the following flag to ensure correct behavior: |
|                               | • -fp-model precise (Linux) |
|                               | • -fp:precise (Windows) |
|                               | <strong>Default:</strong> FALSE |
| KMP_DYNAMIC_MODE              | Selects the method used to determine the number of threads to use for a parallel region when OMP_DYNAMIC=TRUE. <strong>Possible values:</strong> (asat | load_balance | thread_limit), where, |
|                               | • asat: estimates number of threads based on parallel start time; |</p>
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOTE</strong></td>
<td>Support for <code>asat</code> (automatic self-allocating threads) is now deprecated and will be removed in a future release.</td>
</tr>
<tr>
<td><strong>load_balance</strong></td>
<td>tries to avoid using more threads than available execution units on the machine;</td>
</tr>
<tr>
<td><strong>thread_limit</strong></td>
<td>tries to avoid using more threads than total execution units on the machine.</td>
</tr>
<tr>
<td><strong>Default (IA-32 architecture):</strong></td>
<td><code>load_balance</code> (on all supported OSes)</td>
</tr>
<tr>
<td><strong>Default (Intel® 64 architecture):</strong></td>
<td><code>load_balance</code> (on all supported OSes)</td>
</tr>
<tr>
<td><strong>KMP_HOT_TEAMS_MAX_LEVEL</strong></td>
<td>Sets the maximum nested level to which teams of threads will be hot.</td>
</tr>
<tr>
<td><strong>NOTE</strong></td>
<td>A hot team is a team of threads optimized for faster reuse by subsequent parallel regions. In a hot team, threads are kept ready for execution of the next parallel region, in contrast to the cold team, which is freed after each parallel region, with its threads going into a common pool of threads.</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>KMP_HOT_TEAMS_MODE</strong></td>
<td>Specifies the run-time behavior when the number of threads in a hot team is reduced.</td>
</tr>
<tr>
<td><strong>Possible values:</strong></td>
<td>0: Extra threads are freed and put into a common pool of threads.</td>
</tr>
<tr>
<td></td>
<td>1: Extra threads are kept in the team in reserve, for faster reuse in subsequent parallel regions.</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>0</td>
</tr>
</tbody>
</table>
| **KMP_HW_SUBSET**            | Specifies the subset of available hardware resources for the hardware topology hierarchy. The subset is specified in terms of number of units per upper layer unit starting from top layer downwards. For example, the number of sockets (top layer units), cores per socket, and the threads per core, to use with an OpenMP application, as an alternative to writing complicated explicit affinity
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| settings or a limiting process affinity mask. You can also specify an offset value to set which resources to use. When available, you can specify attributes to select different subsets of resources. An extended syntax is available when KMP_TOPOLOGY_METHOD=hwloc. Depending on what resources are detected, you may be able to specify additional resources, such as NUMA nodes and groups of hardware resources that share certain cache levels.  

**Basic syntax:**

\[
\text{num\_unitsID[@offset][:attribute]} \\
[,\text{num\_unitsID[@offset][:attribute]}...]
\]

Supported unit IDs are not case-sensitive.

- **S - socket**  
  \(\text{num\_units}\) specifies the requested number of sockets.

- **D - die**  
  \(\text{num\_units}\) specifies the requested number of dies per socket.

- **C - core**  
  \(\text{num\_units}\) specifies the requested number of cores per die - if any - otherwise, per socket.

- **T - thread**  
  \(\text{num\_units}\) specifies the requested number of HW threads per core.

- **offset** (Optional) The number of units to skip.

- **attribute** (Optional) An attribute differentiating resources at a particular level. This is only available for the core level on machines with Intel® Hybrid Technology. The attributes available to users are:
  
  - **Core type:** Either \text{intel\_atom} or \text{intel\_core}
  
  - **Core efficiency:** Specified as \text{effnum} where \(\text{num}\) is a number from 0 to the number of core efficiencies detected in the machine topology minus one. For example: \text{eff0}. The greater the efficiency number the more performant the core. There may be more core efficiencies than core types and can be viewed by setting KMP_AFFINITY=verbose.
Run-Time Environment Variable | Description
---|---

**NOTE** The hardware cache can be specified as a unit, for example L2 for L2 cache, or LL for last level cache.

**Extended syntax when**

*KMP_TOPOLOGY_METHOD=hwloc:*

Additional IDs can be specified if detected. For example:

- **N - numa**  
  `num_units` specifies the requested number of NUMA nodes per upper layer unit, e.g. per socket.
- **TI - tile**  
  `num_units` specifies the requested number of tiles to use per upper layer unit, e.g. per NUMA node.

When any `numa` or `tile` units are specified in `KMP_HW_SUBSET`, the `KMP_TOPOLOGY_METHOD` will be automatically set to `hwloc`, so there is no need to set it explicitly.

If you don't specify one or more types of resource, such as socket or thread, all available resources of that type are used.

The run-time library prints a warning, and the setting of `KMP_HW_SUBSET` is ignored if:

- a resource is specified, but detection of that resource is not supported by the chosen topology detection method and/or
- a resource is specified twice. An exception to this condition is if attributes differentiate the resource.
- attributes are used when unavailable, not detected in the machine topology, or conflict with each other.

This variable does not work if the OpenMP affinity is set to disabled.

**Default:** If omitted, the default value is to use all the available hardware resources.

**Examples:**

- **2s,4c,2t:** Use the first 2 sockets (s0 and s1), the first 4 cores on each socket (c0 - c3), and 2 threads per core.
- **2s@2,4c@8,2t:** Skip the first 2 sockets (s0 and s1) and use 2 sockets (s2-s3), skip the first 8 cores (c0-c7) and use 4 cores on each socket (c8-c11), and use 2 threads per core.
### Run-Time Environment Variable

<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 5C@1,3T: Use all available sockets, skip the first core and use 5 cores, and use 3 threads per core.</td>
<td></td>
</tr>
<tr>
<td>• 1T: Use all cores on all sockets, 1 thread per core.</td>
<td></td>
</tr>
<tr>
<td>• 1s, 1d, 1n, 1c, 1t: Use 1 socket, 1 die, 1 NUMA node, 1 core, 1 thread - use HW thread as a result.</td>
<td></td>
</tr>
<tr>
<td>• 4c:intel_atom,5c:intel_core: Use all available sockets and use 4 Intel Atom® processor cores and 5 Intel® Core™ processor cores per socket.</td>
<td></td>
</tr>
<tr>
<td>• 2c:eff0,3c:eff1: Use all available sockets and use 2 cores with efficiency 0 and 3 cores with efficiency 1 per socket.</td>
<td></td>
</tr>
<tr>
<td>• 1s, 1c, 1t: Use 1 socket, 1 core, 1 thread. This may result in using single thread on a 3-layer topology architecture, or multiple threads on 4-layer or 5-layer architecture. Result may even be different on the same architecture, depending on KMP_TOPOLOGY_METHOD specified, as hwloc can often detect more topology layers than default method used by the OpenMP runtime library.</td>
<td></td>
</tr>
</tbody>
</table>

To see the result of the setting, you can specify `verbose` modifier in `KMP_AFFINITY` environment variable. The OpenMP run-time library will output to stderr stream the information about discovered HW topology before and after the `KMP_HW_SUBSET` setting was setting applied. For example, on Intel® Xeon Phi™ 7210 cpu in SNC-4 Clustering Mode, the setting `KMP_AFFINITY=verbose KMP_HW_SUBSET=1N,1L2,1L1,1T` outputs various verbose information to stderr, including the following lines about discovered HW topology before and after `KMP_HW_SUBSET` was applied:

- Info #191: `KMP_AFFINITY`: 1 socket x 4 NUMA domains/socket x 8 tiles/NUMA domain x 2 cores/tile x 4 threads/core. (64 total cores)
- Info #191: `KMP_HW_SUBSET` 1 socket x 1 NUMA domain/socket x 1 tile/NUMA domain x 1 core/tile x 1 thread/core (1 total cores)

**KMP_INHERIT_FP_CONTROL**

Enables (TRUE) or disables (FALSE) the copying of the floating-point control settings of the primary thread to the floating-point control settings of the OpenMP worker threads at the start of each parallel region.

**Default:** TRUE
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KMP_LIBRARY</strong></td>
<td>Selects the OpenMP run-time library execution mode. The values for this variable are <em>serial</em>, <em>turnaround</em>, or <em>throughput</em>.</td>
</tr>
<tr>
<td><strong>Default:</strong> throughput</td>
<td></td>
</tr>
<tr>
<td><strong>KMP_PLACE_THREADS</strong></td>
<td>Deprecated; use KMP_HW_SUBSET instead.</td>
</tr>
<tr>
<td><strong>KMP_SETTINGS</strong></td>
<td>Enables (TRUE) or disables (FALSE) the printing of OpenMP run-time library environment variables during program execution. Two lists of variables are printed: user-defined environment variables settings and effective values of variables used by OpenMP run-time library.</td>
</tr>
<tr>
<td><strong>Default:</strong> FALSE</td>
<td></td>
</tr>
<tr>
<td><strong>KMP_STACKSIZE</strong></td>
<td>Sets the number of bytes to allocate for each OpenMP thread to use as its private stack.</td>
</tr>
<tr>
<td><strong>Default (IA-32 architecture):</strong> 2m</td>
<td>Recommended size is 16m.</td>
</tr>
<tr>
<td></td>
<td>Use the optional suffixes to specify byte units: B (bytes), K (Kilobytes), M (Megabytes), G (Gigabytes), or T (Terabytes) to specify the units. If you specify a value without a suffix, the byte unit is assumed to be K (Kilobytes).</td>
</tr>
<tr>
<td></td>
<td><strong>KMP_STACKSIZE overrides GOMP_STACKSIZE, which overrides OMP_STACKSIZE. Default (IA-32 architecture): 2m</strong></td>
</tr>
<tr>
<td><strong>KMP_TOPOLOGY_METHOD</strong></td>
<td>Forces OpenMP to use a particular machine topology modeling method.</td>
</tr>
<tr>
<td></td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• all</td>
</tr>
<tr>
<td></td>
<td>Let OpenMP choose which topology method is most appropriate based on the platform and possibly other environment variable settings.</td>
</tr>
<tr>
<td></td>
<td>• cpuid_leaf11</td>
</tr>
<tr>
<td></td>
<td>Decodes the APIC identifiers as specified by leaf 11 of the <em>cpuid</em> instruction.</td>
</tr>
<tr>
<td></td>
<td>• cpuid_leaf4</td>
</tr>
<tr>
<td></td>
<td>Decodes the APIC identifiers as specified in leaf 4 of the <em>cpuid</em> instruction.</td>
</tr>
<tr>
<td></td>
<td>• cpuinfo</td>
</tr>
<tr>
<td></td>
<td>If KMP_CPUINFO_FILE is not specified, forces OpenMP to parse /proc/cpuinfo to determine the topology (Linux only).</td>
</tr>
<tr>
<td>Run-Time Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>KMP_CPUINFO_FILE</strong></td>
<td>If <code>KMP_CPUINFO_FILE</code> is specified as described above, uses it (Windows or Linux).</td>
</tr>
<tr>
<td>• <strong>group</strong></td>
<td>Models the machine as a 2-level map, with level 0 specifying the different processors in a group, and level 1 specifying the different groups (Windows 64-bit only).</td>
</tr>
</tbody>
</table>

**NOTE**  
Support for `group` is now deprecated and will be removed in a future release. Use `all` instead.

• **flat**  
Models the machine as a flat (linear) list of processors.
• **hwloc**  
Models the machine as the Portable Hardware Locality* (hwloc) library does. This model is the most detailed and includes, but is not limited to: numa nodes, packages, cores, hardware threads, caches, and Windows processor groups.  

**Default:** `all`

| KMP_USER_LEVEL_MWAIT | Enables (TRUE) or disables (FALSE) the use of user-level mwait as alternative to putting waiting threads to sleep, if available, either from ring3 or `WAITPKG`.  
**Default:** `FALSE` |
|----------------------|-------------|

| KMP_VERSION | Enables (TRUE) or disables (FALSE) the printing of OpenMP run-time library version information during program execution.  
**Default:** `FALSE` |
|-------------|-------------|

| KMP_WARNINGS | Enables (TRUE) or disables (FALSE) displaying warnings from the OpenMP run-time library during program execution.  
**Default:** `TRUE` |
|--------------|-------------|

**OpenMP Offload Environment Variables (OMP_, LIBOMPTARGET)**

| OMP_TARGET_OFFLOAD | Controls the program behavior when offloading a target region.  
**Possible values:**  
• **MANDATORY**  
Program execution is terminated if a device construct or device memory routine is encountered and the device is not available or is not supported. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-Time Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DISABLED</td>
<td>Disables target offloading to devices and execution occurs on the host.</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Target offloading is enabled if the device is available and supported.</td>
</tr>
</tbody>
</table>

**Default:** DEFAULT

**LIBOMPTARGET_DEBUG**

Controls whether debugging information will be displayed from the offload runtime.

Possible values:

- 0
  - Disabled.
- 1
  - Displays basic debug information from the plugin actions such as device detection, kernel compilation, memory copy operations, kernel invocations, and other plugin-dependent actions.
- 2
  - Displays which GPU runtime API functions are invoked with which arguments and parameters in addition to the information displayed with value 1.

**Default:** 0

**LIBOMPTARGET_INFO**

Controls whether basic offloading information will be displayed from the offload runtime.

Possible values:

- 0
  - Disabled.
- 1
  - Prints all data arguments upon entering an OpenMP device kernel.
- 2
  - Indicates when a mapped address already exists in the device mapping table.
- 4
  - Dump the contents of the device pointer map if target offloading fails.
- 8
  - Indicates when an entry is changed in the device mapping table.
- 32
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-Time Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>LIBOMPTARGET_PLUGIN</td>
<td>Specifies which offload plugin is used when offloading a target region.</td>
</tr>
<tr>
<td>Possible values:</td>
<td></td>
</tr>
<tr>
<td>• LEVEL_ZERO</td>
<td>LEVEL0</td>
</tr>
<tr>
<td>• OPENCL</td>
<td>opencl</td>
</tr>
<tr>
<td>• X86_64</td>
<td>x86_64</td>
</tr>
<tr>
<td>Default: LEVEL_ZERO</td>
<td></td>
</tr>
<tr>
<td>LIBOMPTARGET_DEVICETYPE</td>
<td>Selects device type to which a target region is offloaded.</td>
</tr>
<tr>
<td>Possible values:</td>
<td></td>
</tr>
<tr>
<td>• GPU</td>
<td>gpu</td>
</tr>
<tr>
<td>• CPU</td>
<td>cpu</td>
</tr>
<tr>
<td>Offload plugin support for device type:</td>
<td></td>
</tr>
<tr>
<td>• Level Zero offload plugin only supports GPU type.</td>
<td></td>
</tr>
<tr>
<td>• OpenCL offload plugin supports both GPU and CPU types.</td>
<td></td>
</tr>
<tr>
<td>• X86_64 offload plugin ignores this variable.</td>
<td></td>
</tr>
<tr>
<td>Default: GPU</td>
<td></td>
</tr>
<tr>
<td>LIBOMPTARGET_PLUGIN_PROFILE</td>
<td>Enables basic plugin profiling and displays the result when program finishes.</td>
</tr>
<tr>
<td>Default: Disabled</td>
<td></td>
</tr>
<tr>
<td>Syntax: &lt;Value&gt;[,usec], where &lt;Value&gt;=1</td>
<td>T</td>
</tr>
<tr>
<td>The unit of reported time is microsecond if ”,usec” is appended, millisecond otherwise.</td>
<td></td>
</tr>
<tr>
<td>LIBOMPTARGET_DYNAMIC_MEMORY_SIZE</td>
<td>Sets the size of preallocated memory in MB to service malloc calls on the device.</td>
</tr>
<tr>
<td>Currently, calls to free memory on the device do not release memory. All the allocated memory is released only when the program exits.</td>
<td>Possible values: Non-negative integer.</td>
</tr>
<tr>
<td>Run-Time Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Default:</strong> 0</td>
<td></td>
</tr>
<tr>
<td>LIBOMPTARGET_LEVEL0_COMPILE_OPTIONS</td>
<td>Passes extra build options when building native target program binaries. Possible values: Valid Level Zero build options.</td>
</tr>
<tr>
<td>LIBOMPTARGET_DEVICES</td>
<td>Controls how subdevices or sub-subdevices are exposed to users if device supports subdevices. Possible values:</td>
</tr>
<tr>
<td>• DEVICE</td>
<td>device Only top-level devices are reported as OpenMP devices and subdevice clause is supported.</td>
</tr>
<tr>
<td>• SUBDEVICE</td>
<td>subdevice Only first-level subdevices are reported as OpenMP devices and subdevice clause is ignored.</td>
</tr>
<tr>
<td>• SUBSUBDEVICE</td>
<td>subsubdevice Only second-level subdevices are reported as OpenMP devices and subdevice clause is ignored.</td>
</tr>
<tr>
<td>• ALL</td>
<td>all All devices and subdevices are reported as OpenMP devices and subdevice clause is ignored. <strong>Default:</strong> DEVICE</td>
</tr>
<tr>
<td>LIBOMPTARGET_LEVEL0_MEMORY_POOL</td>
<td>Controls memory pool configuration. Possible values:</td>
</tr>
<tr>
<td>• 0</td>
<td>Disables using memory pool.</td>
</tr>
</tbody>
</table>
| • <PoolInfoList>=<PoolInfo>[,<PoolInfoList>] |<PoolInfo>=<MemType>[,<AllocMax>[,<Capacity>[,<PoolSize>]]] <MemType>=all | device | host | shared <AllocMax> is a positive integer or empty <Capacity> is a positive integer or empty <PoolSize> is a positive integer or empty <AllocMax> means maximum allocation size in MB supported by the pool, <Capacity> means number of allocations supported by a single
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| `LIBOMPTARGET_LEVEL0_USE_COPY_ENGINE` | Disables use of copy engine for memory transfers. Possible values:  
  - 0 | F | f  
    Disables use of copy engine.  
  **Default:** Enabled |
| `LIBOMPTARGET_LEVEL0_DEFAULT_TARGET_MEM` | Selects memory type returned by the `omp_target_alloc` routine. Possible values:  
  - DEVICE | device  
    Returned memory type is device type. Device owns the memory and data movement is explicit.  
  - SHARED | shared  
    Returned memory type is shared type. Ownership of the memory is shared between host and device, and data movement is implicit.  
  - HOST | host  
    Returned memory type is host type. Host owns the memory and data movement is implicit.  
  **Default:** DEVICE |
| `LIBOMPTARGET_LEVEL0_STAGING_BUFFER_SIZE` | Sets the staging buffer size in KB that is used in data transfer between host and device. Possible values: Non-negative integers where 0 disables use of staging buffer.  
  **Default:** 4 |

**OpenMP Offload Environment Variables for OpenCL Offload Plugin**

memory block allocated from the Level Zero runtime, and `<PoolSize>` means maximum size of the total pool size in MB.

**Examples:**
- all,2,8,1024
  Enables memory pool for all memory types which can allocate up to eight 2MB blocks from a single block allocated from Level Zero with 1GB total pool size allowed.
- device,1,4,512
  Enables memory pool only for device memory type which can allocate up to four 1MB blocks from a single block allocated from Level Zero with 512MB total pool size allowed.

**Default:** all,1,4,256
<table>
<thead>
<tr>
<th>Run-Time Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBOMPTARGET_OPENCL_COMPILATION_OPTIONS</td>
<td>Passes extra compilation options when compiling target programs from SPIRV target images. Possible values: Valid OpenCL compilation options.</td>
</tr>
<tr>
<td>LIBOMPTARGET_OPENCL_LINKING_OPTIONS</td>
<td>Passes extra linking options when linking target programs. Possible values: Valid OpenCL linking options.</td>
</tr>
<tr>
<td>LIBOMPTARGET_OPENCL_USE_SVM</td>
<td>Enables or disables using OpenCL SVM memory for default target memory type. Possible values:</td>
</tr>
<tr>
<td></td>
<td>• 1</td>
</tr>
<tr>
<td></td>
<td>• 0</td>
</tr>
<tr>
<td>LIBOMPTARGET_OPENCL_DATA_TRANSFER_METHOD</td>
<td>Selects memory transfer method to use. This is only effective when OpenCL SVM memory is enabled by setting LIBOMPTARGET_OPENCL_USE_SVM. Possible values:</td>
</tr>
<tr>
<td></td>
<td>• 0</td>
</tr>
<tr>
<td></td>
<td>• 1</td>
</tr>
<tr>
<td></td>
<td>• 2</td>
</tr>
</tbody>
</table>

**DPC++ Environment Variables**

<table>
<thead>
<tr>
<th>DPCPP_CPU_CU_AFFINITY</th>
<th>Set thread affinity to CPU. The value and meaning is the following:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• close - threads are pinned to CPU cores successively through available cores.</td>
</tr>
<tr>
<td></td>
<td>• spread - threads are spread to available cores.</td>
</tr>
<tr>
<td></td>
<td>• master - threads are put in the same cores as master. If DPCPP_CPU_CU_AFFINITY is set, master thread is pinned as well, otherwise master thread is not pinned</td>
</tr>
<tr>
<td></td>
<td>This environment variable is similar to the OMP_PROC_BIND variable used by OpenMP. <strong>Default</strong>: Not set</td>
</tr>
</tbody>
</table>
Run-Time Environment Variable | Description
--- | ---
**DPCPP_CPU_SCHEDULE** | Specify the algorithm for scheduling work-groups by the scheduler. Currently, DPC++ uses TBB for scheduling. The value selects the petitioner used by the TBB scheduler. The value and meaning is the following:
- **dynamic** - TBB auto_partitioner. It performs sufficient splitting to balance load.
- **affinity** - TBB affinity_partitioner. It improves auto_partitioner's cache affinity by its choice of mapping subranges to worker threads compared to
- **static** - TBB static_partitioner. It distributes range iterations among worker threads as uniformly as possible. TBB partitioner relies grain-size to control chunking. Grain-size is 1 by default, indicating every work-group can be executed independently.
**Default:** dynamic

**DPCPP_CPU_NUM_CUS** | Set the numbers threads used for kernel execution. To avoid over subscription, maximum value of DPCPP_CPU_NUM_CUS should be the number of hardware threads. If DPCPP_CPU_NUM_CUS is 1, all the workgroups are executed sequentially by a single thread and this is useful for debugging.

**Default:** Not set. Determined by TBB.

**DPCPP_CPU_PLACES** | Specify the places that affinities are set. The value is \{ sockets | numa_domains | cores | threads \}.

This environment variable is similar to the OMP_PLACES variable used by OpenMP.

**Default:** cores

The following table summarizes CPU environment variables that are recognized at run time.
<table>
<thead>
<tr>
<th>Runtime configuration</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_CONFIG_CPU_FORCE_PRIVATE_MEM_SIZE</td>
<td>32KB</td>
<td>Forces CL_DEVICE_PRIVATE_MEM_SIZE for the CPU device to be the given value. The value must include the unit; for example: 8MB, 8192KB, 8388608B. <strong>NOTE</strong> You must compile your host application with sufficient stack size.</td>
</tr>
<tr>
<td>CL_CONFIG_CPU_FORCE_LOCAL_MEM_SIZE</td>
<td>32KB</td>
<td>Forces CL_DEVICE_LOCAL_MEM_SIZE for CPU device to be the given value. The value needs to be set with size including units, examples: 8MB, 8192KB, 8388608B. <strong>NOTE</strong> You must compile your host application with sufficient stack size. Our recommendation is to set the stack size equal to twice the local memory size to cover possible application and OpenCL Runtime overheads.</td>
</tr>
<tr>
<td>CL_CONFIG_CPU_EXPENSIVE_MEM_OPT</td>
<td>0</td>
<td>A bitmap indicating enabled expensive memory optimizations. These optimizations may lead to more JIT compilation time, but give some performance benefit. <strong>NOTE</strong> Currently, only the least significant bit is available.</td>
</tr>
<tr>
<td>CL_CONFIG_CPU_STREAMING_ALWAYS</td>
<td>False</td>
<td>Controls whether non-temporal instructions are used.</td>
</tr>
</tbody>
</table>

**See Also**
Qopenmp compiler option
Thread Affinity Interface
Compilation Phases

The Intel® oneAPI DPC++/C++ Compiler processes C/C++ and DPC++ language source files. Compilation can be divided into these major phases:

- Preprocessing
- Semantic parsing
- Optimization
- Code generation
- Linking

The first four phases are performed by the compiler:

**Example**

```c
// Linux*

// C++
icx or icpx
//DPC++
dpcpp

// Windows*

// C++
icx or icpx
//DPC++
dpcpp-cl
```

If you specify the `c` option at compilation time, the compiler will generate only object files. You will need to explicitly invoke linker in order to generate the executable.

This content is specific to C++; it does not apply to DPC++. If you are compiling for a 32-bit target, you may either set the environment variable, `INTEL_TARGET_ARCH_IA32`, or use the `[Q]m32` option. If you used the `c` option you will need to pass the `[Q]m32` option to the linker as well.

If you specify the `E` and `P` options when calling the compiler, the compiler will only generate the preprocessed file with an `.i` extension.

**See Also**

- `c` compiler option
- `E` compiler option
- `P` compiler option

Passing Options to the Linker

**Specifying Linker Options**

This topic describes the options that let you control and customize linking with tools and libraries and define the output of the linker.

**Windows**

This section describes options specified at compile-time that take effect at link-time.
You can use the `link` option to pass options specifically to the linker at compile time. For example:

**For C++:**

```
icc a.cpp libfoo.lib /link -delayload:comct132.dll
```

**For DPC++**

```
dpcpp-cl a.cpp libfoo.lib /link -delayload:comct132.dll
```

In this example, the compiler recognizes that `libfoo.lib` is a library that should be linked with `a.cpp`, so it does not need to follow the `link` option on the command line. The compiler does not recognize `-delayload:comct132.dll`, so the `link` option is used to direct the option to the linking phase. On C++, you can use the `Qoption` option to pass options to various tools, including the linker. You can also use `#pragma comment` on C++ to pass options to the linker. This does not apply to DPC++. For example:

```
#pragma comment(linker, "\defaultlib:mylib.lib")
```

**OR**

```
#pragma comment(lib, "mylib.lib")
```

Both examples instruct the compiler to link `mylib.lib` at link time.

---

**Linux**

This section describes options specified at compile-time that take effect at link-time to define the output of the `ld` linker. See the `ld` man page for more information on the linker.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-L directory</code></td>
<td>Instruct the linker to search <code>directory</code> for libraries.</td>
</tr>
<tr>
<td><code>-Qoption tool, list</code></td>
<td>Passes an argument list to another program in the compilation sequence, such as the assembler or linker.</td>
</tr>
<tr>
<td><code>-shared</code></td>
<td>Instructs the compiler to build a Dynamic Shared Object (DSO) instead of an executable.</td>
</tr>
<tr>
<td><code>-shared-libgcc</code></td>
<td><code>-shared-libgcc</code> has the opposite effect of <code>-static-libgcc</code>. When it is used, the GNU standard libraries are linked in dynamically, allowing the user to override the static linking behavior when the <code>-static</code> option is used.</td>
</tr>
</tbody>
</table>

**NOTE**

Note: By default, all C++ standard and support libraries are linked dynamically.

- `-shared-intel` Specifies that all Intel-provided libraries should be linked dynamically.
- `-static` Causes the executable to link all libraries statically, as opposed to dynamically.

  When `-static` is not used:
  - `/lib/ld-linux.so.2` is linked in
  - all other libs are linked dynamically

  When `-static` is used:
### Specifying Alternate Tools and Paths

This content is specific to C++; it does not apply to DPC++.

Use the **Qlocation** option to specify an alternate path for a tool. This option accepts two arguments using the following syntax:

#### Qlocation Syntax

```
# (Linux*)
-Qlocation,tool,path

# (Windows*)
/Qlocation,tool,path
```

where **tool** designates which compilation tool is associated with the alternate **path**.

<table>
<thead>
<tr>
<th><strong>tool</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>cpp</td>
<td>Specifies the preprocessor for the compiler.</td>
</tr>
<tr>
<td>c</td>
<td>Specifies the Intel® oneAPI DPC++/C++ Compiler .</td>
</tr>
<tr>
<td>asm</td>
<td>Specifies the assembler.</td>
</tr>
<tr>
<td>link</td>
<td>Specifies the linker.</td>
</tr>
</tbody>
</table>

Use the **Qoption** option to pass an option specified by **optlist** to a **tool**, where **optlist** is a comma-separated list of options. The syntax for this command is:

#### Qoption Syntax

```
# (Linux*)
-Qoption,tool,optlist

# (Windows*)
/Qoption,tool,optlist
```
where *tool* designates which compilation tool receives the *optlist*.

<table>
<thead>
<tr>
<th>tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpp</td>
<td>Specifies the preprocessor for the compiler.</td>
</tr>
<tr>
<td>c</td>
<td>Specifies the Intel® oneAPI DPC++/C++ Compiler.</td>
</tr>
<tr>
<td>asm</td>
<td>Specifies the assembler.</td>
</tr>
<tr>
<td>link</td>
<td>Specifies the linker.</td>
</tr>
</tbody>
</table>

*optlist* indicates one or more valid argument strings for the designated program. If the argument is a command-line option, you must include the hyphen. If the argument contains a space or tab character, the entire argument must be enclosed in quotation characters (""). Separate multiple arguments with commas.

## Using Configuration Files

You can decrease the time you spend entering command-line options by using the configuration file to automate command-line entries. Configuration files are automatically processed every time you run the Intel® oneAPI DPC++/C++ Compiler. You can insert any valid command-line options into the configuration file. The compiler processes options in the configuration file in the order in which they appear, followed by the specified command-line options when the compiler is invoked.

**NOTE**
Options in the configuration file are executed every time you run the compiler. If you have varying option requirements for different projects, use Using Response Files.

### Sample Configuration Files

**NOTE**
Anytime you instruct the compiler to use a different configuration file, the default configuration file(s) are ignored.

The following examples illustrate basic configuration files. The pound (#) character indicates that the rest of the line is a comment.

In the Windows® examples, the compiler reads the configuration file and invokes the `-I` option every time you run the compiler, along with any options specified on the command line.

```bash
## Sample icpx.cfg file
-1/my_headers

## Sample icx.cfg file
/1c:\my_headers
```

### See Also
- Supported Environment Variables
- Using Response Files
Using Response Files

You can use response files to:

- Specify options used during particular compilations or projects.
- Save this information in individual files.

Response files are invoked as options on the command line. Options in response files are inserted in the command line at the point where the response file is invoked. Unlike configuration files, which are automatically processed every time you run the compiler, response files must be invoked as an option on the command line. If you create a response file without specifying it on the command line, it will not be invoked.

Sample Response Files

Example

(\textit{Linux*})
\begin{verbatim}
# response file: response1.txt
# compile with these options
-w0
# end of response1 file

# response file: response2.txt
# compile with these options
-O0
# end of response2 file
\end{verbatim}

(\textit{Windows*})
\begin{verbatim}
# response file: response1.txt
# compile with these options
/W0
# end of response1 file

# response file: response2.txt
# compile with these options
/Od
# end of response2 file
\end{verbatim}

Use response files to decrease the time spent entering command-line options and to ensure consistency by automating command-line entries. Use individual response files to maintain options for specific projects.

Any number of options or file names can be placed on a line in a response file. Several response files can be referenced in the same command line. The following example shows how to specify a response file on the command line:

Example

\begin{verbatim}
\textbf{NOTE} dpcpp is only available for the Intel® DPC++ Compiler

(\textit{Linux*})
icpx @response1.txt prog1.cpp @response2.txt prog2.cpp

(\textit{Linux*})
dpcpp @response1.txt prog1.cpp @response2.txt prog2.cpp
\end{verbatim}
Example

<table>
<thead>
<tr>
<th># (Windows*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>icx @response1.txt prog1.cpp @response2.txt prog2.cpp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># (Windows*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpcpp-cl @response1.txt prog1.cpp @response2.txt prog2.cpp</td>
</tr>
</tbody>
</table>

**NOTE**
An "at" sign (@) must precede the name of the response file on the command line.

**See Also**
Using Configuration Files

### Global Symbols and Visibility Attributes (Linux*)

This topic applies to C/C++ applications for Linux* only.

A global symbol is one that is visible outside the compilation unit (single source file and its include files) in which it is declared. In C/C++, this means anything declared at file level without the `static` keyword. For example:

```c
int x = 5;       // global data definition
extern int y;   // global data reference
int five()       // global function definition
{ return 5; }   // global function reference
extern int four(); // global function reference
```

A complete program consists of a main program file and possibly one or more shareable object (.so) files that contain the definitions for data or functions referenced by the main program. Similarly, shareable objects might reference data or functions defined in other shareable objects. Shareable objects are so called because if more than one simultaneously executing process has the shareable object mapped into its virtual memory, there is only one copy of the read-only portion of the object resident in physical memory. The main program file and any shareable objects that it references are collectively called the components of the program.

Each global symbol definition or reference in a compilation unit has a visibility attribute that controls how (or if) it may be referenced from outside the component in which it is defined. There are five possible values for visibility:

- **EXTERNAL** – The compiler must treat the symbol as though it is defined in another component. For a definition, this means that the compiler must assume that the symbol will be overridden (preempted) by a definition of the same name in another component. See Symbol Preemption. If a function symbol has external visibility, the compiler knows that it must be called indirectly and can inline the indirect call stub.
- **DEFAULT** – Other components can reference the symbol. Furthermore, the symbol definition may be overridden (preempted) by a definition of the same name in another component.
- **PROTECTED** – Other components can reference the symbol, but it cannot be preempted by a definition of the same name in another component.
- **HIDDEN** – Other components cannot directly reference the symbol. However, its address might be passed to other components indirectly (for example, as an argument to a call to a function in another component, or by having its address stored in a data item reference by a function in another component).
- **INTERNAL** – The symbol cannot be referenced outside its defining component, either directly or indirectly.
Static local symbols (in C/C++, declared at file scope or elsewhere with the keyword static) usually have hidden visibility— they cannot be referenced directly by other components (or, for that matter, other compilation units within the same component), but they might be referenced indirectly.

**NOTE**
Visibility applies to references as well as definitions. A symbol reference's visibility attribute is an assertion that the corresponding definition will have that visibility.

### Specifying Symbol Visibility Explicitly (Linux*)

This topic applies to C/C++ applications for Linux* only.

You can explicitly set the visibility of an individual symbol using the `visibility` attribute on a data or function declaration. For example:

```c
int i __attribute__((visibility("default")))
void x __attribute__((visibility("hidden"))) () {...}
extern void y() __attribute__((visibility("protected")))
```

The `visibility` declaration attribute accepts one of the five keywords:
- `external`
- `default`
- `protected`
- `hidden`
- `internal`

The value of the `visibility` declaration attribute overrides the default set by the options `-fpic` or `-fno-common`.

### Saving Compiler Information in Your Executable

#### On Linux*

To view the information stored in the object file, use the following command:

```bash
objdump -sj comment a.out
strings -a a.out | grep comment:
```

#### On Windows*

To view the linker directives stored in string format in the object file, use the following command:

```cmd
link /dump /directives filename.obj
```

In the output, the `?-comment` linker directive displays the compiler version information. To search your executable for compiler information, use the following command:

```cmd
findstr "Compiler" filename.exe
```

This searches for any strings that have the substring "Compiler" in them.
Linking Debug Information

Windows*
Use option \texttt{Z7} at compile time or option \texttt{debug} at link time to tell the compiler to generate symbolic debugging information in the object file. Alternately, use option \texttt{Zi} at link time to generate executables with debug information in the .pdb file.

Linux*
Use option \texttt{g} at compile time to tell the compiler to generate symbolic debugging information in the object file.

Use option \texttt{gsplit-dwarf} to create a separate object file containing DWARF debug information. Because the DWARF object file is not used by the linker, this reduces the amount of debug information the linker must process and it results in a smaller executable file. See \texttt{gsplit-dwarf} for detailed information.

Ahead of Time Compilation

Ahead of Time (AOT) Compilation is a helpful feature for your development lifecycle or distribution time. It benefits you when you know beforehand what your target device is going to be at application execution time. The AOT feature provides the following benefits:

- No additional compilation time is done when running your application.
- No just-in-time (JIT) bugs encountered due to compilation for the target device, because this step is skipped with AOT compilation.
- Your final code, executing on the target device, can be tested as-is before you deliver it to end-users.

\textbf{NOTE} The program built with AOT compilation for a specific target device will not run on a non-specific device. You must detect the proper target device at runtime and report an error if the targeted device is not present. The use of exception handling with an asynchronous exception handler is recommended.

Data Parallel C++ (DPC++) supports AOT compilation for the following targets: Intel\textsuperscript{®} CPUs, Intel\textsuperscript{®} Processor Graphics (Gen9 or above), and Intel\textsuperscript{®} FPGA.

Prerequisites
To target a GPU with the AOT feature, you must have the OpenCL\textsuperscript{™} Offline Compiler (OCLOC) tool installed. Refer to the Intel\textsuperscript{®} oneAPI Toolkit Installation Guide’s section: Install OCLOC to install the tool on your operating system.

How to Use AOT for the Target Device (Intel\textsuperscript{®} CPUs)
The supported options are:

- \texttt{-fsycl-targets=spir64_x86_64-unknown-unknown-sycldevice}
- \texttt{-Xs "-march=<arch>'}, where \texttt{<arch>} is one of the following:

\begin{table}[h]
\centering
\begin{tabular}{|c|l|}
\hline
Switch & Display Name \\
\hline
\texttt{avx} & Intel\textsuperscript{®} Advanced Vector Extensions (Intel\textsuperscript{®} AVX) \\
\hline
\end{tabular}
\end{table}
Switch | Display Name
--- | ---
**avx2** | Intel® Advanced Vector Extensions 2 (Intel® AVX2)
**avx512** | Intel® Advanced Vector Extensions 512 (Intel® AVX-512)
**sse4.2** | Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2)

Examples:
- **Linux**: dpcpp -fsycl-targets=spir64_x86_64-unknown-unknown-sycldevice -Xs "-march=avx2" main.cpp
- **Windows**: dpcpp-cl /EHsc -fsycl-targets=spir64_x86_64-unknown-unknown-sycldevice -Xs "-march=avx2" test_cpu.cpp

**Building an Application with Multiple Source Files for CPU Targeting**

**Method 1**: Compile your normal files (with no DPC++ kernels) to create host objects. Then compile the file with the kernel code and link it with the rest of the application.

- **Linux**:
  1. dpcpp -c main.cpp
  2. dpcpp -fsycl-targets=spir64_x86_64-unknown-unknown-sycldevice -Xs "-march=avx2" mandel.cpp main.o

- **Windows**:
  1. dpcpp-cl -c /EHsc main.cpp
  2. dpcpp-cl /EHsc -fsycl-targets=spir64_x86_64-unknown-unknown-sycldevice -Xs "-march=avx2" mandel.cpp -link main.obj

**Method 2**: Compile the file with the kernel code and create a fat object. Then compile the rest of the files and linking to create a fat executable:

- **Linux**:
  1. dpcpp -c -fsycl-targets=spir64_x86_64-unknown-unknown-sycldevice -Xs "-march=avx2" mandel.cpp
  2. dpcpp main.cpp mandel.o

- **Windows**:
  1. dpcpp-cl -c /EHsc -fsycl-targets=spir64_x86_64-unknown-unknown-sycldevice -Xs "-march=avx2" mandel.cpp
  2. dpcpp-cl /EHsc main.cpp mandel.obj

**NOTE** Currently, Method 2 only works on a HOST selector.

**How to Use AOT for Intel® Integrated Graphics (Intel® GPU)**

The supported options are:
- `-fsycl-targets=spir64_gen-unknown-unknown-sycldevice`
- `-Xs "-device <arch>"` option, where `<arch>` is one of the following:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Display Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>skl</strong></td>
<td>6th generation Intel® Core™ Processor (Skylake with Intel® Processor Graphics Gen9)</td>
</tr>
</tbody>
</table>
Switch | Display Name
---|---
kbl | 7th generation Intel® Core™ Processor (Kaby Lake with Intel® Processor Graphics Gen9)
cfl | 8th generation Intel® Core™ Processor (Coffee Lake with Intel® Processor Graphics Gen9)
glk | Gemini Lake with Intel® Processor Graphics Gen9
icllp | 10th generation Intel® Core™ Processor (Ice Lake with Intel® Processor Graphics Gen11)
tgllp | 11th generation Intel® Core™ Processor (Tiger Lake with Intel® Processor Graphics Gen12)
dg1 | Intel® Iris® Xe MAX graphics
Gen9 | Intel® Processor Graphics Gen9
Gen11 | Intel® Processor Graphics Gen11
Gen12LP | Intel® Processor Graphics Gen12 (Lower Power)

To see all the device types, use the following command:

```
ocloc compile --help
```

If multiple target devices are listed, the Intel® oneAPI DPC++/C++ Compiler compiles for each of these targets and creates a fat-binary that contains all the device binaries produced this way.

Examples of supported -device patterns:

- **Linux:**
  - To compile for a single target, using `skl` as an example, use:
    
    ```
dpcpp -fsycl-targets=spir64_gen-unknown-unknown-sycldevice -Xs "-device skl" vector-add.cpp
    
    dpcpp -fsycl-targets=spir64_gen-unknown-unknown-sycldevice -Xs "-device skl,icllp" vector-add.cpp
    
    dpcpp -fsycl-targets=spir64_gen-unknown-unknown-sycldevice -Xs "-device *" vector-add.cpp
    
    To compile for a single target, using `skl` as an example, use:
    ```
    dpcpp-cl /EHsc -fsycl-targets=spir64_gen-unknown-unknown-sycldevice -Xs "-device skl" vector-add.cpp
    
    dpcpp-cl /EHsc -fsycl-targets=spir64_gen-unknown-unknown-sycldevice -Xs "-device skl,icllp" vector-add.cpp
    
    dpcpp-cl /EHsc -fsycl-targets=spir64_gen-unknown-unknown-sycldevice -Xs "-device *" vector-add.cpp
    
    **Using AOT in Microsoft Visual Studio***

You can use Microsoft Visual Studio for compiling and linking. Set the flags below to use AOT compilation for CPU or GPU.
For CPU:

- To compile, in the dialog box, select: **Configuration Properties > DPC++ > General > Specify SYCL offloading targets for AOT compilation**
- To link, in the dialog box, select: **Configuration Properties > Linker > General > Specify CPU Target Device for AOT compilation**

For GPU:

- To compile, in the dialog box, select: **Configuration Properties > DPC++ > General > Specify SYCL offloading targets for AOT compilation**
- To link, in the dialog box, select: **Configuration Properties > Linker > General > Specify GPU Target Device for AOT compilation**

---

**Use a Third-Party Compiler as a Host Compiler for DPC++ Code**

This content describes the steps needed to use an external host compiler (G++*) along with the Intel® oneAPI DPC++/C++ Compiler.

In this example, you will use a host compiler to generate the host objects and perform the final link. The host compiler needs to know where to find the required headers and libraries. Follow the example instructions to build a Data Parallel C++ (DPC++) program using a G++ Compiler (g++) for host code and an Intel® oneAPI DPC++/C++ Compiler (dpcpp) for DPC++ code.

**For Linux**

This example includes the following:

- a.cpp: DPC++ code
- b.cpp: DPC++ code
- main.cpp: C++ code

1. Follow the **Get Started with the Intel® oneAPI Base Toolkit for Linux* guide to set up the build environment:**
   
   NOTE The build environment requires GCC* version 5.1 or above to be installed and accessible.

   ```
   source /opt/intel/oneapi/setvars.sh
   ```

2. Set up the headers and library locations:

   ```
   export LIBDIR=<Location of libsyscl.so>
   export INCLUDEDIR=<Location of SYCL headers>
   ```

3. Build the objects for your device:

   ```
   dpcpp -c a.cpp -fPIC -o a.o
   dpcpp -c b.cpp -fPIC -o b.o
   ```

4. Create the integration header files (used by the host compiler):

   ```
   dpcpp -fsycl-device-only -Xclang -fsycl-int-header=a_host.h a.cpp
   dpcpp -fsycl-device-only -Xclang -fsycl-int-header=b_host.h b.cpp
   ```
5. Create the host objects:

```bash
g++ -std=c++17 -c a.cpp -o a_host.o -include a_host.h -fPIC -I$INCLUDEDIR
```

```bash
... b.cpp -o b_host.o -include b_host.h -fPIC -I$INCLUDEDIR...
```

6. Compile other C++ code (or non-DPC++ code) using G++:

```bash
... main.cpp -c -fPIC -I$INCLUDEDIR...
```

7. Create a device object:

```bash
dpcpp -fPIC -fsycl -fsycl-link a.o b.o -o device.o
```

8. Create an archive `libuser.a` that contains the necessary host and device objects:

**NOTE** This step is optional.

```bash
ar -rcs libuser.a a_host.o b_host.o device.o
```

9. Perform the final link to create a `final.exe` executable:

```bash
... main.o a_host.o b_host.o device.o -L$LIBDIR -lOpenCL -lsycl -o finalexexe.exe
```

10. Build the `final.exe` with an archive:

**NOTE** This step is optional.

```bash
... main.o -Wl,--whole-archive libuser.a -Wl,--no-whole-archive -L$LIBDIR -lOpenCL -lsycl -o finalexexe.exe
```

**For Windows**

Windows is not supported in this release.

**Options**

The compiler has two options that let you use an external compiler to perform host side compilation. The options are:

- `fsycl-host-compiler`: Tells the compiler to use the specified compiler for host compilation of the performed offloading compilation.
- `fsycl-host-compiler-options`: Passes options to the compiler specified by the option `fsycl-host-compiler`.

**See Also**

`fsycl-host-compiler`

**See Also**

`fsycl-host-compiler-options`
This section contains information about features related to code optimization and program performance improvement.

Extensions

For the latest information about extensions, see the oneAPI Specification and the DPC++ Language and API Reference.

OpenMP* Support

The Intel® oneAPI DPC++/C++ Compiler supports most of the OpenMP* Application Programming Interface versions 5.0 and 5.1. For the complete OpenMP specification, read the specifications available from the OpenMP web site (http://www.openmp.org; see OpenMP Specifications on that site). The descriptions of OpenMP language characteristics in this documentation often use terms defined in that specification.

The OpenMP API provides symmetric multiprocessing (SMP) with the following major features:

- Relieves you from implementing the low-level details of iteration space partitioning, data sharing, thread creation, scheduling, or synchronization.
- Provides the benefit of performance available from shared memory multiprocessor and multi-core processor systems on all supported Intel architectures, including those processors with Intel® Hyper-Threading Technology (Intel® HT Technology).

The compiler performs transformations to generate multithreaded code based on your placement of OpenMP pragmas in the source program, making it simple to add threading to existing software. The compiler compiles parallel programs and supports the industry-standard OpenMP pragmas.

The compiler provides Intel®-specific extensions to the OpenMP specification including run-time library routines and environment variables. A summary of the compiler options appear in the OpenMP Options Quick Reference.

Parallel Processing with OpenMP

To compile with the OpenMP API, add the pragmas to your code. The compiler processes the code and internally produces a multithreaded version which is then compiled into an executable with the parallelism implemented by threads that execute parallel regions or constructs.

Using Other Compilers

The OpenMP specification does not define interoperability of multiple implementations, so the OpenMP implementation supported by other compilers and OpenMP support in the Intel® oneAPI DPC++/C++ Compiler might not be interoperable. Even if you compile and build the entire application with one compiler, be aware that different compilers might not provide OpenMP source compatibility that enable you to compile and link the same set of application sources with a different compiler and get the expected parallel execution results.
Adding OpenMP* Support to your Application

To add OpenMP* support to your application, do the following:

1. Add the appropriate OpenMP pragmas to your source code.
2. Compile the application with the /Qopenmp (Windows*) or -qopenmp (Linux*) option.
3. For applications with large local or temporary arrays, you may need to increase the stack space available at runtime. In addition, you may need to increase the stack allocated to individual threads by using the OMP_STACKSIZE environment variable or by setting the corresponding library routines.

You can set other environment variables to control multi-threaded code execution.

OpenMP Pragma Syntax

To add OpenMP support to your application, first declare the OpenMP header and then add appropriate OpenMP pragmas to your source code.

To declare the OpenMP header, add the following in your code:

```
#include <omp.h>
```

OpenMP pragmas use a specific format and syntax. Intel Extension Routines to OpenMP describes the OpenMP extensions to the specification that have been added to the Intel® oneAPI DPC++/C++ Compiler.

The following syntax illustrates using the pragmas in your source.

```
Example

<prefix> <pragma> [clause, ...] <newline>
```

where:

- `<prefix>` - Required for all OpenMP pragmas. The prefix must be #pragma omp.
- `<pragma>` - A valid OpenMP pragma. Must immediately follow the prefix.
- `[<clause>]` - Optional. Clauses can be in any order and repeated as necessary, unless otherwise restricted.
- `<newline>` - A required component of pragma syntax. It precedes the structured block that is enclosed by this pragma.

The pragmas are interpreted as comments if you omit the /Qopenmp (Windows) or -qopenmp (Linux) option.

The following example demonstrates one way of using an OpenMP pragma to parallelize a loop.

```
Example

#include <omp.h>
void simple_omp(int *a){
  int i;
  #pragma omp parallel for
  for (i=0; i<1024; i++)
    a[i] = i*2;
}
```

Compile the Application

The /Qopenmp (Windows) or -qopenmp (Linux) option enables the parallelizer to generate multi-threaded code based on the OpenMP pragmas in the source. The code can be executed in parallel on single processor, multi-processor, or multi-core processor systems.

The /Qopenmp (Windows) or -qopenmp (Linux) option works with both -O0 (Linux) and /Od (Windows*) and with any optimization level of O1, O2 and O3.
Specifying -00 (Linux) or /Od (Windows) with the /Qopenmp (Windows) or -qopenmp (Linux) option helps to debug OpenMP applications.

Compile your application using commands similar to those shown below:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icpx -qopenmp source_file</td>
</tr>
<tr>
<td>Windows</td>
<td>icx /Qopenmp source_file</td>
</tr>
</tbody>
</table>

Assume that you compile the sample above, using commands similar to the following, where the c option instructs the compiler to compile the code without generating an executable:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Extended Syntax Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icpx -qopenmp -c parallel.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icx /Qopenmp /c parallel.c</td>
</tr>
</tbody>
</table>

To build your application with target offload support (introduced since OpenMP 4.0) use compiler options to specify the target for which the regions marked with OpenMP "target" pragmas must be compiled. For example:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Syntax Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icpx -qopenmp -fopenmp-targets=spir64 offload.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icx /Qopenmp /fopenmp-targets=spir64 offload.c</td>
</tr>
</tbody>
</table>

Refer to Get Started with OpenMP* Offload to GPU for the Intel® oneAPI DPC/C++ Compiler and Intel® Fortran Compiler for more information.

Configure the OpenMP Environment

Before you run the multi-threaded code, you can set the number of desired threads using the OpenMP environment variable, OMP_NUM_THREADS.

See Also

- c compiler option
- O compiler option
- OpenMP* Examples
- qopenmp, Qopenmp compiler option
- Supported Environment Variables

Parallel Processing Model

A program containing OpenMP* pragmas begins execution as a single thread, called the initial thread of execution. The initial thread executes sequentially until the first parallel construct is encountered.

The omp parallel pragma defines the extent of the parallel construct. When the initial thread encounters a parallel construct, it creates a team of threads, with the initial thread becoming the primary thread of the team. All program statements enclosed by the parallel construct are executed in parallel by each thread in the team, including all routines called from within the enclosed statements.
The statements enclosed lexically within a construct define the static extent of the construct. The dynamic extent includes all statements encountered during the execution of a construct by a thread, including all called routines.

When a thread encounters the end of a structured block enclosed by a parallel construct, the thread waits until all threads in the team have arrived. When that happens the team is dissolved, and only the primary thread continues execution of the code following the parallel construct. The other threads in the team enter a wait state until they are needed to form another team. You can specify any number of parallel constructs in a single program. As a result, thread teams can be created and dissolved many times during program execution.

The following example illustrates, from a high level, the execution model for the OpenMP constructs. The comments in the code explain the structure of each construct or section.

```
Example

main() {                     // Begin serial execution.
...                        // Only the initial thread executes
#pragma omp parallel       // Begin a parallel construct and form a team.
{
  #pragma omp sections     // Begin a worksharing construct.
  {
    #pragma omp section   // One unit of work.
    {...}
    #pragma omp section   // Another unit of work.
    {...}
  }                        // Wait until both units of work complete.
...                      // This code is executed by each team member.
#pragma omp for nowait   // Begin a worksharing Construct
for(...) {               // Each iteration chunk is unit of work.
...                    // Work is distributed among the team members.
}                        // End of worksharing construct.
#pragma omp critical     // Begin a critical section.
{...}                    // Only one thread executes at a time.
...                      // This code is executed by each team member.
#pragma omp barrier      // Wait for all team members to arrive.
...                      // This code is executed by each team member.
}                          // End of Parallel Construct
...                      // Disband team and continue serial execution.
}                            // End serial execution.
```

**Using Orphaned Pragmas**

In routines called from within parallel constructs, you can also use pragmas. Pragmas that are not in the static extent of the parallel construct, but are in the dynamic extent, are called orphaned pragmas. Orphaned pragmas allow you to execute portions of your program in parallel with only minimal changes to the sequential version of the program. Using this functionality, you can code parallel constructs at the top levels of your program call tree and use directives to control execution in any of the called routines. For example:

```
Example

int main(void) {
  #pragma omp parallel {
    phase1();
  }
}
```
Example

```c
void phase1(void) {
    #pragma omp for // This is an orphaned pragma.
    for(i=0; i < n; i++) { some_work(i); }
}
```

This is an orphaned `omp for` loop pragma since the parallel region is not lexically present in routine `phase1`.

**Data Environment**

You can control the data environment of OpenMP constructs by using data environment clauses supported by the construct. You can also privatize named global-lifetime objects by using the `threadprivate` pragma.

Refer to the OpenMP specification for the full list of data environment clauses. Some commonly used ones include:

- `default`
- `shared`
- `private`
- `firstprivate`
- `lastprivate`
- `reduction`
- `linear`
- `map`

You can use several pragma clauses to control the data scope attributes of variables for the duration of the construct in which you specify them; however, if you do not specify a data scope attribute clause on a pragma, the behavior for the variable is determined by the default scoping rules, which are described in the OpenMP specification, for the variables affected by the directive.

**Determining How Many Threads to Use**

For applications where the workload depends on application input that can vary widely, delay the decision about the number of threads to employ until runtime when the input sizes can be examined. Examples of workload input parameters that affect the thread count include things like matrix size, database size, image/video size and resolution, depth/breadth/bushiness of tree-based structures, and size of list-based structures. Similarly, for applications designed to run on systems where the processor count can vary widely, defer choosing the number of threads to employ until application runtime when the machine size can be examined.

For applications where the amount of work is unpredictable from the input data, consider using a calibration step to understand the workload and system characteristics to aid in choosing an appropriate number of threads. If the calibration step is expensive, the calibration results can be made persistent by storing the results in a permanent place like the file system.

Avoid simultaneously using more threads than the number of processing units on the system. This situation causes the operating system to multiplex threads on the processors and typically yields sub-optimal performance.

When developing a library as opposed to an entire application, provide a mechanism whereby the user of the library can conveniently select the number of threads used by the library, because it is possible that the user has outer-level parallelism that renders the parallelism in the library unnecessary or even disruptive.
Use the `num_threads` clause on parallel regions to control the number of threads employed and use the `if` clause on parallel regions to decide whether to employ multiple threads at all. The `omp_set_num_threads()` routine can also be used, but it also affects parallel regions created by the calling thread. The `num_threads` clause is local in its effect, so it does not impact other parallel regions. The disadvantages of explicitly setting the number of threads are:

1. In a system with a large number of processors, your application will use some but not all of the processors.
2. In a system with a small number of processors, your application may force over subscription that results in poor performance.

The Intel OpenMP runtime will create the same number of threads as the available number of logical processors unless you use the `omp_set_num_threads()` routine. To determine the actual limits, use `omp_get_thread_limit()` and `omp_get_max_active_levels()`. Developers should carefully consider their thread usage and nesting of parallelism to avoid overloading the system. The `OMP_THREAD_LIMIT` environment variable limits the number of OpenMP threads to use for the whole OpenMP program. The `OMP_MAX_ACTIVE_LEVELS` environment variable limits the number of active nested parallel regions.

**Binding Sets and Binding Regions**

The binding task set for an OpenMP construct is the set of tasks that are affected by, or provide the context for, the execution of its region. It can be all tasks, the current team tasks, all tasks of the current team that are generated in the region, the binding implicit task, or the generating task.

The binding thread set for an OpenMP construct is the set of threads that are affected by, or provide the context for, the execution of its region. It can be all threads on a device, all threads in a contention group, all primary threads executing an enclosing teams region, the current team, or the encountering thread.

The binding region for an OpenMP construct is the enclosing region that determines the execution context and the scope of the effects of the directive:

- The binding region for an `omp ordered` construct is the innermost enclosing `omp for` loop region.
- The binding region for an `omp taskwait` construct is the innermost enclosing `omp task` region.
- For all other constructs for which the binding thread set is the current team or the binding task set is the current team tasks, the binding region is the innermost enclosing region.
- For constructs for which the binding task set is the generating task, the binding region is the region of the generating task.
- A `omp parallel` construct need not be active to be a binding region.
- A construct need not be explicit to be a binding region.
- A region never binds to any region outside of the innermost enclosing parallel region.

**Worksharing Using OpenMP**

To get the maximum performance benefit from a processor with multi-core and Intel® Hyper-Threading Technology (Intel® HT Technology), an application needs to be executed in parallel. Parallel execution requires threads, and threading an application is not a simple thing to do; using OpenMP* can make the process a lot easier. Using the OpenMP pragmas, most loops with no loop-carried dependencies can be threaded with one simple statement. This topic explains how to start using OpenMP to parallelize loops, which is also called worksharing.

Options that use OpenMP* are available for both Intel® and non-Intel microprocessors, but these options may perform additional optimizations on Intel® microprocessors than they perform on non-Intel microprocessors. The list of major, user-visible OpenMP* constructs and features that may perform differently on Intel® microprocessors than on non-Intel microprocessors includes: locks (internal and user visible), the SINGLE construct, barriers (explicit and implicit), parallel loop scheduling, reductions, memory allocation, and thread affinity and binding.
Most loops can be threaded by inserting one pragma immediately prior to the loop. Further, by leaving the
details to the Intel® oneAPI DPC++/C++ Compiler and OpenMP, you can spend more time determining which
loops should be threaded and how to best restructure the algorithms for maximum performance. The
maximum performance of OpenMP is realized when it is used to thread hotspots, the most time-consuming
loops in your application.

The power and simplicity of OpenMP is demonstrated by looking at an example. The following loop converts a
32-bit RGB (red, green, blue) pixel to an 8-bit gray-scale pixel. One pragma, which has been inserted
immediately before the loop, is all that is needed for parallel execution.

```c
#pragma omp parallel for
for (i=0; i < numPixels; i++) {
    pGrayScaleBitmap[i] = (unsigned BYTE)
        (pRGBBitmap[i].red * 0.299 +
         pRGBBitmap[i].green * 0.587 +
         pRGBBitmap[i].blue * 0.114);
}
```

First, the example uses worksharing, which is the general term used in OpenMP to describe distribution of
work across threads. When worksharing is used with the `for` construct, as shown in the example, the
iterations of the loop are distributed among multiple threads so that each loop iteration is executed exactly
once with different iterations executing if there is more than one available threads. The `for` construct on its
own only distributes the loop iterations among existing threads. The example uses a `parallel for`
construct, which combines `parallel` and `for` constructs to first create a team of threads and then distribute
the loop iterations among the threads. Since there is no explicit `num_threads` clause, OpenMP determines
the number of threads to create and how to best create, synchronize, and destroy them. OpenMP places the
following five restrictions on which loops can be threaded:

- The loop variable must be of type signed or unsigned integer, random access iterator, or pointer.
- The comparison operation must be in the form `loop_variable <, <=, >, >=, or !=
  loop_invariant_expression` of a compatible type.
- The third expression or increment portion of the `for` loop must be either addition or subtraction by a loop
  invariant value.
- If the comparison operation is `< or `<=, the loop variable must increment on every iteration; conversely, if
  the comparison operation is `>` or `>=, the loop variable must decrement on every iteration.
- The loop body must be single-entry-single-exit, meaning no jumps are permitted from inside to outside
  the loop, with the exception of the `exit` statement that terminates the whole application. If the statements`goto` or `break` are used, the statements must jump within the loop, not outside it. Similarly,
  for exception handling, exceptions must be caught within the loop.

Although these restrictions might sound somewhat limiting, non-conforming loops can frequently be
rewritten to follow these restrictions.

**Basics of Compilation**

Using the OpenMP pragmas requires an OpenMP-compatible compiler and thread-safe libraries. Adding
the `/Qopenmp` (Windows*) or `-qopenmp` (Linux*) option to the compiler instructs the compiler to pay
attention to the OpenMP pragmas and to generate multi-threaded code. If you omit the `/Qopenmp` (Windows)
or `-qopenmp` (Linux) option, the compiler will ignore OpenMP pragmas, which provides a very simple way to
generate a single-threaded version without changing any source code. To compile programs containing target
and related constructs for offloading to a GPU, the `-fopenmp-targets=spir64` and `/Qopenmp-
targets:spir64` flags are needed on Linux and Windows respectively.

For conditional compilation, the compiler defines the `__OPENMP` macro. If needed, the macro can be tested as
shown in the following example.
A Few Simple Examples

The following examples illustrate how simple OpenMP is to use. In common practice, additional issues need to be addressed, but these examples illustrate a good starting point.

In the first example, the following loop clips an array to the range from 0 to 255.

```c
// clip an array to 0 <= x <= 255
for (i=0; i < numElements; i++) {
    if (array[i] < 0)
        array[i] = 0;
    else if (array[i] > 255)
        array[i] = 255;
}
```

You can thread it using a single OpenMP pragma; insert the pragma immediately prior to the loop:

```c
#pragma omp parallel for
for (i=0; i < numElements; i++) {
    if (array[i] < 0)
        array[i] = 0;
    else if (array[i] > 255)
        array[i] = 255;
}
```

In the second example, the loop generates a table of square roots for the numbers from 0 to 100.

```c
double value;
double roots[100];
for (value = 0.0; value < 100.0; value ++) { roots[(int)value] = sqrt(value); }
```

Thread the loop by changing the loop variable to a signed integer or unsigned integer and inserting a #pragma omp parallel for pragma.

```c
int value;
double roots[100];
#pragma omp parallel for
for (value = 0; value < 100; value ++) { roots[value] = sqrt((double)value); }
```

Avoiding Data Dependencies and Race Conditions

When a loop meets all five loop restrictions (listed above) and the compiler threads the loop, the loop still might not work correctly due to the existence of data dependencies.
Data dependencies exist when different iterations of a loop (more specifically a loop iteration that is executed on a different thread) read or write the same location in shared memory. Consider the following example that calculates factorials.

```
Example

// Each loop iteration writes a value that a different iteration reads.
#pragma omp parallel for
for (i=2; i < 10; i++) { factorial[i] = i * factorial[i-1]; }
```

The compiler will thread this loop, but the threading will fail because at least one of the loop iterations is data-dependent upon a different iteration. This situation is referred to as a race condition. Race conditions can only occur when using shared resources (like memory) and parallel execution. To address this problem either rewrite the loop or pick a different algorithm, one that does not contain the race condition.

Race conditions are difficult to detect because, for a given case or system, the threads might win the race in the order that happens to make the program function correctly. Because a program works once does not mean that the program will work under all conditions. Testing your program on various machines, some with Intel® Hyper-Threading Technology and some with multiple physical processors, is a good starting point to help identify race conditions.

Traditional debuggers are useless for detecting race conditions because they cause one thread to stop the race while the other threads continue to significantly change the runtime behavior; however, thread checking tools can help.

### Managing Shared and Private Data

Nearly every loop (in real applications) reads from or writes to memory; it's your responsibility, as the developer, to instruct the compiler what memory should be shared among the threads and what memory should be kept private. When memory is identified as shared, all threads access the same memory location. When memory is identified as private, however, a separate copy of the variable is made for each thread to access in private. When the loop ends, the private copies are destroyed. By default, all variables are shared except for the loop variable, which is private.

Memory can be declared as private in two ways:

- Declare the variable inside the loop—really inside the parallel OpenMP pragma—without the static keyword.
- Specify the private clause on an OpenMP pragma.

The following loop fails to function correctly because the variable `temp` is shared. It should be private.

```
Example

// Variable temp is shared among all threads, so while one thread
// is reading variable temp another thread might be writing to it
#pragma omp parallel for
for (i=0; i < 100; i++) {
    temp = array[i];
    array[i] = do_something(temp);
}
```

The following two examples both declare the variable `temp` as private memory, which solves the problem.

```
Example

#pragma omp parallel for
for (i=0; i < 100; i++) {
    int temp; // variables declared within a parallel construct
               // are, by definition, private
```
The *temp* variable can also be made private in the following way:

```c
#pragma omp parallel for private(temp)
for (i=0; i < 100; i++) {
    temp = array[i];
    array[i] = do_something(temp);
}
```

Every time you use OpenMP to parallelize a loop, you should carefully examine all memory references, including the references made by called functions. Variables declared within a parallel construct are defined as private except when they are declared with the `static` declarator, because static variables are not allocated on the stack.

### Reductions

Loops that accumulate a value are fairly common, and OpenMP has a specific clause to accommodate them. Consider the following loop that calculates the sum of an array of integers.

```c
sum = 0;
for (i=0; i < 100; i++) {
    sum += array[i]; // this variable needs to be shared to generate
    // the correct results, but private to avoid
    // race conditions from parallel execution
}
```

The variable sum in the previous loop must be shared to generate the correct result, but it also must be private to permit access by multiple threads. OpenMP provides the `reduction` clause that is used to efficiently combine the mathematical reduction of one or more variables in a loop. The following example demonstrates how the loop can use the `reduction` clause to generate the correct results.

```c
sum = 0;
#pragma omp parallel for reduction(+:sum)
for (i=0; i < 100; i++) { sum += array[i]; }
```

In the case of the example listed above, the reduction provides private copies of the variable `sum` for each thread, and when the threads exit, it adds the values together and places the result in the one global copy of the variable.

The following table lists the possible reduction operations, along with their initial values (mathematical identity values).

<table>
<thead>
<tr>
<th>Operation</th>
<th><em>private</em> Variable Initialization Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ (addition)</td>
<td>0</td>
</tr>
<tr>
<td>- (subtraction)</td>
<td>0</td>
</tr>
</tbody>
</table>
Multiple reductions in a loop are possible by specifying comma-separated variables and operations on a given parallel construct. Reduction variables must meet the following requirements:

- can be listed in just one reduction.
- cannot be declared constant.
- cannot be declared private in the parallel construct.

**Load Balancing and Loop Scheduling**

Load balancing, the equal division of work among threads, is among the most important attributes for parallel application performance. Load balancing is extremely important, because it ensures that the processors are busy most, if not all, of the time. Without a balanced load, some threads may finish significantly before others, leaving processor resources idle and wasting performance opportunities.

Within loop constructs, poor load balancing is often caused by variations in compute time among loop iterations. It is usually easy to determine the variability of loop iteration compute time by examining the source code. In most cases, you will see that loop iterations consume a uniform amount of time. When that is not true, it may be possible to find a set of iterations that consume similar amounts of time. For example, sometimes the set of all even iterations consumes about as much time as the set of all odd iterations. Similarly, it might be the case that the set of the first half of the loop consumes about as much time as the second half. In contrast, it might be impossible to find sets of loop iterations that have a uniform execution time. Regardless of the case, you should provide this extra loop scheduling information to OpenMP so it can better distribute the iterations of the loop across the threads (and therefore processors) for optimum load balancing.

If you know that all loop iterations consume roughly the same amount of time, the OpenMP schedule clause should be used to distribute the iterations of the loop among the threads in roughly equal amounts via the scheduling policy. In addition, you need to minimize the chances of memory conflicts that may arise because of false sharing due to using large chunks. This behavior is possible because loops generally touch memory sequentially, so splitting up the loop in large chunks— like the first half and second half when using two threads— will result in the least chance for overlapping memory. While this may be the best choice for memory issues, it may be bad for load balancing. Unfortunately, the reverse is also true; what might be best for load balancing may be bad for memory performance. You must strike a balance between optimal memory usage and optimal load balancing by measuring the performance to see what method produces the best results.

Use the following general form on the parallel construct to schedule an OpenMP loop:

```
#pragma omp parallel for schedule(kind [, chunk size])
```

Four different loop scheduling types (kinds) can be provided to OpenMP, as shown in the following table. The optional parameter (chunk), when specified, must be a positive integer.
Divide the loop into equal-sized chunks or as equal as possible in the case where the number of loop iterations is not evenly divisible by the number of threads multiplied by the chunk size. By default, chunk size is \( \frac{\text{loop\_count}}{\text{number\_of\_threads}} \).

Set chunk to 1 to interleave the iterations.

Use the internal work queue to give a chunk-sized block of loop iterations to each thread. When a thread is finished, it retrieves the next block of loop iterations from the top of the work queue.

By default, the chunk size is 1. Be careful when using this scheduling type because of the extra overhead involved.

Similar to dynamic scheduling, but the chunk size starts off large and decreases to better handle load imbalance between iterations. The optional chunk parameter specifies the minimum size chunk to use.

By default, the chunk size is approximately \( \frac{\text{loop\_count}}{\text{number\_of\_threads}} \).

When schedule (auto) is specified, the decision regarding scheduling is delegated to the compiler. The programmer gives the compiler the freedom to choose any possible mapping of iterations to threads in the team.

Uses the \texttt{OMP\_SCHEDULE} environment variable to specify which of the three loop-scheduling types should be used. \texttt{OMP\_SCHEDULE} is a string formatted exactly the same as would appear on the parallel construct.

Assume that you want to parallelize the following loop.

```c
for (i=0; i < NumElements; i++) {
    array[i] = StartVal;
    StartVal++;
}
```

As written, the loop contains a data dependency, making it impossible to parallelize without a change. The new loop, shown below, fills the array in the same manner, but without data dependencies. The new loop benefits from using the SIMD instructions generated by the compiler.

```c
#pragma omp parallel for
for (i=0; i < NumElements; i++)
{
    array[i] = StartVal + i;
}
```

Observe that the code is not 100\% identical because the value of variable \texttt{StartVal} is not incremented. As a result, when the parallel loop is finished, the variable will have a value different from the one produced by the serial version. If the value of \texttt{StartVal} is needed after the loop, the additional statement, shown below, is needed.
OpenMP Tasking Model

The OpenMP tasking model enables parallelization of a large range of applications. A task is an instance of executable code and its data environment that can be scheduled for execution by threads.

The task Construct

The task construct defines an explicit task region as shown in the following example:

Example

```c
void test1(LIST *head) {
    #pragma omp parallel shared(head)
    {
        #pragma omp single
        {
            LIST *p = head;
            while (p != NULL) {
                #pragma omp task firstprivate(p)
                {
                    do_work1(p);
                }
                p = p->next;
            }
        }
    }
}
```

The binding thread set of the task region is the current parallel team. A task region binds to the innermost enclosing parallel region. When a thread encounters a task construct, a task is generated from the structured block enclosed in the construct. The encountering thread may immediately execute the task, or defer its execution. A task construct may be nested inside an outer task, but the task region of the inner task is not a part of the task region of the outer task.

Using Clauses with the task Construct

The task construct can take optional clauses. The data environment of the task is created according to the data-sharing attribute clauses on the task construct and any defaults that apply. The example below shows a way to generate N tasks with one thread and execute the generated tasks with the threads in the parallel team:

Example

```c
double data[N];
int i;
#pragma omp parallel shared(data)
{
    #pragma omp single private(i)
```
**Example**

```c
for (i=0, i<N; i++)
{
    #pragma omp task firstprivate(i) shared(data)
    {
        do_work(data, i);
    }
}
```

**Task scheduling**

When a thread reaches a task scheduling point, it may perform a task switch, suspending the current task and beginning or resuming execution of a different task bound to the current team. Refer to the OpenMP 5.1 specifications for the full list of task scheduling point locations. Some examples include:

- the point where a task is explicitly generated.
- the point immediately following the generation of an explicit task.
- after the last instruction of a task region.
- in a taskwait region.
- in a taskyield region.
- in implicit and explicit barrier regions.

**NOTE**

Task scheduling points dynamically divide task regions into parts. Each part is executed from start to finish without interruption. Different parts of the same task region are executed in the order in which they are encountered. In the absence of task synchronization constructs, the order in which a thread executes parts of different schedulable tasks is unspecified. A correct program must behave correctly and consistently with all conceivable scheduling sequences.

**The taskwait Construct**

The taskwait construct specifies a wait on the completion of child tasks generated since the beginning of the current task. A taskwait region binds to the current task region. The binding thread set of the taskwait region is the current team.

The taskwait region includes an implicit task scheduling point in the current task region. The current task region is suspended at the task scheduling point until execution of all its child tasks generated before the taskwait region is completed.

**Example**

```c
#pragma omp task // TASK1
{
    ...
    #pragma omp task // TASK 2 (child of TASK1)
    {
        do_work1();
    }
    #pragma omp task // TASK3 (child of TASK 1)
    {
```
The taskyield Construct

The taskyield construct specifies that the current task can be suspended at that point and the thread may switch to the execution of a different task. You can use this construct to provide an explicit task scheduling point at a particular point in the task.

See Also
OMP_SCHEDULE
openmp, Oopenmp
Supported Environment Variables

Controlling Thread Allocation

The KMP_HW_SUBSET and KMP_AFFINITY environment variables allow you to control how the OpenMP* runtime uses the hardware threads on the processors. These environment variables allow you to try different thread distributions on the cores of the processors and determine how these threads are bound to the cores. You can use the environment variables to work out what is optimal for your application.

The KMP_HW_SUBSET variable controls the allocation of hardware resources and the KMP_AFFINITY variable controls how the OpenMP threads are bound to those resources.

Controlling Thread Distribution

The KMP_HW_SUBSET variable controls the hardware resource that will be used by the program. This variable specifies the number of sockets to use, how many cores to use per socket and how many threads to assign per core. Consider a system with 24 cores and four hardware threads per core. While specifying two threads per core often yields better performance than one thread per core, specifying three or four threads per core may or may not improve the performance. This variable enables you to conveniently measure the performance of up to four threads per core.

For example, you can determine the effects of assigning 24, 48, 72, or the maximum 96 OpenMP threads in a system with 24 cores by specifying the following variable settings:

<table>
<thead>
<tr>
<th>To Assign This Number of Threads ...</th>
<th>... Use This Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>KMP_HW_SUBSET=24c,1t</td>
</tr>
<tr>
<td>48</td>
<td>KMP_HW_SUBSET=24c,2t</td>
</tr>
<tr>
<td>72</td>
<td>KMP_HW_SUBSET=24c,3t</td>
</tr>
<tr>
<td>96</td>
<td>KMP_HW_SUBSET=24c,4t</td>
</tr>
</tbody>
</table>
Caution
Take care when using the OMP_NUM_THREADS variable along with this variable. Using the OMP_NUM_THREADS variable can result in over or under subscription.

NOTE
If you use KMP_HW_SUBSET to specify more resources than the system has, the runtime will issue a warning and ignore the setting. For example, setting KMP_HW_SUBSET=24c,5t will be ignored on a system where each core has four hardware threads.

Controlling Thread Bindings
The KMP_AFFINITY variable controls how the OpenMP threads are bound to the hardware resources allocated by the KMP_HW_SUBSET variable. While this variable can be set to several binding or affinity types, the following are the recommended affinity types to use to run your OpenMP threads on the processor:

- compact: Distribute the threads sequentially among the cores.
- scatter: Distribute the threads among the cores in a round robin manner. Distribution is one thread per core initially, followed by repeat distribution among the cores.

The following table shows how the threads are bound to the cores when you want to use three threads per core on two cores by specifying KMP_HW_SUBSET=2c,3t:

<table>
<thead>
<tr>
<th>Affinity</th>
<th>OpenMP Threads on Core 0</th>
<th>OpenMP Threads on Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMP_AFFINITY=compact</td>
<td>0, 1, 2</td>
<td>3, 4, 5</td>
</tr>
<tr>
<td>KMP_AFFINITY=scatter</td>
<td>0, 2, 4</td>
<td>1, 3, 5</td>
</tr>
</tbody>
</table>

Determining the Best Setting
To determine the best thread distribution and bindings using these variables, use the following:

1. Ensure that your OpenMP code is working properly before using these environment variables.
2. Establish a baseline with your current OpenMP code to compare to the performance when you allocate the threads to a processor.
3. Measure the performance of distributing one, two, three, or four threads per core by use the KMP_HW_SUBSET variable.
4. Measure the performance of binding the threads to the cores by using the KMP_AFFINITY variable.

See Also
Thread Affinity Interface
Supported Environment Variables

OpenMP* Pragmas Summary
This is a summary of the OpenMP* pragmas supported in the Intel® oneAPI DPC++/C++ Compiler. For detailed information about the OpenMP API, see the OpenMP Application Program Interface Version 5.1 specification, which is available from the OpenMP web site.

PARALLEL Pragma
Use this pragma to form a team of threads and execute those threads in parallel.
<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp parallel</td>
<td>Specifies that a structured block should be run in parallel by a team of threads.</td>
</tr>
</tbody>
</table>

**TASKING Pragma**
Use these pragmas for deferring execution.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp task</td>
<td>Specifies a code block whose execution may be deferred.</td>
</tr>
<tr>
<td>omp taskloop</td>
<td>Specifies that the iterations of one or more associated for loops should be executed using OpenMP tasks. The iterations are distributed across tasks that are created by the construct and scheduled to be executed in parallel by the current team.</td>
</tr>
</tbody>
</table>

**WORKSHARING Pragmas**
Use these pragmas to share work among a team of threads.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp for</td>
<td>Specifies a work-sharing loop. Iterations of the loop are executed in parallel by the threads in the team.</td>
</tr>
<tr>
<td>omp loop</td>
<td>Specifies that the iterations of the associated loops can execute in any order or concurrently.</td>
</tr>
<tr>
<td>omp sections</td>
<td>Defines a set of structured blocks that will be distributed among the threads in the team.</td>
</tr>
<tr>
<td>omp single</td>
<td>Specifies that a block of code is to be executed by only one thread in the team.</td>
</tr>
</tbody>
</table>

**SYNCHRONIZATION Pragmas**
Use these pragmas to synchronize between threads.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp atomic</td>
<td>Specifies a computation that must be executed atomically.</td>
</tr>
<tr>
<td>omp barrier</td>
<td>Specifies a point in the code where each thread must wait until all threads in the team arrive.</td>
</tr>
<tr>
<td>omp critical</td>
<td>Specifies a code block that is restricted to access by only one thread at a time.</td>
</tr>
<tr>
<td>omp flush</td>
<td>Identifies a point at which a thread's temporary view of memory becomes consistent with the memory.</td>
</tr>
<tr>
<td>omp masked</td>
<td>Specifies a block of code to be executed by the primary thread (thread 0) of the team.</td>
</tr>
<tr>
<td>omp master (deprecated, see omp masked)</td>
<td>Specifies a code block that must be executed only once by the primary thread of the team.</td>
</tr>
</tbody>
</table>
Pragma Description
omp ordered Specifies a block of code that the threads in a team must execute in the natural order of the loop iterations, or as a stand-alone directive, specifies cross-iteration dependences in a doacross loop-nest.
omp taskgroup Causes the program to wait until the completion of all enclosed and descendant tasks.
omp taskwait Specifies a wait on the completion of child tasks generated since the beginning of the current task.
omp taskyield Specifies that the current task can be suspended at this point in favor of execution of a different task.

Data Environment Pragmas
Use these pragmas to affect the data environment.

Pragma Description
omp threadprivate Specifies a list of globally-visible variables that will be allocated private to each thread.

Offload Target Control Pragmas
Use these pragmas to control execution on one or more offload targets.

Pragma Description
omp declare target Specifies functions and variables that are created or mapped to a device.
omp declare variant Identifies a variant of a base procedure and specifies the context in which this variant is used.
omp dispatch Determines if a procedure variant is called for a given procedure.
omp distribute Specifies that the iterations of one or more loops should be distributed among the initial threads of all thread teams in a league.
omp interop Identifies a foreign runtime context and identifies runtime characteristics of that context, enabling interoperability with it.
omp requires Lists the features that an implementation must support so that the program compiles and runs correctly.
omp target enter data Specifies that variables are mapped to a device data environment.
omp target exit data Specifies that variables are unmapped from a device data environment.
omp teams Creates a league of thread teams inside a target region to execute a structured block in the initial thread of each team.

Vectorization Pragmas
Use these pragmas to control execution on vector hardware.
<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp simd</td>
<td>Transforms the loop into a loop that will be executed concurrently using SIMD instructions. The <code>early_exit</code> clause is an Intel-specific extension of the OpenMP specification. <strong>early_exit</strong> Allows vectorization of multiple exit loops. When this clause is specified: • Each operation before last lexical early exit of the loop may be executed as if early exit were not triggered within the SIMD chunk. • After the last lexical early exit of the loop, all operations are executed as if the last iteration of the loop was found. • Each list item specified in the <code>linear</code> clause is computed based on the last iteration number upon exiting the loop. • The last value for <code>linear</code> clauses and conditional <code>lastprivates</code> clauses are preserved with respect to scalar execution. • The last value for <code>reductions</code> clauses are computed as if the last iteration in the last SIMD chunk was executed up on exiting the loop. • The shared memory state may not be preserved with regard to scalar execution. • Exceptions are not allowed.</td>
</tr>
<tr>
<td>omp declare simd</td>
<td>Creates a version of a function that can process multiple arguments using Single Instruction Multiple Data (SIMD) instructions from a single invocation from a SIMD loop.</td>
</tr>
</tbody>
</table>

### Cancellation Constructs

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp cancel</td>
<td>Requests cancellation of the innermost enclosing region of the type specified, and causes the encountering task to proceed to the end of the cancelled construct.</td>
</tr>
<tr>
<td>omp cancellation point</td>
<td>Defines a point at which implicit or explicit tasks check to see if cancellation has been requested for the innermost enclosing region of the type specified. This construct does not implement a synchronization between threads or tasks.</td>
</tr>
</tbody>
</table>

### User-Defined Reduction Pragma

Use this pragma to define reduction identifiers that can be used as reduction operators in a reduction clause.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp declare reduction</td>
<td>Declares User-Defined Reduction (UDR) functions (reduction identifiers) that can be used as reduction operators in a reduction clause.</td>
</tr>
</tbody>
</table>
Memory Space Allocation Pragma

Use this declarative directive to allocate memory space.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp allocate</td>
<td>Specifies memory allocators to use for object allocation and deallocation</td>
</tr>
</tbody>
</table>

Combined and Composite Pragmas

Use these pragmas as shortcuts for multiple pragmas in sequence. A combined construct is a shortcut for specifying one construct immediately nested inside another construct. A combined construct is semantically identical to that of explicitly specifying the first construct containing one instance of the second construct and no other statements.

A composite construct is composed of two constructs but does not have identical semantics to specifying one of the constructs immediately nested inside the other. A composite construct either adds semantics not included in the constructs from which it is composed or the nesting of the one construct inside the other is not conforming.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp distribute parallel for 1</td>
<td>Specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams.</td>
</tr>
<tr>
<td>omp distribute parallel for simd1</td>
<td>Specifies a loop that will be executed in parallel by multiple threads that are members of multiple teams. It will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp distribute simd 1</td>
<td>Specifies a loop that will be distributed across the primary threads of the teams region. It will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp for simd1</td>
<td>Specifies that the iterations of the loop will be distributed across threads in the team. Iterations executed by each thread can also be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp parallel for</td>
<td>Provides an abbreviated way to specify a parallel region containing only a FOR construct.</td>
</tr>
<tr>
<td>omp parallel for simd</td>
<td>Specifies a parallel construct that contains one for simd construct and no other statement.</td>
</tr>
<tr>
<td>omp parallel sections</td>
<td>Specifies a parallel construct that contains only a sections construct.</td>
</tr>
<tr>
<td>omp target parallel</td>
<td>Creates a device data environment and executes the parallel region on that device.</td>
</tr>
<tr>
<td>omp target parallel for</td>
<td>Provides an abbreviated way to specify a target construct that contains an omp target parallel for construct and no other statement between them.</td>
</tr>
<tr>
<td>omp target parallel for simd</td>
<td>Specifies a target construct that contains an omp target parallel for simd construct and no other statement between them.</td>
</tr>
<tr>
<td>omp target parallel loop</td>
<td>Provides an abbreviated way to specify a target region that contains only a parallel loop construct.</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>omp target simd</td>
<td>Specifies a target construct that contains an omp simd construct and no other statement between them.</td>
</tr>
<tr>
<td>omp target teams</td>
<td>Creates a device data environment and executes the construct on the same device. It also creates a league of thread teams with the primary thread in each team executing the structured block.</td>
</tr>
<tr>
<td>omp target teams distribute</td>
<td>Creates a device data environment and then executes the construct on that device. It also specifies that loop iterations will be distributed among the primary threads of all thread teams in a league created by a teams construct.</td>
</tr>
<tr>
<td>omp target teams distribute parallel for</td>
<td>Creates a device data environment and then executes the construct on that device. It also specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams created by a teams construct. The loop will be distributed across the teams, which will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp target teams distribute parallel for simd</td>
<td>Creates a device data environment and then executes the construct on that device. It also specifies a loop that can be executed in parallel by multiple threads that are members of multiple teams created by a teams construct. It will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp target teams distribute simd</td>
<td>Creates a device data environment and then executes the construct on that device. It also specifies that loop iterations will be distributed among the primary threads of all thread teams in a league created by a teams construct. It will be executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp target teams loop</td>
<td>Provides an abbreviated way to specify a target region that contains only a teams loop construct.</td>
</tr>
<tr>
<td>omp taskloop simd</td>
<td>Specifies a loop that can be executed concurrently using SIMD instructions and that those iterations will also be executed in parallel using OpenMP* tasks.</td>
</tr>
<tr>
<td>omp teams distribute</td>
<td>Creates a league of thread teams and specifies that loop iterations will be distributed among the primary threads of all thread teams in the league.</td>
</tr>
<tr>
<td>omp teams distribute parallel for</td>
<td>Creates a league of thread teams and specifies that the associated loop can be executed in parallel by multiple threads that are members of multiple teams.</td>
</tr>
<tr>
<td>omp teams distribute parallel for simd</td>
<td>Creates a league of thread teams and specifies that the associated loop can be executed concurrently using SIMD instructions in parallel by multiple threads that are members of multiple teams.</td>
</tr>
<tr>
<td>omp teams distribute simd</td>
<td>Creates a league of thread teams and specifies that the associated loop will be distributed across the primary threads of the teams and executed concurrently using SIMD instructions.</td>
</tr>
<tr>
<td>omp teams loop</td>
<td>Provides an abbreviated way to specify a teams construct that contains only a loop construct.</td>
</tr>
</tbody>
</table>

Footnotes:
This directive specifies a composite construct.

## OpenMP* Library Support

This section provides information about OpenMP* run-time library routines, Intel® compiler extension routines to OpenMP, OpenMP support libraries and how to use them, and the thread affinity interface.

### OpenMP* Run-time Library Routines

OpenMP* provides run-time library routines to help you manage your program in parallel mode. Many of these run-time library routines have corresponding environment variables that can be set as defaults. The run-time library routines let you dynamically change these factors to assist in controlling your program. In all cases, a call to a run-time library routine overrides any corresponding environment variable.

**Caution**

Running OpenMP runtime library routines may initialize the OpenMP runtime environment, which might cause a situation where subsequent programmatic setting of OpenMP environment variables has no effect. To avoid this situation, you can use the Intel extension routine `kmp_set_defaults()` to set OpenMP environment variables.

The compiler supports all the OpenMP run-time library routines. Refer to the OpenMP API specification for detailed information about using these routines.

Include the appropriate declarations of the routines in your source code by adding a statement similar to the following:

```c
#include <omp.h>
```

The header files are provided in the `.\include` (Windows*) or `.\include` (Linux*) directory of your compiler installation.

### Thread Team Routines

Routines that affect and monitor thread teams in the current contention group.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void omp_set_num_threads(int nthreads)</code></td>
<td>Sets the number of threads to use for subsequent parallel regions created by the calling thread.</td>
</tr>
<tr>
<td><code>int omp_get_num_threads(void)</code></td>
<td>Returns the number of threads that are being used in the current parallel region. This function does not necessarily return the value inherited by the calling thread from the <code>omp_set_num_threads()</code> function.</td>
</tr>
<tr>
<td><code>int omp_get_max_threads(void)</code></td>
<td>Returns the number of threads available to subsequent parallel regions created by the calling thread.</td>
</tr>
<tr>
<td><code>int omp_get_thread_num(void)</code></td>
<td>Returns the thread number of the calling thread, within the context of the current parallel region.</td>
</tr>
<tr>
<td>Routine</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>int omp_in_parallel(void)</td>
<td>Returns TRUE if called within the dynamic extent of a parallel region executing in parallel; otherwise returns FALSE.</td>
</tr>
<tr>
<td>void omp_set_dynamic(int dynamic_threads)</td>
<td>Enables or disables dynamic adjustment of the number of threads used to execute a parallel region. If dynamic_threads is TRUE, dynamic threads are enabled. If dynamic_threads is FALSE, dynamic threads are disabled. Dynamic threads are disabled by default.</td>
</tr>
<tr>
<td>int omp_get_dynamic(void)</td>
<td>Returns TRUE if dynamic thread adjustment is enabled, otherwise returns FALSE.</td>
</tr>
<tr>
<td>int omp_get_cancellation(void)</td>
<td>Returns TRUE if cancellation is enabled, otherwise returns FALSE. This routine can be affected by the setting for environment variable OMP_CANCELLATION.</td>
</tr>
<tr>
<td>void omp_set_nested(int nested)</td>
<td>Enables or disables nested parallelism. If nested is TRUE, nested parallelism is enabled. If nested is FALSE, nested parallelism is disabled. Nested parallelism is disabled by default.</td>
</tr>
<tr>
<td>NOTE This has been deprecated.</td>
<td></td>
</tr>
<tr>
<td>int omp_get_nested(void)</td>
<td>Returns TRUE if nested parallelism is enabled, otherwise returns FALSE.</td>
</tr>
<tr>
<td>NOTE This has been deprecated.</td>
<td></td>
</tr>
<tr>
<td>void omp_set_schedule(omp_sched_t kind,int chunk_size)</td>
<td>Determines the schedule of a worksharing loop that is applied when 'runtime' is used as the schedule kind.</td>
</tr>
<tr>
<td>void omp_get_schedule(omp_sched_kind *kind,int *chunk_size)</td>
<td>Returns the schedule of a worksharing loop that is applied when the 'runtime' schedule is used.</td>
</tr>
<tr>
<td>int omp_get_thread_limit(void)</td>
<td>Returns the maximum number of simultaneously executing threads in an OpenMP program.</td>
</tr>
<tr>
<td>int omp_get_supported_active_levels(void)</td>
<td>Returns the number of active levels of parallelism supported by the implementation.</td>
</tr>
</tbody>
</table>
**Routine** | **Description**  
---|---  
`void omp_set_max_active_levels(int max_active_levels)` | Limits the number of nested active parallel regions. The value of `max_active_levels` must evaluate to a non-negative integer.  
`int omp_get_max_active_levels(void)` | Returns the maximum number of nested active parallel regions.  
`int omp_get_level(void)` | Returns the number of nested parallel regions (whether active or inactive) enclosing the task that contains the call, not including the implicit parallel region.  
`int omp_get_ancestor_thread_num(int level)` | Returns the thread number of the ancestor at a given nest level of the current thread.  
`int omp_get_team_size(int level)` | Returns the size of the thread team to which the ancestor or the current thread belongs for a given nested level.  
`int omp_get_active_level(void)` | Returns the number of nested, active parallel regions enclosing the task that contains the call.  

**Thread Affinity Routines**  
Routines that affect and access thread affinity policies that are in effect.  

| Function | Description  
---|---  
`omp_proc_bind_t omp_get_proc_bind(void)` | Returns the currently active thread affinity policy, which can be initialized by the environment variable `OMP_PROC_BIND`. This policy is used for subsequent nested parallel regions.  
`int omp_get_num_places(void)` | Returns the number of places available to the execution environment in the place list of the initial task, usually threads, cores, or sockets.  
`int omp_get_place_num_procs(int place_num)` | Returns the number of processors associated with the place numbered `place_num`. The routine returns zero when `place_num` is negative or is greater than or equal to `omp_get_num_places()`.  
`void omp_get_place_proc_ids(int place_num, int *ids)` | Returns the numerical identifiers of each processor associated with the place numbered `place_num`. The numerical identifiers are non-negative and their meaning is implementation defined. The numerical identifiers are returned in the array `ids` and their order in the array is implementation defined. The array `ids` must be sufficiently large to contain `omp_get_place_num_procs(place_num)`
Function | Description
---|---

| int omp_get_place_num(void) | Returns the place number of the place to which the encountering thread is bound. The returned value is between 0 and omp_get_num_places() - 1, inclusive. When the encountering thread is not bound to a place, the routine returns -1.

| int omp_get_partition_num_places(void) | Returns the number of places in the place partition of the innermost implicit task.

| void omp_get_partition_place_nums(int *place_nums) | Returns the list of place numbers corresponding to the places in the place-partition-var ICV of the innermost implicit task. The array place_nums must be sufficiently large to contain omp_get_partition_num_places() elements.

| void omp_set_affinity_format(const char *format) | Sets the affinity format to be used on the device by setting the value of the affinity-format-var ICV.

| size_t omp_get_affinity_format(char *buffer, size_t size) | Returns the value of the affinity-format-var ICV on the device.

| void omp_display_affinity(const char *format) | Prints the OpenMP thread affinity information using the format specification provided.

| size_t omp_capture_affinity(char *buffer, size_t size, const char *format) | Prints the OpenMP thread affinity information into a buffer using the format specification provided.

### Teams Region Routines

Routines that affect and monitor the league of teams that may execute a teams region.

| Function | Description
---|---

| int omp_get_num_teams(void) | Returns the number of initial teams in the current teams region.

| int omp_get_team_num(void) | Returns the initial team number of the calling thread.

| void omp_set_num_teams(int num_teams) | Affects the number of threads to be used for subsequent teams regions that do not specify a num_teams clause.

| int omp_get_max_teams(void) | Returns an upper bound on the number of teams that could be created by a teams construct without a num_teams clause that is encountered after execution returns from this routine.

| void omp_set_teams_thread_limit(int thread_limit) | Defines the maximum number of OpenMP threads that can participate in each contention group created by a teams construct.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int omp_get_teams_thread_limit(void)</td>
<td>Returns the maximum number of OpenMP threads available to participate in each contention group created by a teams construct.</td>
</tr>
</tbody>
</table>

**Tasking Routines**

Routines that pertain to OpenMP explicit tasks.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int omp_get_max_task_priority(void)</td>
<td>Returns the maximum value that can be specified in the priority clause.</td>
</tr>
<tr>
<td>int omp_in_explicit_task(void)</td>
<td>Returns TRUE if called within an explicit task region; otherwise returns FALSE.</td>
</tr>
<tr>
<td>int omp_in_final(void)</td>
<td>Returns TRUE if called within a final task region; otherwise returns FALSE.</td>
</tr>
</tbody>
</table>

**Resource Relinquishing Routines**

Routines that relinquish resources used by the OpenMP runtime. These routines are only effective on the host device.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int omp_pause_resource(omp_pause_resource_t kind, int device_num)</td>
<td>Allows the runtime to relinquish resources used by OpenMP on the specified device. The routine returns zero in case of success, and non-zero otherwise.</td>
</tr>
<tr>
<td>int omp_pause_resource_all(omp_pause_resource_t kind)</td>
<td>Allows the runtime to relinquish resources used by OpenMP on all devices. The routine returns zero in case of success, and non-zero otherwise.</td>
</tr>
</tbody>
</table>

**Device Information Routines**

Routines that pertain to the set of devices that are accessible to an OpenMP program.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int omp_get_num_procs(void)</td>
<td>Returns the number of processors available to the program.</td>
</tr>
<tr>
<td>void omp_set_default_device(int device_number)</td>
<td>Sets the default device number.</td>
</tr>
<tr>
<td>int omp_get_default_device(void)</td>
<td>Returns the default device number.</td>
</tr>
<tr>
<td>int omp_get_num_devices(void)</td>
<td>Returns the number of target devices.</td>
</tr>
<tr>
<td>int omp_get_device_num(void)</td>
<td>Returns the device number of the device on which the calling thread is executing.</td>
</tr>
<tr>
<td>int omp_is_initial_device(void)</td>
<td>Returns TRUE if the current task is running on the host device; otherwise, FALSE.</td>
</tr>
</tbody>
</table>
Function | Description
--- | ---
int omp_get_initial_device(void) | Returns the device number of the host device. The value of the device number is implementation defined. If it is between 0 and omp_get_num_devices()-1, then it is valid in all device constructs and routines; if it is outside that range, then it is only valid in the device memory routines and not in the device clause.

Device Memory Routines

Routines that support allocation of memory and management of pointers in the data environments of target devices.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *omp_target_alloc(size_t size, int device_num)</td>
<td>Allocates memory in a device data environment and returns a device pointer to that memory.</td>
</tr>
<tr>
<td>void omp_target_free(void *device_ptr, int device_num)</td>
<td>Frees device memory that was allocated by the omp_target_alloc.</td>
</tr>
<tr>
<td>int omp_target_is_present(const void *ptr, int device_num)</td>
<td>Returns TRUE if device_num refers to the host device or if ptr refers to storage that has corresponding storage in the device data environment of device_num. Otherwise, it returns FALSE.</td>
</tr>
<tr>
<td>int omp_target_is_accessible(const void *ptr, size_t size, int device_num)</td>
<td>Returns TRUE if the storage of size bytes starting at the address given by ptr is accessible from device device_num. Otherwise, it returns FALSE.</td>
</tr>
</tbody>
</table>
| int omp_target_memcpy(void *dst, const void *src, size_t length, size_t dst_offset, size_t src_offset, int dst_device_num, int src_device_num) | This routine copies length bytes of memory at offset src_offset from src in the device data environment of device src_device_num to dst, starting at offset dst_offset in the device data environment of the device specified by dst_device_num. Returns zero on success and a non-zero value on failure. Use omp_get_initial_device to return the device number you can use to reference the host device and host device data environment. This routine includes a task scheduling point.

The effect of this routine is unspecified when it is called from within a target region.

| int omp_target_memcpy_rect(void *dst, const void *src, size_t element_size, int num_dims, const size_t *volume, const size_t *dst_offsets, const size_t *src_offsets, const size_t *dst_dimensions, const size_t *src_dimensions, int dst_device_num, int src_device_num) | This routine copies a rectangular subvolume of src, in the device data environment of the device specified by src_device_num, to dst, in the device data environment of the device specified by dst_device_num. Specify the volume in terms of the size of an element, the number of its dimensions, and constant arrays of length num_dims. The maximum number of dimensions |
Routine | Description
--- | ---
supported is three or more. The volume array specifies the length, in number of elements, to copy in each dimension from src to dst. The dst_offsets and src_offsets parameters specify the number of elements from the origin of dst and src, in elements. The dst_dimensions and src_dimensions parameters specify the length of each dimension of dst and src. The routine returns zero if successful. Otherwise, it returns a non-zero value. If both dst and src are NULL pointers, the routine returns the number of dimensions supported by the implementation for the specified device numbers. You can use the device number returned by omp_get_initial_device to reference the host device and host device data environment. This routine contains a task scheduling point.
The effect of this routine is unspecified when called from within a target region.

int omp_target_associate_ptr(const void *host_ptr, const void *device_ptr, size_t size, size_t device_offset, int device_num)
Maps a device pointer, which might be returned by omp_target_alloc, to a host pointer.

int omp_target_disassociate_ptr(const void *ptr, int device_num)
Removes the associated pointer for a given device from a host pointer.

void *omp_get_mapped_ptr(const void *ptr, int device_num)
Returns the device pointer that is associated with a host pointer for a given device.

Lock Routines
Use these routines to affect OpenMP locks.

Function | Description
--- | ---
void omp_init_lock(omp_lock_t *lock)
Initializes the lock to the unlocked state.

void omp_init_nest_lock(omp_nest_lock_t *lock)
Initializes the nested lock to the unlocked state. The nesting count for the nested lock is set to zero.

void omp_init_lock_with_hint(omp_lock_t *lock, omp_sync_hint_t hint)
Initializes the lock to the unlocked state, optionally choosing a specific lock implementation based on hint. See the OpenMP specification for the available hints.

void omp_init_nest_lock_with_hint(omp_nest_lock_t *lock, omp_sync_hint_t hint)
Initializes the nested lock to the unlocked state, optionally choosing a specific lock implementation based on hint. The nesting count for the nested lock is set to zero. See the OpenMP specification for the available hints.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void omp_destroy_lock(omp_lock_t *lock)</td>
<td>Changes the state of the lock to uninitialized.</td>
</tr>
<tr>
<td>void omp_destroy_nest_lock(omp_nest_lock_t *lock)</td>
<td>Changes the state of the nested lock to uninitialized.</td>
</tr>
<tr>
<td>void omp_set_lock(omp_lock_t *lock)</td>
<td>Forces the executing thread to wait until the lock is available. The thread is granted ownership of the lock when it becomes available.</td>
</tr>
<tr>
<td>void omp_set_nest_lock(omp_nest_lock_t *lock)</td>
<td>Forces the executing thread to wait until the nested lock is available. If the thread already owns the lock, then the lock nesting count is incremented.</td>
</tr>
<tr>
<td>void omp_unset_lock(omp_lock_t *lock)</td>
<td>Releases the executing thread from ownership of the lock. The behavior is undefined if the executing thread does not own the lock.</td>
</tr>
<tr>
<td>void omp_unset_nest_lock(omp_nest_lock_t *lock)</td>
<td>Decrements the nesting count for the nested lock and releases the executing thread from ownership of the nested lock if the resulting nesting count is zero. Behavior is undefined if the executing thread does not own the nested lock.</td>
</tr>
<tr>
<td>int omp_test_lock(omp_lock_t *lock)</td>
<td>Attempts to set the lock. If successful, returns TRUE, otherwise returns FALSE.</td>
</tr>
<tr>
<td>int omp_test_nest_lock(omp_nest_lock_t *lock)</td>
<td>Attempts to set the nested lock. If successful, returns the nesting count, otherwise returns zero.</td>
</tr>
</tbody>
</table>

**Timing Routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double omp_get_wtime(void)</td>
<td>Returns a double precision value equal to the elapsed wall clock time (in seconds) relative to an arbitrary reference time. The reference time does not change during program execution.</td>
</tr>
<tr>
<td>double omp_get_wtick(void)</td>
<td>Returns a double precision value equal to the number of seconds between successive clock ticks.</td>
</tr>
</tbody>
</table>

**Event Routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void omp_fulfill_event(omp_event_handle_t event)</td>
<td>Fulfills the event associated with the event handle event and destroys the event.</td>
</tr>
</tbody>
</table>
## Interoperability Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int omp_get_num_interop_properties(const omp_interop_t interop)</td>
<td>Returns the number of implementation-defined properties available for interop. The total number of properties available for interop is the returned value minus <code>omp_ipr_first</code>.</td>
</tr>
<tr>
<td>omp_intptr_t omp_get_interop_int(const omp_interop_t interop, omp_interop_property_t property_id, int *ret_code)</td>
<td>Returns the requested integer property, if available, and zero if an error occurs or no value is available.</td>
</tr>
<tr>
<td>void *omp_get_interop_ptr(const omp_interop_t interop, omp_interop_property_t property_id, int *ret_code)</td>
<td>Returns the requested pointer property, if available, and NULL if an error occurs or no value is available.</td>
</tr>
<tr>
<td>const char *omp_get_interop_str(const omp_interop_t interop, omp_interop_property_t property_id, int *ret_code)</td>
<td>Returns the requested string property as a C string, if available, and NULL if an error occurs or no value is available.</td>
</tr>
<tr>
<td>const char *omp_get_interop_name(const omp_interop_t interop, omp_interop_property_t property_id)</td>
<td>Returns the name of the property identified by <code>property_id</code> as a C string.</td>
</tr>
<tr>
<td>const char *omp_get_interop_type_desc(const omp_interop_t interop, omp_interop_property_t property_id)</td>
<td>Returns a C string that describes the type of the property identified by <code>property_id</code> in human-readable form.</td>
</tr>
<tr>
<td>const char *omp_get_interop_rc_desc(const omp_interop_t interop, omp_interop_rc_t ret_code)</td>
<td>Returns a C string that describes the return code <code>ret_code</code> in human-readable form.</td>
</tr>
</tbody>
</table>

## Memory Management Routines

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp_allocator_handle_t omp_init_allocator(omp_memspace_handle_t memspace, int ntraits, const omp_alloctrait_t traits[])</td>
<td>Creates a new allocator that is associated with the <code>memspace</code> memory space and returns a handle to it.</td>
</tr>
<tr>
<td>void omp_destroy_allocator(omp_allocator_handle_t allocator)</td>
<td>Releases all resources used to implement the allocator handle.</td>
</tr>
<tr>
<td>void omp_set_default_allocator(omp_allocator_handle_t allocator)</td>
<td>Sets the default memory allocator to be used by allocation calls, allocate directives and allocate clauses that do not specify an allocator.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>omp_allocator_handle_t</td>
<td>Returns a handle to the memory allocator to be used by allocation calls, allocate directives and allocate clauses that do not specify an allocator.</td>
</tr>
<tr>
<td>omp_get_default_allocator(void)</td>
<td></td>
</tr>
<tr>
<td>void *omp_alloc(size_t size, omp_allocator_handle_t allocator)</td>
<td>Requests a memory allocation of size bytes from the specified memory allocator.</td>
</tr>
<tr>
<td>void *omp_aligned_alloc(size_t alignment, size_t size, omp_allocator_handle_t allocator)</td>
<td>Requests a memory allocation of size bytes from the specified memory allocator. Memory allocated by <code>omp_aligned_alloc</code> will be byte-aligned to at least the maximum of the alignment required by malloc, the alignment trait of the allocator and the alignment argument value.</td>
</tr>
<tr>
<td>void omp_free(void *ptr, omp_allocator_handle_t allocator)</td>
<td>Deallocates the memory to which <code>ptr</code> points. The <code>ptr</code> argument must have been returned by an OpenMP allocation routine.</td>
</tr>
<tr>
<td>void *omp_calloc(size_t nmemb, size_t size, omp_allocator_handle_t allocator)</td>
<td>Requests a memory allocation from the specified memory allocator for an array of <code>nmemb</code> elements each of which has a size of <code>size</code> bytes.</td>
</tr>
<tr>
<td>void *omp_aligned_calloc(size_t alignment, size_t nmemb, size_t size, omp_allocator_handle_t allocator)</td>
<td>Requests a memory allocation from the specified memory allocator for an array of <code>nmemb</code> elements each of which has a size of <code>size</code> bytes. Memory allocated by <code>omp_aligned_calloc</code> will be byte-aligned to at least the maximum of the alignment required by malloc, the alignment trait of the allocator and the alignment argument value.</td>
</tr>
<tr>
<td>void *omp_realloc(void *ptr, size_t size, omp_allocator_handle_t allocator, omp_allocator_handle_t free_allocator)</td>
<td>Deallocates the memory to which <code>ptr</code> points and requests a new memory allocation of <code>size</code> bytes from the specified memory allocator. Upon success it returns a pointer to the allocated memory and the contents of the new object shall be the same as that of the old object prior to deallocation up to the minimum size of old allocated size and <code>size</code> argument.</td>
</tr>
</tbody>
</table>

**Tool Control Routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int omp_control_tool(int command, int modifier, void *arg)</td>
<td>Enables a program to pass commands to an active tool.</td>
</tr>
</tbody>
</table>

**Environment Display Routines**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void omp_display_env(int verbose)</td>
<td>Displays the OpenMP version number and the initial values of ICVs associated with the environment variables.</td>
</tr>
</tbody>
</table>
Intel® Compiler Extension Routines to OpenMP*

The Intel® compiler implements the following group of routines as extensions to the OpenMP* run-time library:

- Get and set the execution environment
- Get and set the stack size for parallel threads
- Memory allocation
- Get and set the thread sleep time for the throughput execution mode
- Target memory allocation

The Intel® extension routines described in this section can be used for low-level tuning to verify that the library code and application are functioning as intended. These routines are generally not recognized by other OpenMP-compliant compilers, which may cause the link stage to fail in the other compiler. To execute these OpenMP routines, use the `/Qopenmp-stubs (Windows*) or -qopenmp-stubs (Linux*) option.

In most cases, environment variables can be used in place of the extension library routines. For example, the stack size of the parallel threads may be set using the `OMP_STACKSIZE` environment variable rather than the `kmp_set_stacksize_s()` library routine.

**NOTE**
A run-time call to an Intel extension routine takes precedence over the corresponding environment variable setting.

### Execution Environment

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void kmp_set_defaults(char const *)</code></td>
<td>Sets OpenMP environment variables defined as a list of variables separated by &quot;</td>
</tr>
<tr>
<td><code>void kmp_set_library_throughput(void)</code></td>
<td>Sets execution mode to throughput, which is the default. Allows the application to determine the runtime environment. Use in multi-user environments.</td>
</tr>
<tr>
<td><code>void kmp_set_library_turnaround(void)</code></td>
<td>Sets execution mode to turnaround. Use in dedicated parallel (single user) environments.</td>
</tr>
<tr>
<td><code>void kmp_set_library_serial(void)</code></td>
<td>Sets execution mode to serial.</td>
</tr>
<tr>
<td><code>void kmp_set_library(int)</code></td>
<td>Sets execution mode indicated by the value passed to the function. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>• 1 - serial mode</td>
</tr>
<tr>
<td></td>
<td>• 2 - turnaround mode</td>
</tr>
<tr>
<td></td>
<td>• 3 - throughput mode</td>
</tr>
<tr>
<td>Call this routine before the first parallel region is executed.</td>
<td></td>
</tr>
<tr>
<td><code>int kmp_get_library(void)</code></td>
<td>Returns a value corresponding to the current execution mode:</td>
</tr>
<tr>
<td></td>
<td>• 1 - serial</td>
</tr>
</tbody>
</table>
### Function Description

- **2 - turnaround**
- **3 - throughput**

## Stack Size

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size_t kmp_get_stacksize_s(void)</td>
<td>Returns the number of bytes that will be allocated for each parallel thread to use as its private stack. This value can be changed with kmp_set_stacksize_s() routine, prior to the first parallel region or via the KMP_STACKSIZE environment variable.</td>
</tr>
<tr>
<td>int kmp_get_stacksize(void)</td>
<td>Provided for backwards compatibility only. Use kmp_get_stacksize_s() routine for compatibility across different families of Intel processors.</td>
</tr>
<tr>
<td>void kmp_set_stacksize_s(size_t size)</td>
<td>Sets to size the number of bytes that will be allocated for each parallel thread to use as its private stack. This value can also be set via the KMP_STACKSIZE environment variable. In order for kmp_set_stacksize_s() to have an effect, it must be called before the beginning of the first (dynamically executed) parallel region in the program.</td>
</tr>
<tr>
<td>void kmp_set_stacksize(int size)</td>
<td>Provided for backward compatibility only. Use kmp_set_stacksize_s() for compatibility across different families of Intel® processors.</td>
</tr>
</tbody>
</table>

## Memory Allocation

The Intel® compiler implements a group of memory allocation routines as an extension to the OpenMP runtime library to enable threads to allocate memory from a heap local to each thread. These routines are: kmp_malloc(), kmp_calloc(), and kmp_realloc().

The memory allocated by these routines must also be freed by the kmp_free() routine. While you can allocate memory in one thread and then free that memory in a different thread, this mode of operation incurs a slight performance penalty.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void* kmp_malloc(size_t size)</td>
<td>Allocate memory block of size bytes from thread-local heap.</td>
</tr>
<tr>
<td>void* kmp_calloc(size_t nelem, size_t elsize)</td>
<td>Allocate array of nelem elements of size elsize from thread-local heap.</td>
</tr>
<tr>
<td>void* kmp_realloc(void* ptr, size_t size)</td>
<td>Reallocate memory block at address ptr and size bytes from thread-local heap.</td>
</tr>
<tr>
<td>void* kmp_free(void* ptr)</td>
<td>Free memory block at address ptr from thread-local heap.</td>
</tr>
</tbody>
</table>
### Thread Sleep Time

In the throughput OpenMP* Support Libraries, threads wait for new parallel work at the ends of parallel regions, and then sleep, after a specified period of time. This time interval can be set by the `KMP_BLOCKTIME` environment variable or by the `kmp_set_blocktime()` function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `int kmp_get_blocktime(void)` | Returns the number of milliseconds that a thread should wait, after completing the execution of a parallel region, before sleeping, as set either by the `KMP_BLOCKTIME` environment variable or by `kmp_set_blocktime()`.
| `void kmp_set_blocktime(int msec)` | Sets the number of milliseconds that a thread should wait, after completing the execution of a parallel region, before sleeping. This routine affects the block time setting for the calling thread and any OpenMP team threads formed by the calling thread. The routine does not affect the block time for any other threads. |

### Target Memory Allocation

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| `void *omp_target_alloc_host(size_t size, int device_num)` | Returns the address of a storage location that is `size` bytes in length allocated in host memory. The same pointer may be used to access the memory on the host and all supported devices. If the allocation request fails, a null pointer is returned.
| `void *omp_target_alloc_device(size_t size, int device_num)` | Returns the address of a storage allocation that is `size` bytes in length. Device allocations are owned by the device specified by `device_num` in device memory if present. Generally, the allocation can be accessed only by the device, but may be copied to other device or host allocated memory. A null pointer return value indicates the allocation was not successful.
| `void *omp_target_alloc_shared(size_t size, int device_num)` | Returns the address of a storage allocation that is `size` bytes in length. The same pointer may be used to access the memory on the host and the specified device. Shared allocations are shared by the host and the specified device, and are intended to migrate between the host and the device. A null pointer is returned if the allocation is unsuccessful. |
void *ompx_target_realloc(void *ptr, size_t size, int device_num)

Deallocates the device memory specified with ptr and allocates a new device memory with the specified size in bytes for the given device device_num. The returned memory can be accessed only by the specified device. The contents of the new memory object are the same as that of the old object prior to deallocation up to the minimum size of old allocated size and size argument.

void *ompx_target_realloc_host(void *ptr, size_t size, int device_num)

Deallocates the device memory specified with ptr and allocates a new device memory with the specified size in bytes for the given device device_num. The returned memory can be accessed by the host and all supported devices. The contents of the new memory object are the same as that of the old object prior to deallocation up to the minimum size of old allocated size and size argument.

void *ompx_target_realloc_device(void *ptr, size_t size, int device_num)

Deallocates the device memory specified with ptr and allocates a new device memory with the specified size in bytes for the given device device_num. The returned memory can be accessed only by the specified device. The contents of the new memory object are the same as that of the old object prior to deallocation up to the minimum size of old allocated size and size argument.

void *ompx_target_realloc_shared(void *ptr, size_t size, int device_num)

Deallocates the device memory specified with ptr and allocates a new device memory with the specified size in bytes for the given device device_num. The returned memory can be accessed by the host and the specified device. The contents of the new memory object are the same as that of the old object prior to deallocation up to the minimum size of old allocated size and size argument.

void *ompx_target_aligned_alloc(size_t alignment, size_t size, int device_num)

Allocates device memory that is aligned to the specified alignment argument align for the specified device device_num. The returned memory can be accessed only by the specified device.

void *ompx_target_aligned_alloc_host(size_t alignment, size_t size, int device_num)

Allocates device memory that is aligned to the specified alignment argument align for the specified device device_num. The returned memory can be accessed by the host and all supported devices.
void
*ompx_target_aligned_alloc_device(size_t alignment, size_t size, int device_num)

Allocates device memory that is aligned to the specified alignment argument `align` for the specified device `device_num`. The returned memory can be accessed only by the specified device.

void
*ompx_target_aligned_alloc_shared(size_t alignment, size_t size, int device_num)

Allocates device memory that is aligned to the specified alignment argument `align` for the specified device `device_num`. The returned memory can be accessed by the host and the specified device.

See Also
openmp-stubs, Qopenmp-stubs  compiler option
OpenMP* Run-time Library Routines
OpenMP* Support Libraries

OpenMP* Support Libraries

The Intel® oneAPI DPC++/C++ Compiler provides support libraries for OpenMP*. There are several kinds of libraries:

- **Performance**: supports parallel OpenMP execution.
- **Stubs**: supports serial execution of OpenMP applications.

Each kind of library is available for both dynamic and static linking on Linux* operating systems. Only dynamic linking is supported on Windows* operating systems.

Performance Libraries

To use these libraries, specify the `/Qopenmp` (Windows*) or `-qopenmp` (Linux*) option.

Options that use OpenMP* are available for both Intel® and non-Intel microprocessors, but these options may perform additional optimizations on Intel® microprocessors than they perform on non-Intel microprocessors. The list of major, user-visible OpenMP* constructs and features that may perform differently on Intel® microprocessors than on non-Intel microprocessors includes: locks (internal and user visible), the SINGLE construct, barriers (explicit and implicit), parallel loop scheduling, reductions, memory allocation, and thread affinity and binding.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Dynamic Link</th>
<th>Static Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>libiomp5.so</td>
<td>libiomp5.a</td>
</tr>
<tr>
<td>Windows</td>
<td>libiomp5md.lib</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>libiomp5md.dll</td>
<td></td>
</tr>
</tbody>
</table>

Many routines in the OpenMP* support libraries are more optimized for Intel® microprocessors than for non-Intel microprocessors.

Stubs Libraries

To use these libraries, specify `/Qopenmp-stubs` (Windows*) or `-qopenmp-stubs` (Linux*) option. These allow you to compile OpenMP applications in serial mode and provide stubs for OpenMP routines and extended Intel-specific routines.
## Execution Modes

The compiler enables you to run an application under different execution modes specified at run time; the libraries support the turnaround, throughput, and serial modes. Use the `KMP_LIBRARY` environment variable to select the modes at run time.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput (default)</td>
<td>The throughput mode allows the program to yield to other running programs and adjust resource usage to produce efficient execution in a dynamic environment. In a multi-user environment where the load on the parallel machine is not constant or where the job stream is not predictable, it may be better to design and tune for throughput. This minimizes the total time to run multiple jobs simultaneously. In this mode, the worker threads yield to other threads while waiting for more parallel work. After completing the execution of a parallel region, threads wait for new parallel work to become available. After a certain period of time has elapsed, they stop waiting and sleep. Until more parallel work becomes available, sleeping allows processor and resources to be used for other work by non-OpenMP threaded code that may execute between parallel regions, or by other applications. The amount of time to wait before sleeping is set either by the <code>KMP_BLOCKTIME</code> environment variable or by the <code>kmp_set_blocktime()</code> function. A small blocktime value may offer better overall performance if your application contains non-OpenMP threaded code that executes between parallel regions. A larger blocktime value may be more appropriate if threads are to be reserved solely for use for OpenMP execution, but may penalize other concurrently-running OpenMP or threaded applications. The turnaround mode is designed to keep active all processors involved in the parallel computation, which minimizes execution time of a single job. In this mode, the worker threads actively wait for more parallel work, without yielding to other threads (although they are still subject to <code>KMP_BLOCKTIME</code> control). In a dedicated (batch or single user) parallel environment where all processors are exclusively allocated to the program for its entire run, it is most important to effectively use all processors all of the time.</td>
</tr>
<tr>
<td>turnaround</td>
<td>Avoid over-allocating system resources. The condition can occur if either too many threads have been specified, or if too few processors are available at run time. If system resources are over-allocated, this mode will cause poor performance. The throughput mode should be used instead if this occurs.</td>
</tr>
<tr>
<td>serial</td>
<td>The serial mode forces parallel applications to run as a single thread.</td>
</tr>
</tbody>
</table>

### Operating System

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Dynamic Link</th>
<th>Static Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>libiompstubs5.so</td>
<td>libiompstubs5.a</td>
</tr>
<tr>
<td>Windows</td>
<td>libiompstubs5md.lib</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>libiompstubs5md.dll</td>
<td></td>
</tr>
</tbody>
</table>
Using the OpenMP* Libraries

This section describes the steps needed to set up and use the OpenMP* Libraries from the command line. On Windows* systems, you can also build applications compiled with the OpenMP libraries in the Microsoft Visual Studio* development environment.

For a list of the options and libraries used by the OpenMP libraries, see OpenMP* Support Libraries.

Set up your environment for access to the Intel® oneAPI DPC++/C++ Compiler to ensure that the appropriate OpenMP library is available during linking. On Windows systems, you can either execute the appropriate batch (.bat) file or use the command-line window supplied in the compiler program folder that already has the environment set up. On Linux* systems, you can source the appropriate script file (setvars file).

During compilation, ensure that the version of omp.h used when compiling is the version provided by that compiler. For example, use the omp.h provided with GCC* when you compile with GCC.

Caution
Be aware that when using the GCC or Microsoft* Compiler, you may inadvertently use inappropriate header/module files. To avoid this, copy the header/module file(s) to a separate directory and put it in the appropriate include path using the -I option.

If a program uses data structures or classes that contain members with data types defined in omp.h file, then source files that use those data structures should all be compiled with the same omp.h file.

The following table lists the commands used by the various command-line compilers for both C and C++ source files:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>C Source Module</th>
<th>C++ Source Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>gcc</td>
<td>g++</td>
</tr>
<tr>
<td></td>
<td>Intel: icx</td>
<td>Intel: icpx</td>
</tr>
<tr>
<td>Windows</td>
<td>Visual C++: cl</td>
<td>Visual C++: cl</td>
</tr>
<tr>
<td></td>
<td>Intel: icx</td>
<td>Intel: icx</td>
</tr>
</tbody>
</table>

For information on the OpenMP libraries and options used by the compiler, see OpenMP* Support Libraries.

Command-Line Examples, Windows

To compile and link (build) the entire application with one command using the Compatibility libraries, specify the following command:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>C source, dynamic link</td>
<td>icx /MD /Qopenmp hello.c</td>
</tr>
<tr>
<td>C++ source, dynamic link</td>
<td>icpx /MD /Qopenmp hello.cpp</td>
</tr>
</tbody>
</table>

When using the Microsoft Visual C++ Compiler, you should link with the Intel® OpenMP compatibility library. You need to avoid linking the Microsoft OpenMP run-time library (vcomp) and explicitly pass the name of the Intel® OpenMP compatibility library as linker options (following /link):
### Type of File | Commands
--- | ---
C source, dynamic link | `cl /MD /openmp hello.c /link /nodefaultlib:vcomp libiomp5md.lib`

### Type of File | Commands
--- | ---
C++ source, dynamic link | `cl /MD /openmp hello.cpp /link /nodefaultlib:vcomp libiomp5md.lib`

You can also use the Intel oneAPI DPC++/C++ Compiler with the Visual C++ Compiler to compile parts of the application and create object files (object-level interoperability). In this example, the Intel oneAPI DPC++/C++ Compiler compiles and links the entire application:

### Type of File | Commands
--- | ---
C source, dynamic link | `cl /MD /openmp /c f1.c f2.c`
`icx /MD /Qopenmp /c f3.c f4.c`
`icx /MD /Qopenmp f1.obj f2.obj f3.obj f4.obj /Feapp /link /nodefaultlib:vcomp`

The first command produces two object files compiled by Visual C++ Compiler, and the second command produces two more object files compiled by the Intel oneAPI DPC++/C++ Compiler. The final command links all four object files into an application.

Alternatively, the third line below uses the Visual C++ linker to link the application and specifies the Compatibility library `libiomp5md.lib` at the end of the third command:

### Type of File | Commands
--- | ---
C source, dynamic link | `cl /MD /openmp /c f1.c f2.c`
`icx /MD /Qopenmp /c f3.c f4.c`
`link f1.obj f2.obj f3.obj f4.obj /out:app.exe /
nodefaultlib:vcomp libiomp5md.lib`

The following example shows the use of interprocedural optimization by the Intel oneAPI DPC++/C++ Compiler on several files, the Visual C++ Compiler compiles several files, and the Visual C++ linker links the object files to create the executable:

### Type of File | Commands
--- | ---
C source, dynamic link | `icx /MD /Qopenmp /O3 /Qipo /Qipo-c f1.c f2.c f3.c`
`cl /MD /openmp /O2 /c f4.c f5.c`
`cl /MD /openmp /O2 ipo_out.obj f4.obj f5.obj /Feapp /link /
nodefaultlib:vcomp libiomp5md.lib`

The first command uses the Intel oneAPI DPC++/C++ Compiler to produce an optimized multi-file object file named `ipo_out.obj` by default (the `/Fe` option is not required). The second command uses the Visual C++ Compiler to produce two more object files. The third command uses the Visual C++ `cl` command to link all three object files using the Intel oneAPI DPC++/C++ Compiler OpenMP library.

**Using Intel OpenMP Libraries from Visual Studio**

When using systems running Windows, you can make certain changes in the Visual C++ Visual Studio* development environment to allow you to use the Intel oneAPI DPC++/C++ Compiler and Visual C++ to create applications that use the Intel OpenMP libraries.
NOTE Microsoft Visual C++ must have the symbol _OPENMP_NOFORCE_MANIFEST defined or it will include the manifest for the vcom90 dlls. While this may not appear to cause a problem on the build system, it will cause a problem when the application is moved to another system that does not have this DLL installed.

Set the project **Property Pages** to indicate the Intel OpenMP run-time library location:

1. Open the project’s property pages in from the main menu: **Project > Properties** (or right click the Project name and select **Properties**) .
2. Select **Configuration Properties > Linker > General > Additional Library Directories**.
3. Enter the path to the Intel®-provided compiler libraries. For example, for an IA-32 architecture system (C++ only), enter:

   `<Intel_compiler_installation_path>\IA32\LIB`

Make the Intel OpenMP dynamic run-time library accessible at run-time; you must specify the corresponding path:

1. Open the project's property pages in from the main menu: **Project > Properties** (or right click the Project name and select **Properties**).
2. Select **Configuration Properties > Debugging > Environment**.
3. Enter the path to the Intel®-provided compiler libraries. For example, for an IA-32 architecture system (C++ only), enter:

   `PATH=%PATH%;<Intel_compiler_installation_path>\IA32\Bin`

Add the Intel OpenMP run-time library name to the linker options and exclude the default Microsoft OpenMP run-time library:

1. Open the project’s property pages in from the main menu: **Project > Properties** (or right click the Project name and select **Properties**).
2. Select **Configuration Properties > Linker > Command Line > Additional Options**.
3. Enter the OpenMP library name and the Visual C++ linker option, `/nodefaultlib`.

### Command-Line Examples, Linux

To compile and link (build) the entire application with one command using the Intel OpenMP libraries, specify the following Intel oneAPI DPC++/C++ Compiler command on Linux platforms:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>C source</td>
<td>icx -qopenmp hello.c</td>
</tr>
<tr>
<td>C++ source</td>
<td>icpx -qopenmp hello.cpp</td>
</tr>
</tbody>
</table>

By default, the Intel oneAPI DPC++/C++ Compiler performs a dynamic link of the OpenMP libraries. To perform a static link (not recommended), add the option `-qopenmp-link=static`. The option `-qopenmp-link` controls whether the linker uses static or dynamic OpenMP libraries on Linux systems (default is `-qopenmp-link=dynamic`).

You can also use the icx/icpx compilers with the gcc/g++ compilers to compile parts of the application and create object files (object-level interoperability).

In this example, gcc compiles the C file `foo.c` (the gcc option `-fopenmp` enables OpenMP support), and the Intel oneAPI DPC++/C++ Compiler links the application using the Intel OpenMP library:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>C source</td>
<td>gcc -fopenmp -c foo.c</td>
</tr>
<tr>
<td>Type of File</td>
<td>Commands</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>C++ source</td>
<td>g++ -fopenmp -c foo.cpp</td>
</tr>
</tbody>
</table>

When using gcc or the g++ compiler to link the application with the Intel oneAPI DPC++/C++ Compiler OpenMP compatibility library, you need to explicitly pass the Intel OpenMP library name using the -l option, the Linux pthread library using the -l option, and path to the Intel libraries where the Intel oneAPI DPC++/C++ Compiler is installed using the -L option:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>C source</td>
<td>gcc -fopenmp -c foo.c bar.c</td>
</tr>
<tr>
<td></td>
<td>gcc foo.o bar.o -liomp5 -lpthread -L&lt;icx_dir&gt;/lib</td>
</tr>
</tbody>
</table>

You can mix object files, but it is easier to use the Intel oneAPI DPC++/C++ Compiler to link the application so you do not need to specify the gcc-l option, -L option, and the -lpthread option:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>C source</td>
<td>gcc -fopenmp -c foo.c</td>
</tr>
<tr>
<td></td>
<td>icx -qopenmp -c bar.c (Linux)</td>
</tr>
<tr>
<td></td>
<td>icx -qopenmp foo.o bar.o (Linux)</td>
</tr>
</tbody>
</table>

You can mix OpenMP object files compiled with GCC, or Intel oneAPI DPC++/C++ Compiler.

**NOTE** You cannot mix object files compiled by the Intel® Fortran Compiler and the gfortran compiler.

The table illustrates examples of using the Intel Fortran Compiler to link all the objects:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed C and Fortran sources</td>
<td>icx -qopenmp -c ibar.c</td>
</tr>
<tr>
<td></td>
<td>gcc -fopenmp -c gbar.c</td>
</tr>
<tr>
<td></td>
<td>ifort -qopenmp -c foo.f</td>
</tr>
<tr>
<td></td>
<td>ifort -qopenmp foo.o ibar.o gbar.o</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed C and GNU Fortran sources</td>
<td>icx -qopenmp -c ibar.c</td>
</tr>
<tr>
<td></td>
<td>gcc -fopenmp -c gbar.c</td>
</tr>
<tr>
<td></td>
<td>gfortran -fopenmp -c foo.f</td>
</tr>
<tr>
<td></td>
<td>gfortran foo.o ibar.o gbar.o -lirc -liomp5 -lpthread -lc -L&lt;icx_dir&gt;/lib</td>
</tr>
</tbody>
</table>

Alternatively, you could use the Intel oneAPI DPC++/C++ Compiler to link the application, but need to pass multiple gfortran libraries using the -l options on the link line:

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed C and Fortran sources</td>
<td>gfortran -fopenmp -c foo.f</td>
</tr>
</tbody>
</table>
### Thread Affinity Interface (Linux* and Windows*)

The Intel® runtime library has the ability to bind OpenMP* threads to physical processing units. The interface is controlled using the **KMP_AFFINITY** environment variable. Depending on the system (machine) topology, application, and operating system, thread affinity can have a dramatic effect on the application speed.

**Thread affinity** restricts execution of certain threads (virtual execution units) to a subset of the physical processing units in a multiprocessor computer. Depending upon the topology of the machine, thread affinity can have a dramatic effect on the execution speed of a program.

Thread affinity is supported on Windows* systems and versions of Linux* systems that have kernel support for thread affinity.

The Intel OpenMP runtime library has the ability to bind OpenMP threads to physical processing units. There are three types of interfaces you can use to specify this binding, which are collectively referred to as the Intel OpenMP Thread Affinity Interface:

- **The high-level affinity interface** uses an environment variable to determine the machine topology and assigns OpenMP threads to the processors based upon their physical location in the machine. This interface is controlled entirely by the **KMP_AFFINITY** environment variable.
- **The mid-level affinity interface** uses an environment variable to explicitly specify which processors (labeled with integer IDs) are bound to OpenMP threads. This interface provides compatibility with the GCC* **GOMP_AFFINITY** environment variable, but you can also invoke it by using the **KMP_AFFINITY** environment variable. The **GOMP_AFFINITY** environment variable is supported on Linux systems only, but users on Windows or Linux systems can use the similar functionality provided by the **KMP_AFFINITY** environment variable.
- **The low-level affinity interface** uses APIs to enable OpenMP threads to make calls into the OpenMP runtime library to explicitly specify the set of processors on which they are to be run. This interface is similar in nature to **sched_setaffinity** and related functions on Linux systems or to **SetThreadAffinityMask** and related functions on Windows systems. In addition, you can specify certain options of the **KMP_AFFINITY** environment variable to affect the behavior of the low-level API interface. For example, you can set the affinity type **KMP_AFFINITY** to disabled, which disables the low-level affinity interface, or you could use the **KMP_AFFINITY** or **GOMP_AFFINITY** environment variables to set the initial affinity mask, and then retrieve the mask with the low-level API interface.

The following terms are used in this section:

- The total number of processing elements on the machine is referred to as the number of **OS thread contexts**.
- Each processing element is referred to as an Operating System processor, or **OS proc**.
- Each OS processor has a unique integer identifier associated with it, called an **OS proc ID**.
- The term **package** refers to a single or multi-core processor chip.
- The term **OpenMP Global Thread ID** (GTID) refers to an integer which uniquely identifies all threads known to the Intel OpenMP runtime library. The thread that first initializes the library is given GTID 0. In the normal case where all other threads are created by the library and when there is no nested parallelism, then \( n \)-**threads-var** - 1 new threads are created with GTIDs ranging from 1 to \( n \)-**threads-var** - 1, and each thread's GTID is equal to the OpenMP thread number returned by function

---

### See Also

- openmp, Qopenmp compiler option
- Using IPO
- OpenMP* Support Libraries
- qopenmp-link, Qopenmp-link compiler option

---

### Commands

<table>
<thead>
<tr>
<th>Type of File</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>icx -qopenmp -c ibar.c</td>
</tr>
<tr>
<td></td>
<td>icx -qopenmp foo.o bar.o -lgfortranbegin -lgfortran</td>
</tr>
</tbody>
</table>
omp_get_thread_num(). The high-level and mid-level interfaces rely heavily on this concept. Hence, their usefulness is limited in programs containing nested parallelism. The low-level interface does not make use of the concept of a GTID and can be used by programs containing arbitrarily many levels of parallelism.

Some environment variables are available for both Intel® microprocessors and non-Intel microprocessors, but may perform additional optimizations for Intel® microprocessors than for non-Intel microprocessors.

The KMP_AFFINITY Environment Variable

**NOTE**
You must set the KMP_AFFINITY environment variable before the first parallel region, or certain API calls including `omp_get_max_threads()`, `omp_get_num_procs()` and any affinity API calls, as described in Low Level Affinity API, below.

The KMP_AFFINITY environment variable uses the following general syntax:

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMP_AFFINITY=[&lt;modifier&gt;,...,]&lt;type&gt;[,&lt;permute&gt;][,&lt;offset&gt;]</td>
</tr>
</tbody>
</table>

For example, to list a machine topology map, specify `KMP_AFFINITY=verbose,none` to use a **modifier** of **verbose** and a **type** of **none**.

The following table describes the supported specific arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifier</td>
<td>noverbose</td>
<td>Optional. String consisting of key and specifier.</td>
</tr>
<tr>
<td></td>
<td>respect</td>
<td>• granularity=&lt;specifier&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• takes takes the following specifiers: fine, thread, core, tile, die, node, group, and socket</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• norespect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• noverbose</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• nowarnings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• proclist={&lt;proc-list&gt;}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• respect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• verbose</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• warnings</td>
</tr>
</tbody>
</table>

The syntax for `<proc-list>` is explained in mid-level affinity interface.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>none</td>
<td>Required string. Indicates the thread affinity to use.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• balanced</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• compact</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• explicit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• scatter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• logical (deprecated; instead use compact, but omit any permute value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• physical (deprecated; instead use scatter, possibly with an offset value)</td>
</tr>
<tr>
<td>permute</td>
<td>0</td>
<td>Optional. Positive integer value. Not valid with type values of explicit, none, or disabled.</td>
</tr>
<tr>
<td>offset</td>
<td>0</td>
<td>Optional. Positive integer value. Not valid with type values of explicit, none, or disabled.</td>
</tr>
</tbody>
</table>

**Affinity Types**

Type is the only required argument.

**type = none (default)**

Does not bind OpenMP threads to particular thread contexts; however, if the operating system supports affinity, the compiler still uses the OpenMP thread affinity interface to determine machine topology. Specify `KMP_AFFINITY=verbose,none` to list a machine topology map.

**type = balanced**

**NOTE** On Windows with multiple processor groups, the norespect affinity modifier is assumed when the process affinity mask equals a single processor group (which is default on Windows). Otherwise, the respect affinity modifier is used.

The logical and physical types are deprecated but supported for backward compatibility.
Places threads on separate cores until all cores have at least one thread, similar to the scatter type. However, when the runtime must use multiple hardware thread contexts on the same core, the balanced type ensures that the OpenMP thread numbers are close to each other, which scatter does not do. This affinity type is supported on the CPU only for single socket systems.

**NOTE**

The OpenMP* environment variable OMP_PROC_BIND=spread is similar to KMP_AFFINITY=balanced and is available on all platforms, including multi-socket CPU systems.

### type = compact

Specifying compact assigns the OpenMP thread \( n \) to a free thread context as close as possible to the context where the \( n \) OpenMP thread was placed. For example, in a topology map, the nearer a node is to the root, the more significance the node has when sorting the threads.

### type = disabled

Specifying disabled completely disables the thread affinity interfaces. This forces the OpenMP run-time library to behave as if the affinity interface was not supported by the operating system. This includes the low-level API interfaces such as kmp_set_affinity and kmp_get_affinity, which have no effect and will return a nonzero error code.

### type = explicit

Specifying explicit assigns OpenMP threads to a list of OS proc IDs that have been explicitly specified by using the proclist= modifier, which is required for this affinity type. See Explicitly Specifying OS Proc IDs (GOMP_CPU_AFFINITY).

### type = scatter

Specifying scatter distributes the threads as evenly as possible across the entire system. scatter is the opposite of compact; so the leaves of the node are most significant when sorting through the machine topology map.

#### Deprecated Types: logical and physical

Types logical and physical are deprecated and may become unsupported in a future release. Both are supported for backward compatibility.

For logical and physical affinity types, a single trailing integer is interpreted as an offset specifier instead of a permute specifier. In contrast, with compact and scatter types, a single trailing integer is interpreted as a permute specifier.

- Specifying logical assigns OpenMP threads to consecutive logical processors, which are also called hardware thread contexts. The type is equivalent to compact, except that the permute specifier is not allowed. Thus, KMP_AFFINITY=logical,n is equivalent to KMP_AFFINITY=compact,0,n (this equivalence is true regardless of the whether or not a granularity=fine modifier is present).
- Specifying physical assigns threads to consecutive physical processors (cores). For systems where there is only a single thread context per core, the type is equivalent to logical. For systems where multiple thread contexts exist per core, physical is equivalent to compact with a permute specifier of 1; that is, KMP_AFFINITY=physical,n is equivalent to KMP_AFFINITY=compact,1,n (regardless of the whether or not a granularity=fine modifier is present). This equivalence means that when the compiler sorts the map it should permute the innermost level of the machine topology map to the outermost, presumably the thread context level. This type does not support the permute specifier.

#### Examples of Types compact and scatter

The following figure illustrates the topology for a machine with two processors, and each processor has two cores; further, each core has Intel® Hyper-Threading Technology (Intel® HT Technology) enabled.
The following figure also illustrates the binding of OpenMP thread to hardware thread contexts when specifying `KMP_AFFINITY=granularity=fine,compact`.

Specifying `scatter` on the same system as shown in the figure above, the OpenMP threads would be assigned the thread contexts as shown in the following figure, which shows the result of specifying `KMP_AFFINITY=granularity=fine,scatter`.

**permute and offset combinations**

For both `compact` and `scatter`, `permute` and `offset` are allowed; however, if you specify only one integer, the compiler interprets the value as a `permute` specifier. Both `permute` and `offset` default to 0.

The `permute` specifier controls which levels are most significant when sorting the machine topology map. A value for `permute` forces the mappings to make the specified number of most significant levels of the sort the least significant, and it inverts the order of significance. The root node of the tree is not considered a separate level for the sort operations.
The offset specifier indicates the starting position for thread assignment.

The following figure illustrates the result of specifying `KMP_AFFINITY=granularity=fine,compact,0,5`.

Consider the hardware configuration from the previous example, running an OpenMP application which exhibits data sharing between consecutive iterations of loops. We would therefore like consecutive threads to be bound close together, as is done with `KMP_AFFINITY=compact`, so that communication overhead, cache line invalidation overhead, and page thrashing are minimized. Now, suppose the application also had a number of parallel regions which did not utilize all of the available OpenMP threads. It is desirable to avoid binding multiple threads to the same core and leaving other cores not utilized, since a thread normally executes faster on a core where it is not competing for resources with another active thread on the same core. Since a thread normally executes faster on a core where it is not competing for resources with another active thread on the same core, you might want to avoid binding multiple threads to the same core while leaving other cores unused. The following figure illustrates this strategy of using `KMP_AFFINITY=granularity=fine,compact,1,0` as a setting.
The OpenMP thread \( n+1 \) is bound to a thread context as close as possible to OpenMP thread \( n \), but on a different core. Once each core has been assigned one OpenMP thread, the subsequent OpenMP threads are assigned to the available cores in the same order, but they are assigned on different thread contexts.

**Modifier Values for Affinity Types**

Modifiers are optional arguments that precede type. If you do not specify a modifier, the `noverbose`, `respect`, and `granularity=core` modifiers are used automatically.

Modifiers are interpreted in order from left to right, and they may conflict. Following conflicting modifier is ignored. For example, specifying `KMP_AFFINITY=verbose,noverbose,scatter` is therefore equivalent to setting `KMP_AFFINITY=verbose,scatter`.

**modifier = noverbose (default)**

Does not print verbose messages.

**modifier = verbose**

Prints messages concerning the supported affinity. The messages include information about the number of packages, number of cores in each package, number of thread contexts for each core, and OpenMP thread bindings to physical thread contexts.

Information about binding OpenMP threads to physical thread contexts is indirectly shown in the form of the mappings between hardware thread contexts and the operating system (OS) processor (proc) IDs. The affinity mask for each OpenMP thread is printed as a set of OS processor IDs.

For example, specifying `KMP_AFFINITY=verbose,scatter` on a dual core system with two processors, with Intel® Hyper-Threading Technology (Intel® HT Technology) disabled, results in a message listing similar to the following when then program is executed:

```plaintext
Verbose, scatter message

... 
KMP_AFFINITY: Initial OS proc set respected: 0,1,2,3 
KMP_AFFINITY: affinity capable, using hwloc. 
KMP_AFFINITY: 4 available OS procs 
KMP_AFFINITY: Uniform topology 
KMP_AFFINITY: 2 sockets x 2 cores/socket x 1 threads/core (4 total cores) 
KMP_AFFINITY: OS proc to physical thread map: 
  KMP_AFFINITY: OS proc 0 maps to socket 0 core 0 thread 0 
  KMP_AFFINITY: OS proc 2 maps to socket 0 core 1 thread 0 
  KMP_AFFINITY: OS proc 1 maps to socket 3 core 0 thread 0 
  KMP_AFFINITY: OS proc 3 maps to socket 3 core 1 thread 0 
KMP_AFFINITY: pid 79739 tid 79739 thread 0 bound to OS proc set 0 
KMP_AFFINITY: pid 79739 tid 79740 thread 2 bound to OS proc set 2 
KMP_AFFINITY: pid 79739 tid 79741 thread 3 bound to OS proc set 3 
KMP_AFFINITY: pid 79739 tid 79742 thread 1 bound to OS proc set 1 
```

The verbose modifier generates several standard, general messages. The following table summarizes how to read the messages.

<table>
<thead>
<tr>
<th>Message String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;affinity capable&quot;</td>
<td>Indicates that all components (compiler, operating system, and hardware) support affinity, so thread binding is possible.</td>
</tr>
<tr>
<td>&quot;decoding x2APIC ids&quot;</td>
<td>Indicates that the machine topology was discovered by binding a thread to each operating system processor and decoding the output of the <code>cpuid</code> instruction.</td>
</tr>
</tbody>
</table>
The mapping from the operating system processors to thread context ID is printed next. The binding of OpenMP thread context ID is printed next unless the affinity type is none. For more information, see Determining Machine Topology.

**modifier = granularity**

Binding OpenMP threads to particular packages and cores will often result in a performance gain on systems with Intel processors with Intel® Hyper-Threading Technology (Intel® HT Technology) enabled; however, it is usually not beneficial to bind each OpenMP thread to a particular thread context on a specific core. Granularity describes the lowest levels that OpenMP threads are allowed to float within a topology map.

This modifier supports the following additional specifiers.

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td>Default. Allows all the OpenMP threads bound to a core to float between the different thread contexts.</td>
</tr>
<tr>
<td>fine or thread</td>
<td>The finest granularity level. Causes each OpenMP thread to be bound to a single thread context. The two specifiers are functionally equivalent.</td>
</tr>
<tr>
<td>tile, or die, or node, or group, or socket</td>
<td>Allows all the OpenMP threads bound to a tile, or die, or NUMA node, or group, or socket to float between the different thread contexts of cores the tile, or die, or NUMA node, or group, or socket consists of.</td>
</tr>
</tbody>
</table>

Specifying \texttt{KMP\_AFFINITY=verbose,granularity=core,compact} on the same dual core system with two processors as in the previous section, but with Intel® Hyper-Threading Technology (Intel® HT Technology) enabled, results in a message listing similar to the following when the program is executed:

**Verbose, granularity=core,compact message**

\texttt{KMP\_AFFINITY: Initial OS proc set respected: 0-7}
\texttt{KMP\_AFFINITY: decoding x2APIC ids.}
\texttt{KMP\_AFFINITY: 8 available OS procs}
\texttt{KMP\_AFFINITY: Uniform topology}
\texttt{KMP\_AFFINITY: 2 sockets x 2 cores/socket x 2 threads/core (4 total cores)}
\texttt{KMP\_AFFINITY: OS proc to physical thread map:}
\texttt{KMP\_AFFINITY: OS proc 0 maps to socket 0 core 0 thread 0}
\texttt{KMP\_AFFINITY: OS proc 4 maps to socket 0 core 0 thread 1}
\texttt{KMP\_AFFINITY: OS proc 2 maps to socket 0 core 1 thread 0}
\texttt{KMP\_AFFINITY: OS proc 6 maps to socket 0 core 1 thread 1}
\texttt{KMP\_AFFINITY: OS proc 1 maps to socket 3 core 0 thread 0}
\texttt{KMP\_AFFINITY: OS proc 5 maps to socket 3 core 0 thread 1}
The affinity mask for each OpenMP thread is shown in the listing (above) as the set of operating system processor to which the OpenMP thread is bound.

The following figure illustrates the machine topology map, for the above listing, with OpenMP thread bindings.

In contrast, specifying `KMP_AFFINITY=verbose,granularity=fine,compact` or `KMP_AFFINITY=verbose,granularity=thread,compact` binds each OpenMP thread to a single hardware thread context when the program is executed:
The OpenMP to hardware context binding for this example was illustrated in the first example.

Specifying `granularity=fine` will always cause each OpenMP thread to be bound to a single OS processor. This is equivalent to `granularity=thread`, currently the finest granularity level.

modifier = respect (default)

Respect the process' original affinity mask, or more specifically, the affinity mask in place for the thread that initializes the OpenMP run-time library. The behavior differs between Linux and Windows:

- On Windows: Respect original affinity mask for the process.
- On Linux: Respect the affinity mask for the thread that initializes the OpenMP run-time library.

NOTE

On Windows with multiple processor groups, the `norespect` affinity modifier is the default when the process affinity mask equals a single processor group (which is default on Windows). Otherwise, the respect affinity modifier is the default.

Specifying `KMP_AFFINITY=verbose,compact` for the same system used in the previous example, with Intel® Hyper-Threading Technology (Intel® HT Technology) enabled, and invoking the library with an initial affinity mask of \{4,5,6,7\} (thread context 1 on every core) causes the compiler to model the machine as a dual core, two-processor system with Intel® HT Technology disabled.

Because there are four thread contexts accessible on the machine, by default the compiler created four threads for an OpenMP parallel construct.
The following figure illustrates the corresponding machine topology map and threads placement in case eight OpenMP threads requested via \texttt{OMP\_NUM\_THREADS=8}.

![Machine topology map](image)

When using the local \texttt{cpuid} information to determine the machine topology, it is not always possible to distinguish between a machine that does not support Intel® Hyper-Threading Technology (Intel® HT Technology) and a machine that supports it, but has it disabled. Therefore, the compiler does not include a level in the map if the elements (nodes) at that level had no siblings, with the exception that the package level is always modeled. As mentioned earlier, the package level will always appear in the topology map, even if there only a single package in the machine.

\textbf{modifier = norespect}

Do not respect original affinity mask for the process. Binds OpenMP threads to all operating system processors.

In early versions of the OpenMP run-time library that supported only the \texttt{physical} and \texttt{logical} affinity types, \texttt{norespect} was the default and was not recognized as a modifier.

The default was changed to respect when types \texttt{compact} and \texttt{scatter} were added; therefore, thread bindings may have changed with the newer compilers in situations where the application specified a partial initial thread affinity mask.

\textbf{modifier = nowarnings}

Do not print warning messages from the affinity interface.

\textbf{modifier = warnings (default)}

Print warning messages from the affinity interface (default).

\textbf{Determining Machine Topology}

On IA-32 and Intel® 64 architecture systems, if the package has an APIC (Advanced Programmable Interrupt Controller), the compiler will use the \texttt{cpuid} instruction to obtain the \texttt{package id}, \texttt{core id}, and \texttt{thread context id}. Under normal conditions, each thread context on the system is assigned a unique APIC ID at boot time. The compiler obtains other pieces of information obtained by using the \texttt{cpuid} instruction, which together with the number of OS thread contexts (total number of processing elements on the machine), determine how to break the APIC ID down into the \texttt{package ID}, \texttt{core ID}, and \texttt{thread context ID}.

There are several ways to specify the APIC ID in the \texttt{cpuid} instruction - the legacy method in leaf 4, and the more modern method in leaf 11 and leaf 31. Only 256 unique APIC IDs are available in leaf 4. Leaf 11 and leaf 31 have no such limitation.
Normally, all core ids on a package and all thread context ids on a core are contiguous; however, numbering assignment gaps are common for package ids, as shown in the figure above.

If the compiler cannot determine the machine topology using any other method, but the operating system supports affinity, a warning message is printed, and the topology is assumed to be flat. For example, a flat topology assumes the operating system process \( N \) maps to package \( N \), and there exists only one thread context per core and only one core for each package.

If the machine topology cannot be accurately determined as described above, the user can manually copy /proc/cpuinfo to a temporary file, correct any errors, and specify the machine topology to the OpenMP runtime library via the environment variable KMP_CPUINFO_FILE=<temp_filename>, as described in the section KMP_CPUINFO_FILE and /proc/cpuinfo.

Regardless of the method used in determining the machine topology, if there is only one thread context per core for every core on the machine, the thread context level will not appear in the topology map. If there is only one core per package for every package in the machine, the core level will not appear in the machine topology map. The topology map need not be a full tree, because different packages may contain a different number of cores, and different cores may support a different number of thread contexts.

The package level will always appear in the topology map, even if there only a single package in the machine.

**KMP_CPUINFO_FILE and /proc/cpuinfo**

One of the methods the Intel® oneAPI DPC++/C++ Compiler OpenMP runtime library can use to detect the machine topology on Linux systems is to parse the contents of /proc/cpuinfo. If the contents of this file (or a device mapped into the Linux file system) are insufficient or erroneous, you can consider copying its contents to a writable temporary file <temp_file>, correct it or extend it with the necessary information, and set KMP_CPUINFO_FILE=<temp_file>.

If you do this, the OpenMP runtime library will read the <temp_file> location pointed to by KMP_CPUINFO_FILE instead of the information contained in /proc/cpuinfo or attempting to detect the machine topology by decoding the APIC IDs. That is, the information contained in the <temp_file> overrides these other methods. You can use the KMP_CPUINFO_FILE interface on Windows systems, where /proc/cpuinfo does not exist.

The content of /proc/cpuinfo or <temp_file> should contain a list of entries for each processing element on the machine. Each processor element contains a list of entries (descriptive name and value on each line). A blank line separates the entries for each processor element. Only the following fields are used to determine the machine topology from each entry, either in <temp_file> or /proc/cpuinfo:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor</td>
<td>Specifies the OS ID for the processing element. The OS ID must be unique.</td>
</tr>
<tr>
<td></td>
<td>The processor and physical id fields are the only ones that are required</td>
</tr>
<tr>
<td></td>
<td>to use the interface.</td>
</tr>
<tr>
<td>physical id</td>
<td>Specifies the package ID, which is a physical chip ID. Each package may</td>
</tr>
<tr>
<td></td>
<td>contain multiple cores. The package level always exists in the compiler's</td>
</tr>
<tr>
<td></td>
<td>OpenMP run-time library model of the machine topology.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>core id :</td>
<td>Specifies the core ID. If it does not exist, it defaults to 0. If every package on the machine contains only a single core, the core level will not exist in the machine topology map (even if some of the core ID fields are non-zero).</td>
</tr>
<tr>
<td>apicid :</td>
<td>Specifies the thread ID. If it does not exist, it defaults to 0. If every core on the machine contains only a single thread, the thread level will not exist in the machine topology map (even if some thread ID fields are non-zero).</td>
</tr>
<tr>
<td>node_n id :</td>
<td>This is a extension to the normal contents of /proc/cpuinfo that can be used to specify the nodes at different levels of the memory interconnect on Non-Uniform Memory Access (NUMA) systems. Arbitrarily many levels ( n ) are supported. The node_0 level is closest to the package level; multiple packages comprise a node at level 0. Multiple nodes at level 0 comprise a node at level 1, and so on.</td>
</tr>
</tbody>
</table>

Each entry must be spelled exactly as shown, in lowercase, followed by optional whitespace, a colon (:) , more optional whitespace, then the integer ID. Fields other than those listed are simply ignored.

**NOTE**

It is common for the thread id field to be missing from /proc/cpuinfo on many Linux variants, and for a field labeled siblings to specify the number of threads per node or number of nodes per package. However, the Intel OpenMP runtime library ignores fields labeled siblings so it can distinguish between the thread id and siblings fields. When this situation arises, the warning message Physical node/pkg/core/thread ids not unique appears (unless the type specified is nowarnings).

**Windows Processor Groups**

On a 64-bit Windows operating system, it is possible for multiple processor groups to accommodate more than 64 processors. Each group is limited in size, up to a maximum value of sixty-four (64) processors.

If multiple processor groups are detected, the default is to model the machine as a 2-level tree, where level 0 are for the processors in a group, and level 1 are for the different groups. Threads are assigned to a group until there are as many OpenMP threads bound to the groups as there are processors in the group. Subsequent threads are assigned to the next group, and so on.

By default, threads are allowed to float among all processors in a group, that is to say, granularity equals the group [granularity=group]. You can override this binding and explicitly use another affinity type like compact, scatter, and so on. If you do so, the granularity must be sufficiently fine to prevent a thread from being bound to multiple processors in different groups.

**Using a Specific Machine Topology Modeling Method (KMP_TOPOLOGY_METHOD)**

You can set the KMP_TOPOLOGY_METHOD environment variable to force OpenMP to use a particular machine topology modeling method.
Explicitly Specifying OS Processor IDs (GOMP_CPU_AFFINITY)

**NOTE**
You must set the GOMP_CPU_AFFINITY environment variable before the first parallel region, or certain API calls including omp_get_max_threads(), omp_get_num_procs() and any affinity API calls, as described in Low Level Affinity API, below.

Instead of allowing the library to detect the hardware topology and automatically assign OpenMP threads to processing elements, the user may explicitly specify the assignment by using a list of operating system (OS) processor (proc) IDs. However, this requires knowledge of which processing elements the OS proc IDs represent.

On Linux systems, when using the Intel OpenMP compatibility libraries enabled by the compiler option -qopenmp-lib compat, you can use the GOMP_AFFINITY environment variable to specify a list of OS processor IDs. Its syntax is identical to that accepted by libgomp (assume that <proc_list> produces the entire GOMP_AFFINITY environment string):

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpuid_leaf11</td>
<td>Decodes the APIC identifiers as specified by leaf 11 of the cpuid instruction.</td>
</tr>
<tr>
<td>cpuid_leaf4</td>
<td>Decodes the APIC identifiers as specified in leaf 4 of the cpuid instruction.</td>
</tr>
<tr>
<td>cpuinfo</td>
<td>If KMP_CPUINFO_FILE is not specified, forces OpenMP to parse /proc/cpuinfo to determine the topology (Linux only). If KMP_CPUINFO_FILE is specified as described above, uses it (Windows or Linux).</td>
</tr>
<tr>
<td>group</td>
<td>Models the machine as a 2-level map, with level 0 specifying the different processors in a group, and level 1 specifying the different groups (Windows 64-bit only).</td>
</tr>
<tr>
<td>flat</td>
<td>Models the machine as a flat (linear) list of processors.</td>
</tr>
<tr>
<td>hwloc</td>
<td>Models the machine as the Portable Hardware Locality* (hwloc) library does. This model is the most detailed and includes, but is not limited to: numa nodes, packages, cores, hardware threads, caches, and Windows processor groups.</td>
</tr>
</tbody>
</table>

Explicitly Specifying OS Processor IDs (GOMP_CPU_AFFINITY)
OS processors specified in this list are then assigned to OpenMP threads, in order of OpenMP Global Thread IDs. If more OpenMP threads are created than there are elements in the list, then the assignment occurs modulo the size of the list. That is, OpenMP Global Thread ID \( n \) is bound to list element \( n \mod \text{<list_size>} \).

Consider the machine previously mentioned: a dual core, dual-package machine without Intel® Hyper-Threading Technology (Intel® HT Technology) enabled, where the OS proc IDs are assigned in the same manner as the example in a previous figure. Suppose that the application creates six OpenMP threads instead of 4 (the default), oversubscribing the machine. If \( \text{GOMP_AFFINITY}=3,0-2 \), then OpenMP threads are bound as shown in the figure below, just as should happen when compiling with \text{gcc} and linking with \text{libgomp}:

The same syntax can be used to specify the OS proc ID list in the proclist=[\(<\text{proc_list}>\)] modifier in the \text{KMP_AFFINITY} environment variable string. There is a slight difference: in order to have strictly the same semantics as in the \text{gcc} OpenMP runtime library \text{libgomp: the GOMP_AFFINITY environment variable implies granularity=fine. If you specify the OS proc list in the KMP_AFFINITY environment variable without a granularity= specifier, then the default granularity is not changed. That is, OpenMP threads are allowed to float between the different thread contexts on a single core. Thus GOMP_AFFINITY=<\text{proc_list}> is an alias for KMP_AFFINITY="granularity=fine,proclist=[<\text{proc_list}>],explicit".}

In the \text{KMP_AFFINITY} environment variable string, the syntax is extended to handle operating system processor ID sets. The user may specify a set of operating system processor IDs among which an OpenMP thread may execute ("float") enclosed in brackets:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\text{range}&gt;)</td>
<td>(&lt;\text{proc_id}&gt; - &lt;\text{proc_id}&gt;</td>
</tr>
<tr>
<td>(&lt;\text{proc_id}&gt;)</td>
<td>&lt;\text{positive_int}&gt;</td>
</tr>
</tbody>
</table>

The same syntax can be used to specify the OS proc ID list in the proclist=[\(<\text{proc_list}>\)] modifier in the \text{KMP_AFFINITY} environment variable string. There is a slight difference: in order to have strictly the same semantics as in the \text{gcc} OpenMP runtime library \text{libgomp: the GOMP_AFFINITY environment variable implies granularity=fine. If you specify the OS proc list in the KMP_AFFINITY environment variable without a granularity= specifier, then the default granularity is not changed. That is, OpenMP threads are allowed to float between the different thread contexts on a single core. Thus GOMP_AFFINITY=<\text{proc_list}> is an alias for KMP_AFFINITY="granularity=fine,proclist=[<\text{proc_list}>],explicit".}

In the \text{KMP_AFFINITY} environment variable string, the syntax is extended to handle operating system processor ID sets. The user may specify a set of operating system processor IDs among which an OpenMP thread may execute ("float") enclosed in brackets:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\text{proc_list}&gt;)</td>
<td>(&lt;\text{proc_id}&gt;</td>
</tr>
</tbody>
</table>
This allows functionality similarity to the `granularity=` specifier, but it is more flexible. The OS processors on which an OpenMP thread executes may exclude other OS processors nearby in the machine topology, but include other distant OS processors. Building upon the previous example, we may allow OpenMP threads 2 and 3 to "float" between OS processor 1 and OS processor 2 by using

```
KMP_AFFINITY="granularity=verbose,fine,proclist=[3,0,\{1,2\},\{1,2\}],explicit",
```

as shown in the figure below:

If `verbose` were also specified, the output when the application is executed would include:

```
KMP_AFFINITY: Initial OS proc set respected: 0,1,2,3
KMP_AFFINITY: decoding x2APIC ids.
KMP_AFFINITY: 4 available OS procs
KMP_AFFINITY: Uniform topology
KMP_AFFINITY: 2 sockets x 2 cores/socket x 1 threads/core (4 total cores)
KMP_AFFINITY: OS proc to physical thread map:
KMP_AFFINITY: OS proc 0 maps to socket 0 core 0 thread 0
KMP_AFFINITY: OS proc 2 maps to socket 0 core 1 thread 0
KMP_AFFINITY: OS proc 1 maps to socket 3 core 0 thread 0
KMP_AFFINITY: OS proc 3 maps to socket 3 core 1 thread 0
KMP_AFFINITY: pid 41464 tid 41464 thread 0 bound to OS proc set 3
KMP_AFFINITY: pid 41464 tid 41465 thread 1 bound to OS proc set 0
KMP_AFFINITY: pid 41464 tid 41466 thread 2 bound to OS proc set 1,2
KMP_AFFINITY: pid 41464 tid 41467 thread 3 bound to OS proc set 1,2
KMP_AFFINITY: pid 41464 tid 41468 thread 4 bound to OS proc set 3
KMP_AFFINITY: pid 41464 tid 41469 thread 5 bound to OS proc set 0
```
Low Level Affinity API

Instead of relying on the user to specify the OpenMP thread to OS proc binding by setting an environment variable before program execution starts (or by using the kmp_settings interface before the first parallel region is reached), each OpenMP thread can determine the desired set of OS procs on which it is to execute and bind to them with the kmp_set_affinity API call.

**Caution**
When you use this affinity interface you take complete control of the hardware resources on which your threads run. To do that sensibly you need to understand in detail how the logical CPUs, the enumeration of hardware threads controlled by the OS, map to the physical hardware of the specific machine on which you are running. That mapping can be, and likely is, different on different machines, so you risk binding machine-specific information into your code, which can result in explicitly forcing bad affinities when your code runs on a different machine. And if you are concerned with optimization at this level of detail, your code is probably valuable, and therefore will probably move to another machine.

This interface may also allow you to ignore the resource limitations that were set by the program startup mechanism, such as Message Passing Interface (MPI), specifically to prevent multiple OpenMP processes on the same node from using the same hardware threads. Again, this can result in explicitly forcing affinities that cause bad performance, and the OpenMP runtime will neither prevent this from happening, nor warn you when it does. These are expert interfaces and you must use them with caution.

It is recommended, therefore, to use the higher level affinity settings if you possibly can, because they are more portable and do not require this low level knowledge.

The C/C++ API interfaces follow, where the type name kmp_affinity_mask_t is defined in omp.h:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int kmp_set_affinity (kmp_affinity_mask_t *mask)</td>
<td>Sets the affinity mask for the current OpenMP thread to *mask, where *mask is a set of OS proc IDs that has been created using the API calls listed below, and the thread will only execute on OS procs in the set. Returns either a zero (0) upon success or a nonzero error code.</td>
</tr>
<tr>
<td>int kmp_get_affinity (kmp_affinity_mask_t *mask)</td>
<td>Retrieves the affinity mask for the current OpenMP thread, and stores it in *mask, which must have previously been initialized with a call to kmp_create_affinity_mask(). Returns either a zero (0) upon success or a nonzero error code.</td>
</tr>
<tr>
<td>int kmp_get_affinity_max_proc (void)</td>
<td>Returns the maximum OS proc ID that is on the machine, plus 1. All OS proc IDs are guaranteed to be between 0 (inclusive) and kmp_get_affinity_max_proc() (exclusive).</td>
</tr>
<tr>
<td>void kmp_create_affinity_mask (kmp_affinity_mask_t *mask)</td>
<td>Allocates a new OpenMP thread affinity mask, and initializes *mask to the empty set of OS procs. The implementation is free to use an object of kmp_affinity_mask_t either as the set itself, a pointer to the actual set, or an index into a table describing the set. Do not make any assumption as to what the actual representation is.</td>
</tr>
</tbody>
</table>
**Syntax**

| void kmp_destroy_affinity_mask (kmp_affinity_mask_t *mask) | Deallocates the OpenMP thread affinity mask. For each call to kmp_create_affinity_mask(), there should be a corresponding call to kmp_destroy_affinity_mask(). |
| int kmp_set_affinity_mask_proc (int proc, kmp_affinity_mask_t *mask) | Adds the OS proc ID proc to the set *mask, if it is not already. Returns either a zero (0) upon success or a nonzero error code. |
| int kmp_unset_affinity_mask_proc (int proc, kmp_affinity_mask_t *mask) | If the OS proc ID proc is in the set *mask, it removes it. Returns either a zero (0) upon success or a nonzero error code. |
| int kmp_get_affinity_mask_proc (int proc, kmp_affinity_mask_t *mask) | Returns 1 if the OS proc ID proc is in the set *mask; if not, it returns 0. |

Once an OpenMP thread has set its own affinity mask via a successful call to kmp_set_affinity(), then that thread remains bound to the corresponding OS proc set until at least the end of the parallel region, unless reset via a subsequent call to kmp_set_affinity().

Between parallel regions, the affinity mask (and the corresponding OpenMP thread to OS proc bindings) can be considered thread private data objects, and have the same persistence as described in the OpenMP Application Program Interface. For more information, see the OpenMP API specification (http://www.openmp.org), some relevant parts of which are provided below:

In order for the affinity mask and thread binding to persist between two consecutive active parallel regions, all three of the following conditions must hold:

- Neither parallel region is nested inside another explicit parallel region.
- The number of threads used to execute both parallel regions is the same.
- The value of the dyn-var internal control variable in the enclosing task region is false at entry to both parallel regions."

Therefore, by creating a parallel region at the start of the program whose sole purpose is to set the affinity mask for each thread, you can mimic the behavior of the KMP_AFFINITY environment variable with low-level affinity API calls, if program execution obeys the three aforementioned rules from the OpenMP specification.

The following example shows how these low-level interfaces can be used. This code binds the executing thread to the specified logical CPU:

```c
// Force the executing thread to execute on logical CPU i
// Returns 1 on success, 0 on failure.
int forceAffinity(int i){
    kmp_affinity_mask_t mask;
    kmp_create_affinity_mask(&mask);
    kmp_set_affinity_mask_proc(i, &mask);
    return (kmp_set_affinity(&mask) == 0);
}
```
This program fragment was written with knowledge about the mapping of the OS proc IDs to the physical processing elements of the target machine. On another machine, or on the same machine with a different OS installed, the program would still run, but the OpenMP thread to physical processing element bindings could differ and you might be explicitly force a bad distribution.

OpenMP* Memory Spaces and Allocators

For storage and retrieval variables, OpenMP* provides memory known as memory spaces. Different memory spaces have different traits. Depending on how a variable is to be used and accessed determines which memory space is appropriate for allocation of the variable.

Each memory space has a unique allocator that is used to allocate and deallocate memory in that space. The allocators allocate variables in contiguous space that does not overlap any other allocation in the memory space. Multiple memory spaces with different traits may map to a single memory resource.

The behavior of the allocator is affected by the allocator traits that you specify. The allocator traits, their possible values, and their default values are shown in the following table:

<table>
<thead>
<tr>
<th>Allocator Trait</th>
<th>Values That Can Be Specified</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>• all</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td>• cgroup</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• pteam</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• thread</td>
<td></td>
</tr>
<tr>
<td>alignment</td>
<td>A positive integer value that is a power of 2 specifying number of bytes</td>
<td>1 byte</td>
</tr>
<tr>
<td>fallback</td>
<td>• abort_fb</td>
<td>default_mem_fb</td>
</tr>
<tr>
<td></td>
<td>• allocator_fb</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• default_mem_fb</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• null_fb</td>
<td></td>
</tr>
<tr>
<td>fb_data</td>
<td>An allocator handle</td>
<td>None</td>
</tr>
<tr>
<td>partition</td>
<td>• blocked</td>
<td>environment</td>
</tr>
<tr>
<td></td>
<td>• environment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• interleaved</td>
<td>nearest</td>
</tr>
<tr>
<td>pinned</td>
<td>• true</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>• false</td>
<td></td>
</tr>
<tr>
<td>pool_size</td>
<td>a positive integer value</td>
<td>Implementation defined</td>
</tr>
<tr>
<td>sync_hint</td>
<td>• contended</td>
<td>contended</td>
</tr>
<tr>
<td></td>
<td>• uncontended</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• private</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• serialized</td>
<td></td>
</tr>
</tbody>
</table>

The access trait specifies the accessibility of the allocated memory. The following are values you can specify for access:

- **all**
  
  This value indicates that the allocated memory must be accessible by all threads in the device where the memory allocation occurs.

  This is the default setting.

- **cgroup**
This value indicates that the allocated memory must be accessible by all threads of the same contention group as the thread that requested the allocation. Accessing the allocated memory thread that is not part of the same contention group results in undefined behavior.

- **pteam**
  This value indicates that the allocated memory is accessible by all threads that bind to the same parallel region as the thread that requests the allocations. Access to the memory by a thread that does not bind to the same parallel region as the thread that allocated the memory results in undefined behavior.

- **thread**
  This value indicates that the memory allocated is accessible only by the thread that allocated it. Attempts to allocate the memory by another thread result in undefined behavior.

The **alignment** trait specifies how allocated variables will be aligned. Variables will be byte-aligned to at least the value specified for this trait. The default setting is 1 byte. Alignment can also be affected by directives and OpenMP runtime allocator routines that specify alignment requirements.

The **fallback** trait indicates how an allocator behaves if it is unable to satisfy an allocation request. The following are values you can specify for **fallback**:

- **abort_fb**
  This value indicates that the program terminates if the allocation request fails.

- **allocator_fb**
  If this value is specified and the allocation request fails, the allocation will be tried by the allocator specified by the **fb_data** trait.

- **default_mem_fb**
  This value indicates that a failed allocation request will be retried in the **omp_default_mem_space** memory space. All traits for the **omp_default_mem_space** allocator should be set to the default trait values, except the **fallback** trait should be set to **null_fb**. This is the default setting.

- **null_fb**
  This value indicates the allocator returns a zero value when an allocation request fails.

The **fb_data** trait lets you specify a fall back allocator to be used if the requested allocator fails to satisfy the allocation request. The **fallback** trait of the failing allocator must be set to the **fb_data** trait. The **allocator** specified by the **fb_data** trait to be used.

The **partition** trait describes the partitioning of allocated memory over the storage resources represented by the memory space of the allocator. The following are values you can specify for **partition**:

- **blocked**
  This value indicates the allocated memory is partitioned into blocks of memory of approximately equal size with one block per storage resource.

- **environment**
  This value indicates the allocated memory placement is determined by the runtime execution environment. This is the default setting.

- **interleaved**
  This value indicates the allocated memory is distributed in a round-robin fashion across the storage resources.

- **nearest**
  This value indicates that the allocated memory will be placed in the storage resource nearest to the thread that requested the allocation.

If the **pinned** trait has the value **true**, the allocator ensures each allocation made by the allocator will remain in the storage resource at the same location where it was allocated until it is deallocated. The default setting is **false**.

The value of **pool_size** is the total number of bytes of storage available to an allocator when there have been no allocations. The following affect **pool_size**:
If the `access` trait has the value all, the value of `pool_size` is the limit for all allocations for all threads having access to the allocator.

If the `access` trait of the allocator has the value cgroup, the value of `pool_size` is the limit for allocations made from the threads within the same contention group.

For allocators with the `access` trait value of pteam, the value of `pool_size` is the limit for allocations made within the same parallel team.

If the `access` trait has the value thread, the value of `pool_size` is the limit for allocations made from each thread using the allocator.

An allocation request for more space than the value of `pool_size` results in the allocator not fulfilling the allocation request.

The `sync_hint` trait describes the way that multiple threads can access an allocator. The following are values you can specify for `sync_hint`:

- **contended** or **uncontended**
  
  Value contended indicates that many threads are anticipated to make simultaneous allocation requests while the value uncontended indicates that few threads are anticipated to make simultaneous allocation. The default setting is contended.

- **private**
  
  This value indicates that all allocation requests will come from the same thread. Specifying `private` when this is not the case and two or more threads make allocation requests by the same allocator results in undefined behavior.

- **serialized**
  
  This value indicates that only one thread will request an allocation at a given time. The behavior is undefined if two threads request an allocation simultaneously by an allocator whose `sync_hint` value is serialized.

There are five predefined memory spaces in OpenMP:

- The system default memory is referred to as `omp_default_mem_space`.
- Large capacity memory is referred to as `omp_large_cap_mem_space`.
- High bandwidth memory is referred to as `omp_high_bw_mem_space`.
- Low latency memory is referred to as `omp_low_lat_mem_space`.
- Memory designed for optimal storage of constant values is referred to as `omp_const_mem_space`.

It can be initialized with compile-time constant expressions or by using a firstprivate clause.

Writing to variables in `omp_const_mem_space` results in undefined behavior.

There are three additional predefined memory spaces that are extensions to the OpenMP standard:

- `omp_target_host_mem_space` is host memory that is accessible by the device.
- `omp_target_shared_mem_space` is memory that can migrate between the host and the device.
- `omp_target_device_mem_space` is memory that is accessible to the device.

The following table shows the predefined memory allocators, the memory space they are associated with, and the non-default memory trait values they possess.

<table>
<thead>
<tr>
<th>Allocator Name</th>
<th>Associated Memory Space</th>
<th>Non-Default Trait Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp_default_mem_alloc</td>
<td>omp_default_mem_space</td>
<td>fallback=null_fb</td>
</tr>
<tr>
<td>omp_large_cap_mem_alloc</td>
<td>omp_large_cap_mem_space</td>
<td>none</td>
</tr>
<tr>
<td>omp_low_lat_mem_alloc</td>
<td>omp_low_lat_mem_space</td>
<td>none</td>
</tr>
<tr>
<td>omp_high_bw_mem_alloc</td>
<td>omp_high_bw_mem_space</td>
<td>none</td>
</tr>
<tr>
<td>omp_const_mem_alloc</td>
<td>omp_const_mem_space</td>
<td>none</td>
</tr>
<tr>
<td>Allocator Name</td>
<td>Associated Memory Space</td>
<td>Non-Default Trait Values</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>omp_cgroup_mem_alloc</td>
<td>implementation/system defined</td>
<td>access=cgroup</td>
</tr>
<tr>
<td>omp_pteam_mem_alloc</td>
<td>implementation/system defined</td>
<td>access=pteam</td>
</tr>
<tr>
<td>omp_thread_mem_alloc</td>
<td>implementation/system defined</td>
<td>access=thread</td>
</tr>
<tr>
<td>omp_target_host_mem_alloc</td>
<td>omp_target_host_mem_space</td>
<td>none</td>
</tr>
<tr>
<td>omp_target_shared_mem_alloc</td>
<td>omp_target_shared_mem_space</td>
<td>none</td>
</tr>
<tr>
<td>omp_target_device_mem_alloc</td>
<td>omp_target_device_mem_space</td>
<td>none</td>
</tr>
</tbody>
</table>

See Also
OpenMP* Run-time Library Routines

OpenMP* Advanced Issues

This topic discusses how to use the OpenMP* library functions and environment variables and discusses some guidelines for enhancing performance with OpenMP.

OpenMP provides specific function calls, and environment variables. See the following topics to refresh your memory about the primary functions and environment variable used in this topic:

- OpenMP Run-time Library Routines
- OpenMP Environment Variables

To use the function calls, include the omp.h header file. This file is installed in the INCLUDE directory during the compiler installation and compile the application using the /Qopenmp (Windows*) or -qopenmp (Linux*) option.

The following example, which demonstrates how to use the OpenMP functions to print the alphabet, also illustrates several important concepts:

1. When using functions instead of pragmas, your code must be rewritten; rewrites can mean extra debugging, testing, and maintenance efforts.
2. It becomes difficult to compile without OpenMP support.
3. It is very easy to introduce simple bugs, as in the loop (below) that fails to print all the letters of the alphabet when the number of threads is not a multiple of 26.
4. You lose the ability to adjust loop scheduling without creating your own work-queue algorithm, which is a lot of extra effort. You are limited by your own scheduling, which is mostly likely static scheduling as shown in the example.

Example

```c
#include <stdio.h>
#include <omp.h>

int main(void) {
    int i;
    omp_set_num_threads(4);
    ```
Example

```c
#pragma omp parallel private(i)
{
    // OMP_NUM_THREADS is not a multiple of 26,
    // which can be considered a bug in this code.
    int LettersPerThread = 26 / omp_get_num_threads();
    int ThisThreadNum = omp_get_thread_num();
    int StartLetter = 'a'+ThisThreadNum*LettersPerThread;
    int EndLetter = 'a'+ThisThreadNum*LettersPerThread+LettersPerThread;

    for (i=StartLetter; i<EndLetter; i++) { printf("%c", i); }
}
```

Debugging threaded applications is a complex process because debuggers change the run-time performance, which can mask race conditions. Even print statements can mask issues, because they use synchronization and operating system functions. OpenMP itself also adds some complications, because it introduces additional structure by distinguishing private variables and shared variables and inserts additional code. A debugger that supports OpenMP can help you to examine variables and step through threaded code. You can use Intel® Inspector to detect many hard-to-find threading errors analytically. Sometimes, a process of elimination can help identify problems without resorting to sophisticated debugging tools.

Remember that most mistakes are race conditions. Most race conditions are caused by shared variables that really should have been declared private. Start by looking at the variables inside the parallel regions and make sure that the variables are declared private when necessary. Next, check functions called within parallel constructs. By default, variables declared on the stack are private, but the C/C++ keyword `static` changes the variable to be placed on the global heap and therefore shared for OpenMP loops.

The default(none) clause, shown below, can be used to help find those hard-to-spot variables. If you specify default(none), then every variable must be declared with a data-sharing attribute clause.

Example

```c
#pragma omp parallel for default(none) private(x,y) shared(a,b)
```

Another common mistake is using uninitialized variables. Remember that private variables do not have initial values upon entering a parallel construct. Use the `firstprivate` and `lastprivate` clauses to initialize them only when necessary, because doing so adds extra overhead.

If you still can't find the bug, then consider the possibility of reducing the scope. Try a binary-hunt. Force parallel sections to be serial again with `if(0)` on the parallel construct or commenting out the pragma altogether. Another method is to force large chunks of a parallel region to be critical sections. Pick a region of the code that you think contains the bug and place it within a critical section. Try to find the section of code that suddenly works when it is within a critical section and fails when it is not. Now look at the variables, and see if the bug is apparent. If that still doesn't work, try setting the entire program to run in serial by setting the compiler-specific environment variable `KMP_LIBRARY=serial`.

If the code is still not working, and you are not using any OpenMP API function calls, compile it without the `/Qopenmp` (Windows) or `-qopenmp` (Linux) option to make sure the serial version works. If you are using OpenMP API function calls, use the `/Qopenmp-stubs` (Windows) or `-qopenmp-stubs` (Linux) option.

Performance

OpenMP threaded application performance is largely dependent upon the following things:

- The underlying performance of the single-threaded code.
- CPU utilization, idle threads, and load balancing.
- The percentage of the application that is executed in parallel by multiple threads.
• The amount of synchronization and communication among the threads.
• The overhead needed to create, manage, destroy, and synchronize the threads, made worse by the number of single-to-parallel or parallel-to-single transitions called fork-join transitions.
• Performance limitations of shared resources such as memory, bus bandwidth, and CPU execution units.
• Memory conflicts caused by shared memory or falsely shared memory.

Performance always begins with a properly constructed parallel algorithm or application. For example, parallelizing a bubble-sort, even one written in hand-optimized assembly language, is not a good place to start. Keep scalability in mind; creating a program that runs well on two CPUs is not as efficient as creating one that runs well on \( n \) CPUs. With OpenMP, the number of threads is chosen by the compiler, so programs that work well regardless of the number of threads are highly desirable. Producer/consumer architectures are rarely efficient, because they are made specifically for two threads.

Once the algorithm is in place, make sure that the code runs efficiently on the targeted Intel® architecture; a single-threaded version can be a big help. Turn off the /Qopenmp (Windows) or -qopenmp (Linux) option to generate a single-threaded version, or build with the /Qopenmp-stubs (Windows) or -qopenmp-stubs (Linux) option, and run the single-threaded version through the usual set of optimizations.

Once you have gotten the single-threaded performance, it is time to generate the multi-threaded version and start doing some analysis.

Optimizations are really a combination of patience, experimentation, and practice. Make little test programs that mimic the way your application uses the computer resources to get a feel for what things are faster than others. Be sure to try the different scheduling clauses for the parallel sections of code. If the overhead of a parallel region is large compared to the compute time, you may want to use an \texttt{if} clause to execute the section serially.

See Also
OpenMP* Run-time Library Routines
Worksharing Using OpenMP*
openmp, Qopenmp
openmp-stubs, Qopenmp-stubs

OpenMP* Implementation-Defined Behaviors

This topic summarizes the behaviors that are described as implementation defined in the OpenMP* API specification.

\textbf{NOTE}
Internal Control Variables (ICVs) mentioned below are discussed in the OpenMP API specification.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>single \texttt{construct}</td>
<td>The first thread that encounters the single construct executes the structured block.</td>
</tr>
<tr>
<td>teams \texttt{construct}</td>
<td>The number of teams that are created is equal to 1 if you don’t specify the \texttt{num_teams} clause.</td>
</tr>
<tr>
<td>dist_schedule \texttt{clause}, distribute \texttt{construct}</td>
<td>If you don’t specify the \texttt{dist_schedule} clause, then the schedule for the distribute \texttt{construct} is static.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>omp_set_num_threads</code> routine</td>
<td>If the argument is not a positive integer, then Intel's OpenMP implementation sets the value of the first element of the <code>nthreads-var</code> ICV of the current task to 1.</td>
</tr>
<tr>
<td><code>omp_set_max_active_levels</code> routine</td>
<td>If the argument is a negative integer this call is ignored and the last valid setting is used.</td>
</tr>
<tr>
<td><code>omp_get_max_active_levels</code> routine</td>
<td>When called from within any explicit parallel region the binding thread set, and binding region, if required, for the <code>omp_get_max_active_levels</code> region is the current task region.</td>
</tr>
<tr>
<td><code>OMP_SCHEDULE</code> environment variable</td>
<td>If the value of the variable does not conform to the specified format then the value of the <code>run-sched-var</code> ICV is set to static.</td>
</tr>
<tr>
<td><code>OMP_NUM_THREADS</code> environment variable</td>
<td>If any value of the list specified in the environment variable is negative then the whole list is ignored. If any value of the list is zero then this value is set to 1.</td>
</tr>
<tr>
<td><code>OMP_PROC_BIND</code> environment variable</td>
<td>If the value is not true, false, or a comma separated list of master (deprecated), primary, close, or spread, then Intel's OpenMP implementation sets the value of <code>bind-var</code> ICV to false.</td>
</tr>
<tr>
<td><code>OMP_DYNAMIC</code> environment variable</td>
<td>If the value is neither true nor false, then the implementation sets the value of <code>dyn-var</code> ICV to false.</td>
</tr>
<tr>
<td><code>OMP_NESTED</code> environment variable</td>
<td>If the value is neither true nor false, then the implementation sets the value of <code>nest-var</code> ICV to false.</td>
</tr>
<tr>
<td><code>OMP_STACKSIZE</code> environment variable</td>
<td>If the value does not conform to the specified format or the implementation cannot provide a stack of the specified size, then Intel's OpenMP implementation sets the value of <code>stacksize-var</code> ICV to the default size, which is specified as being from 1MB to 4MB depending on the architecture. On Linux* or macOS*, the implementation can set the value of <code>stacksize-var</code> ICV up to 256MB, respecting the operating system’s stack size limit.</td>
</tr>
<tr>
<td><code>OMP_MAX_ACTIVE_LEVELS</code> environment variable</td>
<td>If the value is a negative integer or is greater than the number of parallel levels an implementation can support, then Intel's OpenMP implementation sets the value of the <code>max-active-levels-var</code> ICV to 1.</td>
</tr>
<tr>
<td><code>OMP_THREAD_LIMIT</code> environment variable</td>
<td>If the requested value is greater than the number of threads an implementation can support, or if the value is a negative integer, then Intel's OpenMP implementation sets the value of the <code>thread-limit-var</code> ICV to the maximum number of threads.</td>
</tr>
</tbody>
</table>
OpenMP* Examples

The following examples show how to use several OpenMP* features.

A Simple Difference Operator

This example shows a simple parallel loop where the amount of work in each iteration is different. Dynamic scheduling is used to improve load balancing.

The `for` pragma has a `nowait` clause because there is an implicit barrier at the end of the parallel region. Therefore it is not necessary to also have a barrier at the end of the `for` region.

```
Example

void for1(float a[], float b[], int n) {
    int i, j;
    #pragma omp parallel shared(a,b,n) {
        #pragma omp for schedule(dynamic,1) private (i,j) nowait
        for (i = 1; i < n; i++)
            for (j = 0; j < i; j++)
                b[j + n*i] = (a[j + n*i] + a[j + n*(i-1)]) / 2.0;
    }
}
```

Two Difference Operators: for Loop Version

The example uses two parallel loops fused to reduce fork/join overhead. The first `for` pragma has a `nowait` clause because all the data used in the second loop is different than all the data used in the first loop.

```
Example

void for2(float a[], float b[], float c[], float d[], int n, int m) {
    int i, j;
    #pragma omp parallel shared(a,b,c,d,n,m) private(i,j) {
        #pragma omp for schedule(dynamic,1) nowait
        for (i = 1; i < n; i++)
            for (j = 0; j < i; j++)
                b[j + n*i] = (a[j + n*i] + a[j + n*(i-1)]) / 2.0;
        #pragma omp for schedule(dynamic,1) nowait
        for (i = 1; i < m; i++)
            for (j = 0; j < i; j++)
                d[j + m*i] = (c[j + m*i] + c[j + m*(i-1)]) / 2.0;
    }
}
```
Two Difference Operators: sections Version

This example demonstrates the use of the sections pragma. The logic is identical to the preceding for pragma example, but uses a sections pragma instead of a for pragma. Here the speedup is limited to two because there are only two units of work whereas in the example above there are \((n-1) + (m-1)\) units of work.

Example

```c
void sections1(float a[], float b[], float c[], float d[], int n, int m) {
    int i, j;
    #pragma omp parallel shared(a,b,c,d,n,m) private(i,j) {
        #pragma omp sections nowait {
            #pragma omp section
                for (i = 1; i < n; i++)
                    for (j = 0; j < i; j++)
                        b[j + n*i] = ( a[j + n*i] + a[j + n*(i-1)] )/2.0;
            #pragma omp section
                for (i = 1; i < m; i++)
                    for (j = 0; j < i; j++)
                        d[j + m*i] = ( c[j + m*i] + c[j + m*(i-1)] )/2.0;
        }
    }
}
```

Updating a Shared Scalar

This example demonstrates how to use a single construct to update an element of the shared array a. The optional nowait clause after the first loop is omitted because it is necessary to wait at the end of the loop before proceeding into the single construct to avoid a race condition.

Example

```c
void sp_1a(float a[], float b[], int n) {
    int i;
    #pragma omp parallel shared(a,b,n) private(i) {
        #pragma omp for
            for (i = 0; i < n; i++)
                a[i] = 1.0 / a[i];
        #pragma omp single
            a[0] = MIN( a[0], 1.0 );
        #pragma omp for nowait
            for (i = 0; i < n; i++)
                b[i] = b[i] / a[i];
    }
}
```

Intel® oneAPI Level Zero Introduction

The objective of the Intel® oneAPI Level Zero (Level Zero) Application Programming Interface (API) is to provide direct-to-metal interfaces to offload accelerator devices. Its programming interface can be tailored to any device needs and can be adapted to support broader set of languages features such as function pointers, virtual functions, unified memory, and I/O capabilities.
Most applications should not require the additional control provided by the Level Zero API. The Level Zero API is intended for providing explicit controls needed by higher-level runtime APIs and libraries.

While initially influenced by other low-level APIs, such as OpenCL™ API and Vulkan*, the Level Zero APIs are designed to evolve independently. While initially influenced by graphics processing unit architecture, the Level Zero APIs are designed to be supportable across different compute device architectures, such as Field Programmable Gate Arrays (FPGAs) and other types of accelerator architectures.

**Intel® oneAPI Level Zero Switch**

**Intel® oneAPI Level Zero Introduction**

Data Parallel C++ (DPC++) is just one of the many components of the oneAPI project. The Intel® oneAPI Level Zero (Level Zero) API provides low-level direct-to-metal interfaces that are tailored to the devices on a oneAPI project. While heavily influenced by other low-level APIs, such as OpenCL™ API, Level Zero is designed to evolve independently.

More information on Level Zero is available in the [oneAPI Specification](#).

**Packages to Install**

The packages you must install are `intel-level-zero-gpu` and `level-zero`.

**Level Zero Loader**

Level Zero is supportable across different oneAPI compute device architectures. The Level Zero loader discovers all Level Zero drivers in the system. In addition, the Level Zero loader is also the Level Zero software development kit: It carries the Level Zero headers and libraries where you build Level Zero programs.

**Notes**

The Level Zero loader does not come with DPC++ and therefore must be installed independently.

More information about the Level Zero loader is available in the [Loader section](#).

**Level Zero GPU Driver**

The first Level Zero driver was created to support an Intel graphics processing unit (GPU): Gen9+. The driver is open-source and regular public releases are maintained. It does not come with DPC++ and therefore must be installed independently. The Level Zero driver and OpenCL™ Driver come in the same package. More info about the Level Zero driver is available at [GitHub](#).

**DPC++ Plugins**

DPC++ targets a variety of devices: CPU, GPU, and Field Programmable Gate Array (FPGA). Different devices can be operated through different low-level drivers, such as OpenCL for FPGA. The Plugin Interface (PI) is a unified DPC++ API for working with different devices in a unified way. Plugins of DPC++ implement specific translations of the PI API into low-level runtime. The Level Zero PI Plugin was created in DPC++ to enable devices supported through the Level Zero system. More information on PI is available at [GitHub](#).

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPC++ Device Selection</td>
<td>The PI for DPC++ performs device discovery of all available devices through all available PI plugins. The same physical hardware device can be seen as multiple different DPC++ devices if multiple plugins support it (for example, OpenCL Gen90 and Level</td>
</tr>
</tbody>
</table>
## Scenario: Discovery of Multiple PI Plugins

The DPC++ runtime performs device selection from the available devices based on device selectors. The device selectors can be user-defined or built in (for example, `gpu_selector`). The implication of support for the discovery of multiple plugins is that the same GPU card can be seen as multiple different GPU devices available under different PI plugins.

**NOTE** Corresponding runtimes (OpenCL and/or Level Zero) must be installed correctly and independently for PI to see their devices. The DPC++ /SYCL* specification does not define which device will be used if there are multiple devices that match criteria (for example, `is_gpu()`).

## Scenario: Default Preference is Given to a Level Zero GPU on Linux*

By default, if no special action is taken and the Level Zero runtime reports support for the installed GPU, then the DPC++ runtime uses the installed GPU. This is true for standard built-in device selectors and custom device selectors, where no action is taken to change the default behavior. Currently, on Windows*, the preference is given to an OpenCL GPU.

Devices that are not supported with the Level Zero runtime (CPU/FPGA) continue to run with OpenCL.

**How to Change the Default Preference**

Use the `SYCL_BE` environment variable to change the default preference. The valid values are `PI_OPENCL` and `PI_LEVEL0`.

For example, if you specify `SYCL_BE=PI_OPENCL` and the PI OpenCL plugin reports the availability of the device of the required type, then that device is used. It overrides the default preference that is given to the Level Zero GPU, if the GPU is supported by the installed version of OpenCL.

**NOTE** The `SYCL_BE` setting only works when there are multiple choices.

**Recommendation** If your code does not work, try running it with `SYCL_BE=PI_OPENCL` to see if the problem is related to Level Zero.
### Intel® oneAPI Level Zero Backend Specification

#### Introduction

This extension introduces a Level Zero backend for Data Parallel C++ (DPC++), which is built on top of Level Zero runtime enabled with the oneAPI Level Zero Specification. The supported targets are Intel GPUs, starting with Gen9.

**NOTE** This specification is a draft. It is not complete or exhaustive in its descriptions. More information, including explanations on mapping the Data Parallel C++ (DPC++) programming model to a Level Zero API, is forthcoming. In the future, it will conform to the SYCL* 2020 spec.

#### Prerequisites

The Level Zero loader and drivers must be installed on your system for the DPC++ runtime to recognize and enable the Level Zero backend. Visit Intel® oneAPI DPC++/C++ Compiler System Requirements for specific instructions.
User-visible Level Zero Backend Selection and Default Backend

The Level Zero backend is added to the `cl::sycl::backend` enumeration with:

```cpp
class backend {
    // ...
    level_zero,
    // ...
};
```

The sections below explain the different ways the Level Zero backend can be selected.

Through an Environment Variable

The `SYCL_DEVICE_FILTER` environment variable limits the DPC++ runtime to use only a subset of the system's devices. By using `level_zero` for the backend in `SYCL_DEVICE_FILTER`, you can select the use of Level Zero as a DPC++ backend. For more information, see the Environment Variables.

Through a Programming API

The Filter Selector extension is described in SYCL* Proposals: Filter Selector. Similar to how the `SYCL_DEVICE_FILTER` applies filtering to the entire process, this device selector can be used to programmatically select the Level Zero backend.

If the environment variable or filtering device selector is NOT used, the implementation chooses the Level Zero backend for GPU devices that are supported by the installed Level Zero runtime. The serving backend for a DPC++ platform can be queried with the `get_backend()` member function of the `cl::sycl::platform` command.

Interoperability with the Level Zero API

The sections below describe the various interoperabilities that are possible between DPC++ and Level Zero. The application must include the following headers to use any of the inter-operation APIs described in this section. These headers must be included in the order shown:

```cpp
#include "level_zero/ze_api.h"
#include "sycl/backend/level_zero.hpp"
```

Mapping of DPC++ Objects to Level Zero Handles

These DPC++ objects encapsulate the corresponding Level Zero handles:

<table>
<thead>
<tr>
<th>DPC++ Object</th>
<th>Level Zero Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td><code>ze_driver_handle_t</code></td>
</tr>
<tr>
<td>Device</td>
<td><code>ze_device_handle_t</code></td>
</tr>
<tr>
<td>Context</td>
<td><code>ze_context_handle_t</code></td>
</tr>
<tr>
<td>Queue</td>
<td><code>ze_command_queue_handle_t</code></td>
</tr>
<tr>
<td>Program</td>
<td><code>ze_module_handle_t</code></td>
</tr>
</tbody>
</table>

Obtaining Built-in Level Zero Handles from DPC++ Objects

The `get_native<cl::sycl::backend::level_zero>()` member function is how you can use a raw native Level Zero handle to obtain a specific DPC++ object. The function is supported for the DPC++ `platform`, `device`, `context`, `queue`, `event` and `program` classes. You can use a free-function defined in the `cl::sycl` namespace instead of the member function with:

```cpp
template <backend BackendName, class SyclObjectT>
auto get_native(const SyclObjectT &Obj) ->
    typename interop<BackendName, SyclObjectT>::type;
```
Construct a DPC++ Object from a Level Zero Handle

The following free functions, defined in the `cl::sycl::level_zero` namespace, allow an application to create a DPC++ object that encapsulates a corresponding Level Zero object:

<table>
<thead>
<tr>
<th>Level Zero Interoperability Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>make&lt;platform&gt;(ze_driver_handle_t);</code></td>
<td>Constructs a DPC++ platform instance with <code>ze_driver_handle_t</code>.</td>
</tr>
<tr>
<td><code>make&lt;device&gt;(const platform &amp;, ze_device_handle_t);</code></td>
<td>Constructs a DPC++ device instance with <code>ze_device_handle_t</code>. The platform argument gives a DPC++ platform, which encapsulates a Level Zero driver that supports the passed Level Zero device.</td>
</tr>
<tr>
<td><code>make&lt;context&gt;(const vector_class&lt;device&gt; &amp;, ze_context_handle_t);</code></td>
<td>Constructs a DPC++ context instance with <code>ze_context_handle_t</code>. The context is created against the devices that are passed in. You must give at least one device and all the devices must be from the same DPC++ platform (from the same Level Zero driver).</td>
</tr>
<tr>
<td><code>make&lt;queue&gt;(const context &amp;, ze_command_queue_handle_t);</code></td>
<td>Constructs a DPC++ queue instance with <code>ze_command_queue_handle_t</code>. The context argument must be a valid DPC++ context that encapsulates a Level Zero context. The queue is attached to the first device in the passed DPC++ context.</td>
</tr>
<tr>
<td><code>make&lt;program&gt;(const context &amp;, ze_module_handle_t);</code></td>
<td>Constructs a DPC++ program instance with <code>ze_module_handle_t</code>. The context argument must be a valid DPC++ that encapsulates a Level Zero context. The Level Zero module must be fully linked (example: it does not require further linking through <code>zeModuleDynamicLink</code>) and then the DPC++ program is created in the linked state.</td>
</tr>
</tbody>
</table>

Level Zero Handle Ownership and Thread-safety

The Level Zero runtime does not do reference-counting of its objects, so it is crucial to adhere to these practices of how Level Zero handles are managed:

- **DPC++ Runtime Takes Ownership**: Whenever the application creates a DPC++ object from the corresponding Level Zero handle, via one of the `make<T>()` functions, the DPC++ runtime takes ownership of the Level Zero handle. The application must not use the Level Zero handle after the last host copy of the DPC++ object is destroyed. The application must not destroy the Level Zero handle. For more information see the SYCL Common Reference Semantics section: https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf
- **DPC++ Runtime Assumes Ownership**: The application may call the `get_native<T>()` member function of a DPC++ object to retrieve the underlying Level Zero handle, however, the DPC++ runtime continues to retain ownership of this handle. The application must not use this handle after the last host copy of the DPC++ object is destroyed. The application must not destroy the Level Zero handle. For more information see the SYCL Common Reference Semantics section: https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf
• **Considerations for Multi-threaded Environment**: The Level Zero API is not thread-safe, refer to Multithreading and Concurrency for more information. Applications must make sure that the Level Zero handles are not used simultaneously from different threads. The DPC++ runtime takes ownership of the Level Zero handles and should not attempt further direct use of those handles.

## Vectorization

Vectorization is the process of converting an algorithm from a scalar implementation, which does an operation one pair of operands at a time, to a vector process where a single instruction can refer to a vector (a series of adjacent values).

### Automatic Vectorization

The automatic vectorizer (also called the auto-vectorizer) is a component of the compiler that automatically uses SIMD instructions in the Intel® Streaming SIMD Extensions (Intel® SSE, Intel® SSE2, Intel® SSE3 and Intel® SSE4), Supplemental Streaming SIMD Extensions (SSSE3) instruction sets, Intel® Advanced Vector Extensions (Intel® AVX, Intel® AVX2) instruction sets, and Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instruction set. The vectorizer detects operations in the program that can be done in parallel and converts the sequential operations to parallel; for example, the vectorizer converts the sequential SIMD instruction that processes up to 16 elements into a parallel operation, depending on the data type.

Automatic vectorization occurs when the compiler generates packed SIMD instructions to unroll a loop. Because the packed instructions operate on more than one data element at a time, the loop executes more efficiently. This process is referred to as auto-vectorization only to emphasize that the compiler identifies and optimizes suitable loops on its own, without external input. However, it is useful to note that in some cases, certain keywords or directives may be applied in the code for auto-vectorization to occur.

The compiler supports a variety of auto-vectorizing hints that can help the compiler to generate effective vector instructions. Automatic vectorization is supported on IA-32 (for C++ only) and Intel® 64 architectures. Intel® Advisor, a separate tool included in the Intel® oneAPI Base Toolkit, provides a Vectorization Advisor feature that can analyze the compiler's optimization reports and make recommendations for enhancing vectorization.

**NOTE**

Using this option enables vectorization at default optimization levels for both Intel® microprocessors and non-Intel microprocessors. Vectorization may call library routines that can result in additional performance gain on Intel® microprocessors than on non-Intel microprocessors.

### Programming Guidelines for Vectorization

The goal of including the vectorizer component in the Intel® oneAPI DPC++/C++ Compiler is to exploit single-instruction multiple data (SIMD) processing automatically. Users can help by supplying the compiler with additional information; for example, by using auto-vectorizer hints or pragmas.

**NOTE**

Using this option enables vectorization at default optimization levels for both Intel® microprocessors and non-Intel microprocessors. Vectorization may call library routines that can result in additional performance gain on Intel® microprocessors than on non-Intel microprocessors.

### Guidelines to Vectorize Innermost Loops

Follow these guidelines to vectorize innermost loop bodies.
**Use:**
- straight-line code (a single basic block)
- vector data only; that is, arrays and invariant expressions on the right hand side of assignments.
  Array references can appear on the left hand side of assignments.
- only assignment statements.

**Avoid:**
- function calls (other than math library calls)
- non-vectorizable operations (either because the loop cannot be vectorized, or because an operation is emulated through a number of instructions)
- mixing vectorizable types in the same loop (leads to lower resource utilization)
- data-dependent loop exit conditions (leads to loss of vectorization)

To make your code vectorizable, you will often need to make some changes to your loops. You should only make changes needed to enable vectorization, and avoid these common changes:
- loop unrolling, which the compiler performs automatically
- decomposing one loop with several statements in the body into several single-statement loops

**Restrictions**
There are a number of restrictions that you should consider. Vectorization depends on two major factors: hardware and style of source code.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>The compiler is limited by restrictions imposed by the underlying hardware. In the case of Intel® Streaming SIMD Extensions (Intel® SSE), the vector memory operations are limited to stride-1 accesses with a preference to 16-byte-aligned memory references. This means that if the compiler abstractly recognizes a loop as vectorizable, it still might not vectorize it for a distinct target architecture.</td>
</tr>
<tr>
<td>Style of source code</td>
<td>The style in which you write source code can inhibit vectorization. For example, a common problem with global pointers is that they often prevent the compiler from being able to prove that two memory references refer to distinct locations. Consequently, this prevents certain reordering transformations.</td>
</tr>
</tbody>
</table>

Many stylistic issues that prevent automatic vectorization by compilers are found in loop structures. The ambiguity arises from the complexity of the keywords, operators, data references, pointer arithmetic, and memory operations within the loop bodies.

By understanding these limitations and by knowing how to interpret diagnostic messages, you can modify your program to overcome the known limitations and enable effective vectorization.

**Guidelines for Writing Vectorizable Code**
Follow these guidelines to write vectorizable code:
- Use simple for loops. Avoid complex loop termination conditions – the upper iteration limit must be invariant within the loop. For the innermost loop in a nest of loops, you could set the upper limit iteration to be a function of the outer loop indices.
- Write straight-line code. Avoid branches such as switch, goto, or return statements; most function calls; or if constructs that can not be treated as masked assignments.
- Avoid dependencies between loop iterations or at the least, avoid read-after-write dependencies.
- Try to use array notations instead of the use of pointers. C programs in particular impose very few restrictions on the use of pointers; aliased pointers may lead to unexpected dependencies. Without help, the compiler often cannot tell whether it is safe to vectorize code containing pointers.
• Wherever possible, use the loop index directly in array subscripts instead of incrementing a separate counter for use as an array address.
• Access memory efficiently:
  • Favor inner loops with unit stride.
  • Minimize indirect addressing.
  • Align your data to 16-byte boundaries (for Intel® SSE instructions).
• Choose a suitable data layout with care. Most multimedia extension instruction sets are rather sensitive to alignment. The data movement instructions of Intel® SSE, for example, operate much more efficiently on data that is aligned at a 16-byte boundary in memory. Therefore, the success of a vectorizing compiler also depends on its ability to select an appropriate data layout which, in combination with code restructuring (like loop peeling), results in aligned memory accesses throughout the program.
• Use aligned data structures: Data structure alignment is the adjustment of any data object in relation with other objects.

You can use the declaration __declspec(align).

Caution
Use this hint with care. Incorrect usage of aligned data movements result in an exception when using Intel® SSE.

• Use structure of arrays (SoA) instead of array of structures (AoS): An array is the most common type of data structure that contains a contiguous collection of data items that can be accessed by an ordinal index. You can organize this data as an array of structures (AoS) or as a structure of arrays (SoA). While AoS organization is excellent for encapsulation it can be a hindrance for use of vector processing. To make vectorization of the resulting code more effective, you can also select appropriate data structures.

Using Aligned Data Structures

Data structure alignment is the adjustment of any data object with relation to other objects. The Intel® oneAPI DPC++/C++ Compiler may align individual variables to start at certain addresses to speed up memory access. Misaligned memory accesses can incur large performance losses on certain target processors that do not support them in hardware.

Alignment is a property of a memory address, expressed as the numeric address modulo of powers of two. In addition to its address, a single datum also has a size. A datum is called 'naturally aligned' if its address is aligned to its size, otherwise it is called 'misaligned'. For example, an 8-byte floating-point datum is naturally aligned if the address used to identify it is aligned to eight (8).

A data structure is a way of storing data in a computer so that it can be used efficiently. Often, a carefully chosen data structure allows a more efficient algorithm to be used. A well-designed data structure allows a variety of critical operations to be performed, using as little resources - both execution time and memory space - as possible.

Example

```c
struct MyData{
    short   Data1;
    short   Data2;
    short   Data3;
};
```

In the example data structure above, if the type short is stored in two bytes of memory then each member of the data structure is aligned to a boundary of two bytes. Data1 would be at offset 0, Data2 at offset 2 and Data3 at offset 4. The size of this structure is six bytes. The type of each member of the structure usually has a required alignment, meaning that it is aligned on a pre-determined boundary, unless you request
otherwise. In cases where the compiler has taken sub-optimal alignment decisions, you can use the declaration `declspec(align(base,offset))`, where 0 <= offset < base and base is a power of two, to allocate a data structure at offset from a certain base.

Consider as an example, that most of the execution time of an application is spent in a loop of the following form:

```
Example

double a[N], b[N];
...
for (i = 0; i < N; i++) { a[i+1] = b[i] * 3; }
```

If the first element of both arrays is aligned at a 16-byte boundary, then either an unaligned load of elements from `b` or an unaligned store of elements into `a` must be used after vectorization.

**NOTE**
In this case, peeling off an iteration will not help.

However, you can enforce the alignment shown below, which results in two aligned access patterns after vectorization (assuming an 8-byte size for doubles):

```
Example: Alignment Enforcement

declspec(align(16, 8)) double a[N];
decspec(align(16, 0)) double b[N];
/* or simply "align(16)" */
```

If pointer variables are used, the compiler is usually not able to determine the alignment of access patterns at compile time. Consider the following simple `fill()` function:

```
Example

void fill(char *x) {
    int i;
    for (i = 0; i < 1024; i++) { x[i] = 1; }
}
```

Without more information, the compiler cannot make any assumption on the alignment of the memory region accessed by the above loop. At this point, the compiler may decide to vectorize this loop using unaligned data movement instructions or, generate the run-time alignment optimization shown here:

```
Example

peel = x & 0x0f;
if (peel != 0) {
    peel = 16 - peel;
    /* runtime peeling loop */
    for (i = 0; i < peel; i++) { x[i] = 1; }
}
/* aligned access */
for (i = peel; i < 1024; i++) { x[i] = 1; }
```
Run-time optimization provides a generally effective way to obtain aligned access patterns at the expense of a slight increase in code size and testing. If incoming access patterns are guaranteed to be aligned at a 16-byte boundary, you can avoid this overhead with the hint __assume_aligned(x, 16); in the function to convey this information to the compiler.

For example, suppose you can introduce an optimization in the case where a block of memory with address \( n2 \) is aligned on a 16-byte boundary. You could use __assume(n2%16==0).

**Caution**
Use this hint with care. Incorrect use of aligned data movements result in an exception for Intel® SSE.

### Using Structure of Arrays versus Array of Structures

The most common and well-known data structure is the array that contains a contiguous collection of data items, which can be accessed by an ordinal index. This data can be organized as an array of structures (AoS) or as a structure of arrays (SoA). While AoS organization works excellently for encapsulation, for vector processing it works poorly.

You can select appropriate data structures to make vectorization of the resulting code more effective. To illustrate this point, compare the traditional array of structures (AoS) arrangement for storing the \( r, g, b \) components of a set of three-dimensional points with the alternative structure of arrays (SoA) arrangement for storing this set.

**Point Structure with Data in AoS Arrangement**

```c
struct Point{
    float r;
    float g;
    float b;
}
```

**Points Structure with Data in SoA Arrangement**

```c
struct Points{
    float* x;
    float* y;
    float* z;
}
```

With the AoS arrangement, a loop that visits all components of an RGB point before moving to the next point exhibits a good locality of reference because all elements in the fetched cache lines are utilized. The disadvantage of the AoS arrangement is that each individual memory reference in such a loop exhibits a non-unit stride, which, in general, adversely affects vector performance. Furthermore, a loop that visits only one component of all points exhibits less satisfactory locality of reference because many of the elements in the fetched cache lines remain unused.
In contrast, with the SoA arrangement the unit-stride memory references are more amenable to effective vectorization and still exhibit good locality of reference within each of the three data streams. Consequently, an application that uses the SoA arrangement may ultimately outperform an application based on the AoS arrangement when compiled with a vectorizing compiler, even if this performance difference is not directly apparent during the early implementation phase.

Before you start vectorization, try out some simple rules:

- Make your data structures vector-friendly.
- Make sure that inner loop indices correspond to the outermost (last) array index in your data (row-major order).
- Use structure of arrays over array of structures.

For instance when dealing with three-dimensional coordinates, use three separate arrays for each component (SoA), instead of using one array of three-component structures (AoS). To avoid dependencies between loops that will eventually prevent vectorization, use three separate arrays for each component (SoA), instead of one array of three-component structures (AoS). When you use the AoS arrangement, each iteration produces one result by computing XYZ, but it can at best use only 75% of the SSE unit because the forth component is not used. Sometimes, the compiler may only use one component (25%). When you use the SoA arrangement, each iteration produces four results by computing XXXX, YYYY and ZZZZ, using 100% of the SSE unit. A drawback for the SoA arrangement is that your code will likely be three times as long. On the other hand, the compiler might not be able to vectorize AoS arranged code at all.

If your original data layout is in AoS format, you may even want to consider a conversion to SoA on the fly, before the critical loop. If it gets vectorized, it may be worth the effort!

To summarize:

- Use the smallest data types that gives the needed precision to maximize potential SIMD width. (If only 16-bits are needed, using a short rather than an int can make the difference between 8-way or four-way SIMD parallelism, respectively.)
- Avoid mixing data types to minimize type conversions.
- Avoid operations not supported in SIMD hardware.
- Use all the instruction sets available for your processor. Use the appropriate command line option for your processor type, or select the appropriate IDE option (Windows* only):
  - Project > Properties > C/C++ > Code Generation > Intel Processor-Specific Optimization, if your application runs only on Intel® processors.
  - Project > Properties > C/C++ > Code Generation > Enable Enhanced Instruction Set, if your application runs on compatible, non-Intel processors.
- Vectorizing compilers usually have some built-in efficiency heuristics to decide whether vectorization is likely to improve performance. The Intel® oneAPI DPC++/C++ Compiler disables vectorization of loops with many unaligned or non-unit stride data access patterns. If experimentation reveals that vectorization improves performance, you can override this behavior using the #pragma vector always hint before the loop; the compiler vectorizes any loop regardless of the outcome of the efficiency analysis (provided, of course, that vectorization is safe).

See Also

__declspec(align)

Vectorization and Loops

Loop Constructs

Using Automatic Vectorization

Automatic vectorization is supported on Intel® 64 (for C++, DPC++, and Fortran) architectures. The information below will guide you in setting up the auto-vectorizer.
Vectorization Speed-up

Where does the vectorization speedup come from? Consider the following sample code fragment, where \( a, b \) and \( c \) are integer arrays:

<table>
<thead>
<tr>
<th>Sample Code Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (i=0;i&lt;=MAX;i++)</td>
</tr>
<tr>
<td>( c[i] = a[i] + b[i]; )</td>
</tr>
</tbody>
</table>

If vectorization is not enabled, that is, you compile using the \( \text{O1-no-vec} \) (Linux*) or \( /Qvec- \) (Windows*) option, for each iteration, the compiler processes the code such that there is a lot of unused space in the SIMD registers, even though each of the registers could hold three additional integers. If vectorization is enabled (compiled using \( \text{O2 or higher options} \)), the compiler may use the additional registers to perform four additions in a single instruction. The compiler looks for vectorization opportunities whenever you compile at default optimization (\( \text{O2} \)) or higher.

**NOTE**

Using this option enables vectorization at default optimization levels for both Intel® microprocessors and non-Intel microprocessors. Vectorization may call library routines that can result in additional performance gain on Intel® microprocessors than on non-Intel microprocessors.

To get details about the type of loop transformations and optimizations that took place, use the \( [Q]\text{opt-report-phase} \) option by itself or along with the \( [Q]\text{opt-report} \) option.

How significant is the performance enhancement? To evaluate performance enhancement yourself, run \text{vec_samples}:

1. Open an Intel®oneAPI DPC++/C++ Compiler command-line window.
   - **On Windows*:** Under the Start menu item for your Intel product, select an icon under Intel oneAPI 2021 > Intel oneAPI Command Prompt for oneAPI Compilers.
   - **On Linux*:** Source an environment script such as \text{vars.sh} in the \(<\text{installldir}>\) directory and use the attribute appropriate for the architecture.

2. Navigate to the \(<\text{installldir}>\text{Samples}\langle\text{locale}>\text{C++ (for C++ or DPC++) directory. On Windows, unzip the sample project vec_samples.zip to a writable directory. This small application multiplies a vector by a matrix using the following loop:}

<table>
<thead>
<tr>
<th>Example: Vector Matrix Multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (j = 0; j &lt; size2; j++) { b[i] += a[i][j] * x[j]; }</td>
</tr>
</tbody>
</table>

3. Build and run the application, first without enabling auto-vectorization. The default \( \text{O2} \) optimization enables vectorization, so you need to disable it with a separate option. Note the time taken for the application to run.

<table>
<thead>
<tr>
<th>Example: Building and Running an Application without Auto-vectorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>// (Linux) icx -O2 -no-vec Multiply.c -o NoVectMult ./NoVectMult</td>
</tr>
<tr>
<td>// (Windows) icx /O2 /Qvec- Multiply.c /FeNoVectMult NoVectMult</td>
</tr>
</tbody>
</table>

4. Now build and run the application, this time with auto-vectorization. Note the time taken for the application to run.
Example: Building and Running an Application with Auto-vectorization

// (Linux)
vicc -O2 -qopt-report=1 -qopt-report-phase=vec Multiply.c -o VectMult
./VectMult

// (Windows for C++)
icx /O2 /Qopt-report:1 /Qopt-report-phase:vec Multiply.c /FeVectMult VectMult

// (Windows for DPC++)

When you compare the timing of the two runs, you may see that the vectorized version runs faster. The time for the non-vectorized version is only slightly faster than would be obtained by compiling with the O1 option.

Obstacles to Vectorization

The following do not always prevent vectorization, but frequently either prevent it or cause the compiler to decide that vectorization would not be worthwhile.

- **Non-contiguous memory access:** Four consecutive integers or floating-point values, or two consecutive doubles, may be loaded directly from memory in a single SSE instruction. But if the four integers are not adjacent, they must be loaded separately using multiple instructions, which is considerably less efficient. The most common examples of non-contiguous memory access are loops with non-unit stride or with indirect addressing, as in the examples below. The compiler rarely vectorizes such loops, unless the amount of computational work is large compared to the overhead from non-contiguous memory access.

Example: Non-contiguous Memory Access

// arrays accessed with stride 2
for (int i=0; i<SIZE; i+=2)  b[i] += a[i] * x[i];

// inner loop accesses a with stride SIZE
for (int j=0; j<SIZE; j++) {
  for (int i=0; i<SIZE; I++)   b[i] += a[i][j] * x[j];
}

// indirect addressing of x using index array
for (int i=0; i<SIZE; i+=2)  b[i] += a[i] * x[index[i]];

The typical message from the vectorization report is: vectorization possible but seems inefficient, although indirect addressing may also result in the following report: Existence of vector dependence.

- **Data dependencies:** Vectorization entails changes in the order of operations within a loop, since each SIMD instruction operates on several data elements at once. Vectorization is only possible if this change of order does not change the results of the calculation.

  - The simplest case is when data elements that are written (stored to) do not appear in any other iteration of the individual loop. In this case, all the iterations of the original loop are independent of each other, and can be executed in any order, without changing the result. The loop may be safely executed using any parallel method, including vectorization. All the examples considered so far fall into this category.

  - When a variable is written in one iteration and read in a subsequent iteration, there is a “read-after-write” dependency, also known as a flow dependency, as in this example:
### Example: Flow Dependency

```c
A[0]=0;
// this is equivalent to:
```

So the value of j gets propagated to all A[j]. This cannot safely be vectorized: if the first two iterations are executed simultaneously by a SIMD instruction, the value of A[1] is used by the second iteration before it has been calculated by the first iteration.

- When a variable is read in one iteration and written in a subsequent iteration, this is a write-after-read dependency, also known as an anti-dependency, as in the following example:

### Example: Write-after-read Dependency

```c
// this is equivalent to:
```

This write-after-read dependency is not safe for general parallel execution, since the iteration with the write may execute before the iteration with the read. However, for vectorization, no iteration with a higher value of j can complete before an iteration with a lower value of j, and so vectorization is safe (that is, it gives the same result as non-vectorized code) in this case. The following example, however, may not be safe, since vectorization might cause some elements of A to be overwritten by the first SIMD instruction before being used for the second SIMD instruction.

### Example: Unsafe Vectorization

```c
for (j=1; j<MAX; j++)  {
    B[j]=A[j]*2;
}
// this is equivalent to:
```

- Read-after-read situations are not really dependencies, and do not prevent vectorization or parallel execution. If a variable is unwritten, it does not matter how often it is read.
- Write-after-write, or 'output', dependencies, where the same variable is written to in more than one iteration, are in general unsafe for parallel execution, including vectorization.
- One important exception, that apparently contains all of the above types of dependency:

### Example: Dependency Exception

```c
sum=0;
for (j=1; j<MAX; j++)  sum = sum + A[j]*B[j]
```
Although `sum` is both read and written in every iteration, the compiler recognizes such reduction idioms, and is able to vectorize them safely. The loop in the first example was another example of a reduction, with a loop-invariant array element in place of a scalar.

These types of dependencies between loop iterations are sometimes known as loop-carried dependencies.

The above examples are of proven dependencies. The compiler cannot safely vectorize a loop if there is even a potential dependency. Consider the following example:

<table>
<thead>
<tr>
<th>Example: Potential Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (i = 0; i &lt; size; i++) { c[i] = a[i] * b[i]; }</td>
</tr>
</tbody>
</table>

In the above example, the compiler needs to determine whether, for some iteration `i`, `c[i]` might refer to the same memory location as `a[i]` or `b[i]` for a different iteration. Such memory locations are sometimes said to be aliased. For example, if `a[i]` pointed to the same memory location as `c[i-1]`, there would be a read-after-write dependency as in the earlier example. If the compiler cannot exclude this possibility, it will not vectorize the loop unless you provide the compiler with hints.

**Helping the Intel® oneAPI DPC++/C++ Compiler to Vectorize**

Sometimes the compiler has insufficient information to decide to vectorize a loop. There are several ways to provide additional information to the compiler:

- **Pragmas:**
  - `#pragma ivdep:` may be used to tell the compiler that it may safely ignore any potential data dependencies. (The compiler will not ignore proven dependencies). Use of this pragma when there are dependencies may lead to incorrect results.

There are cases where the compiler cannot tell by a static dependency analysis that it is safe to vectorize. Consider the following loop:

<table>
<thead>
<tr>
<th>Loop Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>void copy(char *cp_a, char *cp_b, int n) {</td>
</tr>
<tr>
<td>for (int i = 0; i &lt; n; i++) { cp_a[i] = cp_b[i]; }</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Without more information, a vectorizing compiler must conservatively assume that the memory regions accessed by the pointer variables `cp_a` and `cp_b` may (partially) overlap, which gives rise to potential data dependencies that prohibit straightforward conversion of this loop into SIMD instructions. At this point, the compiler may decide to keep the loop serial or, as done by the Intel® oneAPI DPC++/C++ Compiler, generate a run-time test for overlap, where the loop in the true-branch can be converted into SIMD instructions:

<table>
<thead>
<tr>
<th>Example: True-branch Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (cp_a + n &lt; cp_b</td>
</tr>
<tr>
<td>/* vector loop */</td>
</tr>
<tr>
<td>for (int i = 0; i &lt; n; i++) cp_a[i] = cp_b[i];</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>/* serial loop */</td>
</tr>
<tr>
<td>for (int i = 0; i &lt; n; i++) cp_a[i] = cp_b[i];</td>
</tr>
</tbody>
</table>

Run-time data-dependency testing provides a generally effective way to exploit implicit parallelism in C or C++ code at the expense of a slight increase in code size and testing overhead. If the function `copy` is only used in specific ways, however, you can assist the vectorizing compiler as follows:
If the function is mainly used for small values of n or for overlapping memory regions, you can simply prevent vectorization and, hence, the corresponding run-time overhead by inserting a #pragma novector hint before the loop.

Conversely, if the loop is guaranteed to operate on non-overlapping memory regions, you can provide this information to the compiler by means of a #pragma ivdep hint before the loop, which informs the compiler that conservatively assumed data dependencies that prevent vectorization can be ignored. This results in vectorization of the loop without run-time data-dependency testing.

Example: Ignoring Data Dependencies with #pragma ivdep

```c
#pragma ivdep
void copy(char *cp_a, char *cp_b, int n) {
    for (int i = 0; i < n; i++) { cp_a[i] = cp_b[i]; }
}
```

NOTE
You can also use the restrict keyword.

• #pragma loop count (n): may be used to advise the compiler of the typical trip count of the loop. This may help the compiler to decide whether vectorization is worthwhile, or whether or not it should generate alternative code paths for the loop.
• #pragma vector always: asks the compiler to vectorize the loop if it is safe to do so, whether or not the compiler thinks that will improve performance.
• #pragma vector align: asserts that data within the following loop is aligned (to a 16-byte boundary, for Intel® SSE instruction sets).
• #pragma novector: asks the compiler not to vectorize a particular loop.
• #pragma vector nontemporal: gives a hint to the compiler that data will not be reused, and therefore to use streaming stores that bypass cache.

Keywords: The restrict keyword may be used to assert that the memory referenced by a pointer is not aliased, i.e. that it is not accessed in any other way. The keyword requires the use of the [Q]std=c99 compiler option. The example under #pragma ivdep above can also be handled using the restrict keyword.

You may use the restrict keyword in the declarations of cp_a and cp_b, as shown below, to inform the compiler that each pointer variable provides exclusive access to a certain memory region. The restrict qualifier in the argument list lets the compiler know that there are no other aliases to the memory to which the pointers point. In other words, the pointer for which it is used provides the only means of accessing the memory in question in the scope in which the pointers live. Even if the code gets vectorized without the restrict keyword, the compiler checks for aliasing at run-time, if the restrict keyword was used.

Example: Restrict Keyword

```c
void copy(char * __restrict cp_a, char * __restrict cp_b, int n) {
    for (int i = 0; i < n; i++) cp_a[i] = cp_b[i];
}
```

This method is convenient in case the exclusive access property holds for pointer variables that are used in a large portion of code with many loops because it avoids the need to annotate each of the vectorizable loops individually. Note, however, that both the loop-specific #pragma ivdep hint, as well as the pointer variable-specific restrict hint must be used with care because incorrect usage may change the semantics intended in the original program.

Another example is the following loop that may also not get vectorized because of a potential aliasing problem between pointers a, b and c:
Example: Potential Unsupported Loop Structure

```c
void add(float *a, float *b, float *c) {
    for (int i=0; i<SIZE; i++) { c[i] += a[i] + b[i]; }
}
```

If the `restrict` keyword is added to the parameters, the compiler will trust you, that you will not access the memory in question with any other pointer and vectorize the code properly:

Example: Using Pointers with the `Restrict` Keyword

```c
// let the compiler know, the pointers are safe with restrict
void add(float * __restrict a, float * __restrict b, float * __restrict c) {
    for (int i=0; i<SIZE; i++) { c[i] += a[i] + b[i]; }
}
```

The down-side of using `restrict` is that not all compilers support this keyword, so your source code may lose portability.

- **Options/switches:** You can use options to enable different levels of optimizations to achieve automatic vectorization:
  - **Interprocedural optimization (IPO):** Enable IPO using the `-Qipo` option across source files. You provide the compiler with additional information (trip counts, alignment, or data dependencies) about a loop. Enabling IPO may also allow inlining of function calls.
  - **High-level optimizations (HLO):** Enable HLO with option `-O3`. This will enable additional loop optimizations that make it easier for the compiler to vectorize the transformed loops.

See Also

qopt-report, Qopt-report  compiler option

Vectorization and Loops

This topic provides more information on the interaction between the auto-vectorizer and loops.

See Programming Guidelines for Vectorization.

In some rare cases, a successful loop parallelization (either automatically or by means of OpenMP* directives) may affect the messages reported by the compiler for a non-vectorizable loop in a non-intuitive way.

Types of Vectorized Loops

For integer loops, the 128-bit Intel® Streaming SIMD Extensions (Intel® SSE) and the Intel® Advanced Vector Extensions (Intel® AVX) provide SIMD instructions for most arithmetic and logical operators on 32-bit, 16-bit, and 8-bit integer data types, with limited support for the 64-bit integer data type.

Vectorization may proceed if the final precision of integer wrap-around arithmetic is preserved. A 32-bit shift-right operator, for instance, is not vectorized in 16-bit mode if the final stored value is a 16-bit integer. Also, note that because the Intel® SSE and the Intel® AVX instruction sets are not fully orthogonal (shifts on byte operands, for instance, are not supported), not all integer operations can actually be vectorized.

For loops that operate on 32-bit single-precision and 64-bit double-precision floating-point numbers, Intel® SSE provides SIMD instructions for the following arithmetic operators:

- addition (+)
- subtraction (-)
- multiplication (*)
- division (/)
Additionally, Intel® SSE provide SIMD instructions for the binary \texttt{MIN} and \texttt{MAX} and unary \texttt{SQRT} operators. SIMD versions of several other mathematical operators (like the trigonometric functions \texttt{SIN}, \texttt{COS}, and \texttt{TAN}) are supported in software in a vector mathematical run-time library that is provided with the Intel® oneAPI DPC++/C++ Compiler.

To be vectorizable, loops must be:

- **Countable:** The loop trip count must be known at entry to the loop at runtime, though it need not be known at compile time (that is, the trip count can be a variable but the variable must remain constant for the duration of the loop). This implies that exit from the loop must not be data-dependent.
- **Single entry and single exit:** as is implied by stating that the loop must be countable.
- **Contain straight-line code:** SIMD instruction perform the same operation on data elements from multiple iterations of the original loop, therefore, it is not possible for different iterations to have different control flow; that is, they must not branch. It follows that \texttt{switch} statements are not allowed. However, if statements are allowed if they can be implemented as masked assignments, which is usually the case. The calculation is performed for all data elements but the result is stored only for those elements for which the mask evaluates to true.
- **Innermost loop of a nest:** The only exception is if an original outer loop is transformed into an inner loop as a result of some other prior optimization phase, such as unrolling, loop collapsing or interchange, or an original outermost loop is transformed to an innermost loop due to loop materialization.
- **Without function calls:** Even a \texttt{print} statement is sufficient to prevent a loop from getting vectorized. The vectorization report message is typically: non-standard loop is not a vectorization candidate. The two major exceptions are for intrinsic math functions and for functions that may be inlined.

Intrinsic math functions are allowed, because the compiler runtime library contains vectorized versions of these functions. See the table below for a list of these functions; most exist in both float and double versions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>acos</td>
<td>ceil</td>
<td>fabs</td>
<td>round</td>
<td></td>
</tr>
<tr>
<td>acosh</td>
<td>cos</td>
<td>floor</td>
<td>sin</td>
<td></td>
</tr>
<tr>
<td>asin</td>
<td>cosh</td>
<td>fmax</td>
<td>sinh</td>
<td></td>
</tr>
<tr>
<td>asinh</td>
<td>erf</td>
<td>fmin</td>
<td>sqrt</td>
<td></td>
</tr>
<tr>
<td>atan</td>
<td>erfc</td>
<td>log</td>
<td>tan</td>
<td></td>
</tr>
<tr>
<td>atan2</td>
<td>erfinv</td>
<td>log10</td>
<td>tanh</td>
<td></td>
</tr>
<tr>
<td>atanh</td>
<td>exp</td>
<td>log2</td>
<td>trunc</td>
<td></td>
</tr>
<tr>
<td>cbrt</td>
<td>exp2</td>
<td>pow</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Statements in the Loop Body**

The vectorizable operations are different for floating-point and integer data.

**Integer Array Operations**

The statements within the loop body may contain \texttt{char}, \texttt{unsigned char}, \texttt{short}, \texttt{unsigned short}, \texttt{int}, and \texttt{unsigned int}. Calls to functions such as \texttt{sqrt} and \texttt{fabs} are also supported. Arithmetic operations are limited to addition, subtraction, bitwise \texttt{AND}, \texttt{OR}, and \texttt{XOR} operators, division (via run-time library call), multiplication, \texttt{min}, and \texttt{max}. You can mix data types but this may potentially cost you in terms of lowering efficiency. Some example operators where you can mix data types are multiplication, shift, or unary operators.
Other Operations

No statements other than the preceding floating-point and integer operations are allowed. In particular, note that the special __m64, __m128, and __m256 data types are not vectorizable. The loop body cannot contain any function calls. Use of Intel® SSE intrinsics (for example, _mm_add_ps) or Intel® AVX intrinsics (for example, _mm256_add_ps) are not allowed.

Product and Performance Information

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.
Notice revision #20201201

See Also

Programming Guidelines for Vectorization
qopt-report, Qopt-report compiler option

Loop Constructs

Loops can be formed with the usual for and while constructs. Loops must have a single entry and a single exit to be vectorized. The following examples illustrate loop constructs that can and cannot be vectorized.

Example: Vectorizable structure

```c
void vec(float a[], float b[], float c[]) {
    int i = 0;
    while (i < 100) {
        // The if branch is inside body of loop.
        a[i] = b[i] * c[i];
        if (a[i] < 0.0)
            a[i] = 0.0;
        i++;
    }
}
```

The following example shows a loop that cannot be vectorized because of the inherent potential for an early exit from the loop.

Example: Non-vectorizable structure

```c
void no_vec(float a[], float b[], float c[]) {
    int i = 0;
    while (i < 100) {
        if (a[i] < 50)
            // The next statement is a second exit
            // that allows an early exit from the loop.
            break;
        i++;
    }
}
```

Loop Exit Conditions

Loop exit conditions determine the number of iterations a loop executes. For example, fixed indexes for loops determine the iterations. The loop iterations must be countable; in other words, the number of iterations must be expressed as one of the following:
A constant.
A loop invariant term.
A linear function of outermost loop indices.

In the case where a loop's exit depends on computation, the loops are not countable. The examples below show loop constructs that are countable and non-countable.

**Example: Countable Loop**

```c
void cnt1(float a[], float b[], float c[], int n, int lb) {
    int cnt=n, i=0;
    // Exit condition specified by "N-lb+1"
    while (cnt >= lb) {
        // lb is not affected within loop.
        a[i] = b[i] * c[i];
        cnt--;
        i++;
    }
}
```

The following example demonstrates a different countable loop construct.

**Example: Countable Loop**

```c
void cnt2(float a[], float b[], float c[], int m, int n) {
    int i=0, l;
    // Number of iterations is 
    for (l=m; l<n; l+=2) {
        a[i] = b[i] * c[i];
        i++;
    }
}
```

The following examples demonstrate a loop construct that is non-countable due to dependency loop variant count value.

**Example: Non-Countable Loop**

```c
void no_cnt(float a[], float b[], float c[]) {
    int i=0;
    // Iterations dependent on a[i].
    while (a[i]>0.0) {
        a[i] = b[i] * c[i];
        i++;
    }
}
```

**Strip-mining and Cleanup**

Strip-mining, also known as loop sectioning, is a loop transformation technique for enabling SIMD-encodings of loops, as well as a means of improving memory performance. By fragmenting a large loop into smaller segments or strips, this technique transforms the loop structure in two ways:

- By increasing the temporal and spatial locality in the data cache if the data are reusable in different passes of an algorithm.
• By reducing the number of iterations of the loop by a factor of the length of each vector, or number of operations being performed per SIMD operation. In the case of Intel® Streaming SIMD Extensions, this vector or strip-length is reduced by four times: four floating-point data items per single Intel® SSE single-precision floating-point SIMD operation are processed.

First introduced for vectorizers, this technique consists of the generation of code when each vector operation is done for a size less than or equal to the maximum vector length on a given vector machine.

The compiler automatically strip-mines your loop and generates a cleanup loop. For example, assume the compiler attempts to strip-mine the following loop:

Example: Before Vectorization

```c
i=0;
while(i<n) {
    // Original loop code
    a[i]=b[i]+c[i];
    ++i;
}
```

The compiler might handle the strip mining and loop cleaning by restructuring the loop in the following manner:

Example: After Vectorization

```c
// The vectorizer generates the following two loops
i=0;
while(i<(n-n%4)) {
    // Vector strip-mined loop
    // Subscript [i:i+3] denotes SIMD execution
    a[i:i+3]=b[i:i+3]+c[i:i+3];
    i=i+4;
}
while(i<n) {
    // Scalar clean-up loop
    a[i]=b[i]+c[i];
    ++i;
}
```

Loop Blocking

It is possible to treat loop blocking as strip-mining in two or more dimensions. Loop blocking is a useful technique for memory performance optimization. The main purpose of loop blocking is to eliminate as many cache misses as possible. This technique transforms the memory domain into smaller chunks rather than sequentially traversing through the entire memory domain. Each chunk should be small enough to fit all the data for a given computation into the cache, thereby maximizing data reuse.

Consider the following example, loop blocking allows arrays A and B to be blocked into smaller rectangular chunks so that the total combined size of two blocked (A and B) chunks is smaller than cache size, which can improve data reuse.

Example: Original loop

```c
#include <time.h>
#include <stdio.h>
#define MAX 7000

void add(int a[][MAX], int b[][MAX]);
int main() {
```
Example: Original loop

```c
int i, j;
int A[MAX][MAX];
int B[MAX][MAX];
time_t start, elaspe;
int sec;

//Initialize array
for(i=0;i<MAX;i++) {
    for(j=0;j<MAX; j++) {
        A[i][j]=j;
        B[i][j]=j;
    }
}

start= time(NULL);
add(A, B);
elaspe=time(NULL);
sec = elaspe - start;
printf("Time %d",sec); //List time taken to complete add function
}
```

The following example illustrates loop blocking the add function (from the previous example). In order to benefit from this optimization you might have to increase the cache size.

Example: Transformed Loop after Blocking

```c
#include <stdio.h>
#include <time.h>
#define MAX 7000
void add(int a[][MAX], int b[][MAX]);

int main() {
    #define BS 8  //Block size is selected as the loop-blocking factor.
    int i, j;
    int A[MAX][MAX];
    int B[MAX][MAX];
    time_t start, elaspe;
    int sec;

    //initialize array
    for(i=0;i<MAX;i++) {
        for(j=0;j<MAX; j++) {
            A[i][j]=j;
            B[i][j]=j;
        }
    }
```
Example: Transformed Loop after Blocking

```c
start = time(NULL);
add(A, B);
elapse = time(NULL);
sec = elapse - start;
printf("Time %d", sec); //Display time taken to complete loop Blocking function
}
```

void add(int a[][MAX], int b[][MAX]) {
    int i, j, ii, jj;
    for (i = 0; i < MAX; i += BS) {
        for (j = 0; j < MAX; j += BS) {
            for (ii = i; ii < i + BS; ii++) {   //outer loop
                for (jj = j; jj < j + BS; jj++) {  //Array B experiences one cache miss
                    //for every iteration of outer loop
                    a[ii][jj] = a[ii][jj] + b[jj][ii]; //Add the two arrays
                }
            }
        }
    }
}
```

Loop Interchange and Subscripts: Matrix Multiply

Loop interchange is often used for improving memory access patterns. Matrix multiplication is commonly written as shown in the following example:

Example: Typical Matrix Multiplication

```c
void matmul_slow(float *a[], float *b[], float *c[]) {
    int N = 100;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

The use of $B(K, J)$ is not a stride-1 reference and therefore will not be vectorized efficiently.

If the loops are interchanged, however, all the references will become stride-1 as shown in the following example.

Example: Matrix Multiplication with Stride-1

```c
void matmul_fast(float *a[], float *b[], float *c[]) {
    int N = 100;
    for (int i = 0; i < N; i++)
        for (int k = 0; k < N; k++)
            for (int j = 0; j < N; j++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

Interchanging is not always possible because of dependencies, which can lead to different results.
Explicit Vector Programming

This section contains information about explicit vector programming.

User-Mandated or SIMD Vectorization

User-mandated or SIMD vectorization supplements automatic vectorization just like OpenMP* parallelization supplements automatic parallelization. The following figure illustrates this relationship. User-mandated vectorization is implemented as a single-instruction-multiple-data (SIMD) feature and is referred to as SIMD vectorization.

NOTE
The SIMD vectorization feature is available for both Intel® microprocessors and non-Intel microprocessors. Vectorization may call library routines that can result in additional performance gain on Intel® microprocessors than on non-Intel microprocessors.

The following figure illustrates how SIMD vectorization is positioned among various approaches that you can take to generate vector code that exploits vector hardware capabilities. The programs written with SIMD vectorization are very similar to those written using auto-vectorization hints. You can use SIMD vectorization to minimize the amount of code changes that you may have to go through in order to obtain vectorized code.
SIMD vectorization uses the `#pragma omp simd` pragma to effect loop vectorization.

Consider an example in C++ where the function `add_floats()` uses too many unknown pointers for the compiler’s automatic runtime independence check optimization to kick in. You can give a data dependence assertion using the auto-vectorization hint via `#pragma ivdep` and let the compiler decide whether the auto-vectorization optimization should be applied to the loop. Or you can now enforce vectorization of this loop by using `#pragma omp simd`.

The one big difference between using `#pragma omp simd` and auto-vectorization hints is that with `#pragma omp simd`, the compiler generates a warning when it is unable to vectorize the loop. With auto-vectorization hints, actual vectorization is still under the discretion of the compiler, even when you use the `#pragma vector always` hint.

`#pragma omp simd` has optional clauses to guide the compiler on how vectorization must proceed. Use these clauses appropriately so that the compiler obtains enough information to generate correct vector code. For more information on the clauses, see the `#pragma omp simd` description.

**Additional Semantics**

Note the following points when using the `omp simd` pragma.

- A variable may belong to zero or one of the following: private, linear, or reduction.
- Within the vector loop, an expression is evaluated as a vector value if it is private, linear, reduction, or it has a sub-expression that is evaluated to a vector value. Otherwise, it is evaluated as a scalar value (that is, broadcast the same value to all iterations). Scalar value does not necessarily mean loop invariant, although that is the most frequently seen usage pattern of scalar value.
- A vector value may not be assigned to a scalar L-value. It is an error.
- A scalar L-value may not be assigned under a vector condition. It is an error.
- The `switch` statement is not supported.
NOTE
You may find it difficult to describe vector semantics using the SIMD pragma for some auto-vectorizable loops. One example is \texttt{MIN/MAX} reduction in C since the language does not have \texttt{MIN/MAX} operators.

Restrictions on Using a \texttt{#pragma omp declare simd} declaration

Vectorization depends on two major factors: hardware and the style of source code. When using the vector declaration, the following features are not allowed:

- Thread creation and joining through OpenMP* \texttt{parallel/for/sections/task/target/teams}, and explicit threading API calls.
- Locks, barriers, atomic construct, critical sections (These are allowed inside \texttt{#pragma omp ordered simd} blocks).
- Inline ASM code, VM and Vector Intrinsics (for example, SVML intrinsics).
- Using \texttt{setjmp}, \texttt{longjmp}, SHE and computed \texttt{GOTO}.
- EH is not allowed and all vector functions are considered noexcept.
- The \texttt{switch} statement (in some cases this may be supported and converted to \texttt{if} statements, but this is not reliable).
- The \texttt{exit()/abort()} calls.

Non-vector function calls are generally allowed within vector functions but calls to such functions are serialized lane-by-lane and so might perform poorly. Also for SIMD-enabled functions it is not allowed to have side effects except writes by their arguments. This rule can be violated by non-vector function calls, so be careful executing such calls in SIMD-enabled functions.

Formal parameters must be of the following data types:

- (un)signed 8, 16, 32, or 64-bit integer
- 32- or 64-bit floating point
- 64- or 128-bit complex
- A pointer (C++ reference is considered a pointer data type)

See Also
Function Annotations and the SIMD Directive for Vectorization

\texttt{omp simd pragma} is described in the OpenMP* spec at www.openmp.org.

SIMD-Enabled Functions

SIMD-enabled functions (formerly called elemental functions) are a general language construct to express a data parallel algorithm. A SIMD-enabled function is written as a regular C/C++ function, and the algorithm within describes the operation on one element, using scalar syntax. The function can then be called as a regular C/C++ function to operate on a single element or it can be called in a data parallel context to operate on many elements.

How SIMD-Enabled Functions Work

When you write a SIMD-enabled function, the compiler generates short vector variants of the function that you requested, which can perform your function's operation on multiple arguments in a single invocation. The short vector variant may be able to perform multiple operations as fast as the regular implementation performs a single one by utilizing the vector instruction set architecture (ISA) in the CPU. When a call to a SIMD-enabled function occurs in a SIMD loop or another SIMD-enabled function, the compiler replaces the scalar call with the best fit from the available short-vector variants of the function.
In addition, when invoked from a `pragma omp` construct, the compiler may assign different copies of the SIMD-enabled functions to different threads (or workers), executing them concurrently. The end result is that your data parallel operation executes on the CPU utilizing both the parallelism available in the multiple cores and the parallelism available in the vector ISA. In other words, if the short vector function is called inside a parallel loop, an auto-parallelized loop that is vectorized, you can achieve both vector-level and thread-level parallelism.

**Declaring a SIMD-Enabled Function**

In order for the compiler to generate the short vector function, you need to use the appropriate syntax from below in your code:

**Windows***:

Use the `__declspec(vector (clauses))` declaration, as follows:

```c
__declspec(vector (clauses)) return_type simd_enabled_function_name(parameters)
```

**Linux***:

Use the `__attribute__((vector (clauses)))` declaration, as follows:

```c
__attribute__((vector (clauses))) return_type simd_enabled_function_name(parameters)
```

Alternately, you can use the following OpenMP* pragma, which requires the `[q or Q]openmp` or `[q or Q]openmp-simd` compiler option:

```c
#pragma omp declare simd clauses
```

The clauses in the vector declaration may be used for achieving better performance by overriding defaults. These clauses at SIMD-enabled function definition declare one or several short vector variants for a SIMD-enabled functions. Multiple vector declarations with different set of clauses may be attached to one function in order to declare multiple different short vector variants available for a SIMD-enabled function.

The clauses are defined as follows:

**processor(cpuid)**

Tells the compiler to generate a vector variant using the instructions, the caller/callee interface, and the default vectorlength selection scheme suitable to the specified processor. Use of this clause is highly recommended, especially for processors with wider vector register support (i.e., `core_2nd_gen_avx` and newer).

`cpuid` takes one of the following values:

- `core_4th_gen_avx_tsx`
- `core_4th_gen_avx`
- `core_3rd_gen_avx`
- `core_2nd_gen_avx`
- `core_aes_pclmulqdq`
- `core_i7_sse4_2`
- `atom`
- `core_2_duo_sse4_1`
- `core_2_duo_ssse3`
- `pentium_4_sse3`
- `pentium_m`
- `pentium_4`
- `haswell`
- `broadwell`
- `skylake`
- skylake_avx512
- knl
- knm

\[ \text{vectorlength}(n) / \text{simdlen}(n) \]
(for omp declare simd)

Where \( n \) is a vector length that is a power of 2, no greater than 32.

The \text{simdlen} clause tells the compiler that each routine invocation at
the call site should execute the computation equivalent to \( n \) times the
scalar function execution. When omitted the compiler selects the
vector length automatically depending on the routine return value,
parameters, and/or the processor clause. When multiple vector
variants are called from one vectorization context (for example, two
different functions called from the same vector loop), explicit use of
identical \text{simdlen} values are advised to achieve good performance.

\[ \text{linear(list_item[,}
\text{list_item...])} \]
where \( \text{list_item} \) is one of:
  \( \text{param[:step]} \),
  \( \text{val(param[:step])} \),
  \( \text{ref(param[:step])} \), or
  \( \text{uval(param[:step])} \)

The \text{linear} clause tells the compiler that for each consecutive
invocation of the routine in a serial execution, the value of \( \text{param} \) is
incremented by \( \text{step} \), where \( \text{param} \) is a formal parameter of the
specified function or the C++ keyword this. The \text{linear} clause can
be used on parameters that are either scalar (non-arrays and of non-
structured types), pointers, or C++ references. \( \text{step} \) is a compile-
time integer constant expression, which defaults to 1 if omitted.

If more than one step is specified for a particular parameter, a
compile-time error occurs.

Multiple \text{linear} clauses will be merged as a union.

The meaning of each variant of the clause is as follows:

\[ \text{linear(param[:step]}\) \]
For parameters that are not C++ references: the clause tells the compiler that on each
iteration of the loop from which the routine is
called the value of the parameter will be
incremented by \( \text{step} \). The clause can also be
used for C++ references for backward
compatibility, but it is not recommended.

\[ \text{linear(val(param[:step])}) \]
For parameters that are C++ references: the clause tells the compiler that on each iteration
of the loop from which the routine is called the
referenced value of the parameter will be
incremented by \( \text{step} \).

\[ \text{linear(uval(param[:step])}) \]
For C++ references: means the same as
\text{linear(val())}. It differs from \text{linear(val())} so that
in case of \text{linear(val())} a vector of references is
passed to vector variant of the routine but in
case of \text{linear(uval())} only one reference is
passed (and thus \text{linear(uval())} is better to use
in terms of performance).

\[ \text{linear(ref(param[:step])}) \]
For C++ references: means that the reference
itself is linear, i.e. the referenced values (that
form a vector for calculations) are located
sequentially, like in array with the distance
between elements equal to \( \text{step} \).
uniform(param [, param,]...) Where param is a formal parameter of the specified function or the C++ keyword this.

The uniform clause tells the compiler that the values of the specified arguments can be broadcast to all iterations as a performance optimization. It is often useful in generating more favorable vector memory references. On the other hand, lack of uniform clause may allow broadcast operations to be hoisted out of the caller loop. Evaluate carefully the performance implications. Multiple uniform clauses are merged as a union.

mask / nomask

The mask and nomask clauses tell the compiler to generate only masked or unmasked (respectively) vector variants of the routine. When omitted, both masked and unmasked variants are generated. The masked variant is used when the routine is called conditionally.

inbranch / notinbranch

The inbranch and notinbranch clauses are used with #pragma omp declare simd. The inbranch clause works the same as the mask clause above and the notinbranch clause works the same as the nomask clause above.

Write the code inside your function using existing C/C++ syntax and relevant built-in functions (see the section on __intel_simd_lane() below).

Usage of Vector Function Specifications

You may define several vector variants for one routine with each variant reflecting a possible usage of the routine. Encountering a call, the compiler matches vector variants with actual parameter kinds and chooses the best match. Matching is done by priorities. In other words, if an actual parameter is the loop invariant and the uniform clause was specified for the corresponding formal parameter, then the variant with the uniform clause has a higher priority. Linear specifications have the following order, from high priority to low: linear(uval()), linear(), linear(val()), linear(ref()). Consider the following example loops with the calls to the same routine.

Example: OpenMP*

// routine prototype
#pragma omp declare simd // universal but slowest definition matches the use in all three loops
#pragma omp declare simd linear(in1) linear(ref(in2)) uniform(mul) // matches the use in the first loop
#pragma omp declare simd linear(ref(in2)) // matches the use in the second and the third loops
#pragma omp declare simd linear(ref(in2)) linear(mul) // matches the use in the second loop
#pragma omp declare simd linear(val(in2:2)) // matches the use in the third loop
extern int func(int* in1, int& in2, int mul);

int *a, *b, mul, *c;
int *ndx, nn;
...

// loop examples
for (int i = 0; i < nn; i++) {
    c[i] = func(a + i, *(b + i), mul); // in the loop, the first parameter is changed linearly,
Example: OpenMP*

```c
} for (int i = 0; i < nn; i++) {     // the second reference is changed linearly too
c[i] = func(&a[ndx[i]], b[i], i + 1);  // the value of the first parameter is
}                                              // the third parameter is not changed

} for (int i = 0; i < nn; i++) {
  c[i] = func(&a[ndx[i]], b[i], i + 1);  // the value of the first parameter is
}                                              // the second reference is changed linearly

#pragma omp simd
for (int i = 0; i < nn; i++) {                 // the third parameter is changed linearly
  k->k_vec[i] = func(&a[ndx[i]], b[i]);      // the value of the first parameter is unpredictable,
}                                              // the second reference and value can be considered

```

**SIMD-Enabled Functions and C++**

You should use SIMD-enabled functions in modern C++ with caution: C++ imposes strict requirements on compilation and execution environments which may not compose well with semantically-rich language extensions such as SIMD-enabled functions. There are three key aspects of C++ that interrelate with SIMD-enabled functions concept: exception handling, dynamic polymorphism, and the C++ type system.

**SIMD-Enabled Functions and Exception Handling**

Exceptions are currently not supported in SIMD contexts: exceptions cannot be thrown and/or caught in SIMD loops and SIMD-enabled functions. Therefore, all SIMD-enabled functions are considered `noexcept` in C++11 terms. This affects not only short vector variants of a function, but its original scalar routine as well. This is enforced when the function is compiled: it is checked against throw construct and against function calls throwing exceptions. It is also enforced when the SIMD-enabled function call is compiled.

**SIMD-Enabled Functions and Dynamic Polymorphism**

Vector attributes can be applied to virtual functions of classes with some limitations and taken into account during polymorphic virtual function calls. The syntax of vector declarations is the same as for regular SIMD-enabled class methods: just attach vector declarations as described above to the method declarations inside the class declaration.

Vector function attributes for virtual methods are inherited. If a vector attribute is specified for an overriding virtual function, it must match that of the overridden function. Even if the virtual method implementation is overridden in a derived class the vector declarations are inherited and applied. A set of vector variants is produced for the override according to vector variants set on parent. This rule also applies when the parent does not have any vector variants. If some virtual method is introduced as non-SIMD-enabled (no vector declarations supplied) it cannot become SIMD-enabled in the derived class even if the derived class contains its own implementation of the virtual method.
Matching vector variants for a virtual methods is done by the declared (static) type of an object for which the method is called. The actual (dynamic) type of an object may either coincide with the static type or be inherited from it.

Unlike regular function calls which transfer control to one target function, the call target of a virtual function depends on the dynamic type of the object for which the method is called and accomplished indirectly via the virtual function table of a class. In a single SIMD chunk, the virtual method may be invoked for objects of multiple classes, for example, elements of a polymorphic collection. This requires multiple calls to different targets within a single SIMD chunk. This works as follows:

1. If a SIMD-enabled virtual function call is matched to a variant with a uniform *this* parameter, multiple calls are not needed. The compiler makes an indirect call to the matched vector variant of a virtual method of the object's dynamic class.

2. If a SIMD-enabled virtual function call is matched to a variant with a non-uniform *this* parameter, all objects in a SIMD chunk may still share the same virtual method implementation. This is checked and a single, indirect call to the matched vector variant of the target virtual method implementation is invoked.

3. Otherwise, lanes sharing virtual call targets are masked-in and a masked vector variant corresponding to the match is invoked in a loop for each unique virtual call target. If a masked variant is not provided for matching a vector variant and a *this* parameter is not declared uniform, the match will be rejected.

The following example illustrates SIMD-enabled virtual functions:

```
Example: OpenMP*

struct Base {
    #pragma omp declare simd
    #pragma omp declare simd uniform(this)
    virtual int process(int);
};

struct Child1 : Base {
    // int process(int); is inherited
};

struct Child11 : Child1 {
    int process(int); // Overrides implementation, inherits vector declarations
};

struct Child2 : Base {
    int process(int); // Overrides implementation, inherits vector declarations
};

int main() {
    int arr[100];
    Base* c2 = new Child2();
    Base* objs[100];
    int res = 0;

    // SIMD-enabled virtual function call for uniform object
    #pragma omp simd reduction(+:res)
    for (int i = 0; i < 100; i++) {
        res += c2->process(arr[i]); // Variant with uniform this is matched
            // call to vector variant of
            // Child2::process() is invoked
    }

    // Initialize polymorphic array of objects
```
The following are limitations to SIMD-enabled virtual function support:

- **Multiple inheritance**, including virtual inheritance, is not supported for classes having SIMD-enabled virtual methods. This is because calls to virtual functions in multiple inheritance cases may be done through special functions called thunks which adjust the 'this' pointer and/or virtual function table pointer. The current implementation doesn't support thunks for SIMD-enabled virtual calls because in this case thunks should themselves become SIMD-enabled functions which is not implemented.
- **It is not possible to get the address of a SIMD-enabled virtual method.** Support of SIMD-enabled virtual functions would require additional information, so their binary representation is different. Such cases will not be handled properly by code expecting a regular pointer to the virtual member.

**SIMD-Enabled Functions and the C++ Type System**

Vector attributes are attributes in the C++11 sense and so are not part of a functional type of SIMD-Enabled functions. Vector attributes are bound to the function itself, an instance of a functional type. This has the following implications:

- **Template instantiations having SIMD-enabled functions as template parameters won't catch vector attributes**, so it is currently impossible to reliably preserve vector attributes in function wrapper templates like `std::bind` which add indirection. This indirection may sometimes be optimized away by compiler and the resulting direct call will have all vector attributes associated.
- **There is no way to overload or specialize templates by vector attributes.**
- **There is no way to write functional traits to capture vector attributes for the sake of template metaprogramming.**

The example below depicts various situations where this situation may be observed:
Example: OpenMP*

```cpp
template <int f(int)>  // Function value template - captures exact function  // not a function type
    int caller1(int x[100]) {  
        int res = 0;  
        #pragma omp simd reduction(+:res)  
        for (int i = 0; i < 100; i++) {  
            res += f(x[i]);  // Exact function put here upon instantiation  
        }  
        return res;  
    }

template <typename F>  // Generic functional type template - captures  // object type for functors or entire functional type  // for functions. If vector attributes were part of  // a functional type they might be captured and applied  // but currently they are not.
    int caller2(F f, int x[100]) {  
        int res = 0;  
        #pragma omp simd reduction(+:res)  
        for (int i = 0; i < 100; i++) {  
            res += f(x[i]);  // Will call matching function 'f' indirectly  
                             // Will call matching f.operator() directly  
        }  
        return res;  
    }

template <typename RET, typename ARG>  // Type-decomposing template  // captures argument and return types.  // Vector attributes would be lost  // even if they were part of a  // functional type.
    int caller3(RET (*f)(ARG), int x[100]) {  
        int res = 0;  
        #pragma omp simd reduction(+:res)  
        for (int i = 0; i < 100; i++) {  
            res += f(x[i]);  // Will call matching function 'f' indirectly  
        }  
        return res;  
    }

#pragma omp declare simd
int function(int x);  // SIMD-enabled function
int nv_function(int x);  // Regular scalar function
struct functor {  
    #pragma omp declare simd  
    // SIMD-enabled operator()  
    int operator()(int x);  
};

int arr[100];

int main() {  
    int res;  
    #pragma noinline
```
Example: OpenMP

```cpp
res = caller1<function>(arr); // This will be instantiated for
   // function() and call short vector variant
#pragma noinline
res += caller1<nv_function>(arr); // This will be separately instantiated
   // for nv_function()
#pragma noinline
res += caller2(function, arr); // This will be instantiated for
   // int(*)(int) type and will call scalar
   // function() indirectly
#pragma noinline
res += caller2(nv_function, arr); // This will call the same
   // instantiation as above on nv_function
#pragma noinline
res += caller2(functor(), arr); // This will be instantiated for
   // functor type and will call short vector
   // variant of functor::operator()
#pragma noinline
res += caller3(function, arr); // This will be instantiated for
   // <int, int> types and will call scalar
   // function() indirectly
#pragma noinline
res += caller3(nv_function, arr); // This will call the same
   // instantiation as above on nv_function
return res;
```

**NOTE** If calls to `caller1`, `caller2` and `caller3` are inlined, the compiler is able to replace indirect calls by direct calls in all cases. In this case `caller2(function, arr)` and `caller3(function, arr)` both call short vector variants of a function as result of the usual replacement of direct calls to `function()` by matching short vector variants in the SIMD loop.

**Invoking a SIMD-Enabled Function with Parallel Context**

Typically, the invocation of a SIMD-enabled function provides arrays wherever scalar arguments are specified as formal parameters.

The following two invocations will give instruction-level parallelism by having the compiler issue special vector instructions.

```cpp
a[:]=ef_add(b[:],c[:]); //operates on the whole extent of the arrays a, b, c
a[0:n:s]=ef_add(b[0:n:s],c[0:n:s]); //use the full array notation construct to also specify n as an extend and s as a stride
```

**NOTE** The array notation syntax, as well as calling the SIMD-enabled function from the regular `for` loop, results in invoking the short vector function in each iteration and utilizing the vector parallelism but the invocation is done in a serial loop, without utilizing multiple cores.

Use of array notation syntax and SIMD-enabled functions in a regular `for` loop results in invoking the short vector function in each iteration and utilizing the vector parallelism, but the invocation is done in a serial loop without utilizing multiple cores.
Using the __intel_simd_lane() Built-in Function

When called from within a vectorized loop, the __intel_simd_lane() built-in function will return a number between 0 and vectorlength - 1 that reflects the current "lane id" within the SIMD vector. __intel_simd_lane() will return zero if the loop is not vectorized. Calling __intel_simd_lane() outside of an explicit vector programming construct is discouraged. It may prevent auto-vectorization and such a call often results in the function returning zero instead of a value between 0 and vectorlength-1.

To see how __intel_simd_lane() can be used, consider the following example:

```c
void accumulate(float *a, float *b, float *c, float d){
   *a+=sin(d);
   *b+=cos(d);
   *c+=log(d);
}

for (i=low; i<high; i++){
   accumulate(&suma, &sumb, &sumc, d[i]);
}
```

Example: OpenMP*

```c
#define VL 16
#pragma omp declare simd uniform(a,b,c) linear(i)
void accumulate(float *a, float *b, float *c, float d, i){
   a[i & (VL-1)]+=sin(d);
   b[i & (VL-1)]+=cos(d);
   c[i & (VL-1)]+=log(d);
}

float a[VL] = {0.0f};
float b[VL] = {0.0f};
float c[VL] = {0.0f};
#pragma omp simd for simdlen(VL)
for (i=low; i<high; i++){
   accumulate(a, b, c, d[i], i);
}
for(i=0;i<VL;i++){
   suma += a[i];
   sumb += b[i];
   sumc += c[i];
}
```

The gather-scatter type memory addressing caused by the references to arrays A, B, and C in the SIMD-enabled function accumulate() will significantly hurt performance making the whole conversion useless. To avoid this penalty you may use the __intel_simd_lane() built-in function as follows:

Example: OpenMP*

```c
#pragma omp declare simd uniform(a,b,c) aligned(a,b,c)
void accumulate(float *a, float *b, float *c, float d){
   // No need to take "loop index". No need to know VL.
   a[__intel_simd_lane()]+=sin(d);
   b[__intel_simd_lane()]+=cos(d);
   c[__intel_simd_lane()]+=log(d);
}
```

#define VL 16 // actual SIMD code may use vectorlength of 4 but it's okay.
Example: OpenMP*

```c
float a[VL] = {0.0f};
float b[VL] = {0.0f};
float c[VL] = {0.0f};
#pragma omp simd for simdlen(VL)
for (i=low; i<high; i++) {
    // If low is known to be zero at compile time, “i & (VL-1)”
    // would accomplish what __intel_simd_lane() is intended for,
    // but only on the caller side.
    accumulate(a, b, c, d[i]);
}
for (i=0; i<VL; i++) {
    suma += a[i];
    sumb += b[i];
    sumc += c[i];
}
```

With use of __intel_simd_lane() the references to the arrays in accumulate() will have unit-stride.

Limitations

The following language constructs are not allowed within SIMD-enabled functions:

- The GOTO statement.
- The switch statement with 16 or more case statements.
- Operations on classes and structs (other than member selection).
- Any OpenMP* construct.

See Also

User-Mandated or SIMD Vectorization
Function Annotations and the SIMD Directive for Vectorization
SIMD-Enabled Function Pointers

SIMD-Enabled Function Pointers

SIMD-enabled functions (formerly called elemental functions) are a general language construct to express a data parallel algorithm. A SIMD-enabled function is written as a regular C/C++ function, and the algorithm within describes the operation on one element, using scalar syntax. The function can then be called as a regular C/C++ function to operate on a single element or it can be called in a data parallel context to operate on many elements.

In some cases it is desirable to have a pointer for SIMD-enabled functions, but without special effort, the vector nature of a function will be lost: function pointers will point to the scalar function and there will be no way to call the short vector variants existing for this scalar function.

In order to support indirect calls to vector variants of SIMD-enabled functions, SIMD-enabled function pointers were introduced. A SIMD-enabled function pointer is a special kind of pointer incompatible with a regular function pointer. They refer to an entire set of short vector variants as well as the scalar function. This incompatibility incurs the risk of inappropriate misuse, especially in C++ code. Therefore vector function pointer support is disabled by default.

How SIMD-Enabled Function Pointers Work

When you write a SIMD-enabled function, the compiler generates short vector variants of the function that you requested, which can perform your function’s operation on multiple arguments in a single invocation. The short vector variants may be able to perform multiple operations as fast as the regular implementation performs just one such operation by utilizing the vector instruction set architecture (ISA) in the CPU. When a call to SIMD-enabled function occurs in a SIMD loop or another SIMD-enabled function, the compiler replaces the scalar call with the best fit short vector variant of the function among those available.
Indirect SIMD-enabled function calls are handled similarly, but the set of available variants should be associated with the function pointer variable, not the target function, because actual call targets are unknown at the indirect call. That means all SIMD-enabled functions to be referenced by a SIMD-enabled function pointer should have a set of variants that match the set of variants declared for the pointer.

**Declaring a SIMD-Enabled Function Pointer Variable**

In order for the compiler to generate a pointer to a SIMD-enabled function, you need to provide an indication in your code.

Windows*:

Use the `__declspec(vector (clauses))` attribute, as follows:

```
__declspec(vector (clauses)) return_type (*function_pointer_name) (parameters)
```

Linux:

Use the `__attribute__((vector (clauses)))` attribute, as follows:

```
__attribute__((vector (clauses))) return_type (*function_pointer_name) (parameters)
```

Alternately, you can use OpenMP* #pragma omp declare simd, which requires the 
[q or Q]openmp or [q or Q]openmp-simd compiler option.

The clauses are described in the previous topic on SIMD-enabled functions.

**Usage of Vector Function Attributes on Pointers**

You may associate several vector attributes with one SIMD-enabled function pointer which reflects all the variants available for the target functions to be called through the pointer. The attributes usually reflect a possible use of the function pointer in the loops. Encountering an indirect call, the compiler matches the vector variants declared on the function pointer with the actual parameter kinds and chooses the best match. Matching is done exactly the same way as with direct calls (see the previous topic on SIMD-enabled functions). Consider the following example of the declaration of vector function pointers and loops with indirect calls.

**Example: OpenMP**

```
// pointer declaration
#pragma omp declare simd                           // universal but slowest definition matches
the use in all three loops
#pragma omp declare simd linear(in1) linear(ref(in2)) uniform(mul)   // matches the use in the
first loop
#pragma omp declare simd linear(ref(in2))                                      // matches the use in the
second and the third loops
#pragma omp declare simd linear(ref(in2)) linear(mul)                         // matches the use in the
second loop
#pragma omp declare simd linear(val(in2:2))                                    // matches the use in the
third loop
int (*func)(int* in1, int& in2, int mul);
int *a, *b, mul, *c;
int *ndx, nn;
...
// loop examples
for (int i = 0; i < nn; i++) {
    c[i] = func(a + i, *(b + i), mul); // in the loop, the first parameter is changed
    linearly,
    // the second reference is changed linearly too
```

731
Example: OpenMP*

```c
for (int i = 0; i < nn; i++) {
    c[i] = func(&a[ndx[i]], b[i], i + 1); // the value of the first parameter is unpredictable,
    // the second reference is changed linearly
    // the third parameter is changed linearly
}
```

```c
#pragma omp simd
for (int i = 0; i < nn; i++) {
    int k = i * 2;  // during vectorization, private variables are transformed into arrays:
    k->k_vec[vector_length]
    c[i] = func(&a[ndx[i]], k, b[i]); // the value of the first parameter is unpredictable,
    // the second reference and value can be considered linear
    // the third parameter has unpredictable value
    // (the __declspec(vector(linear(val(in2:2)))) will be chosen from the two matching variants)
}
```

Before any use in a call, the function pointer should be assigned either the address of a function or another function pointer. Just as with function pointers, vector function pointers should be compatible at assignment and initialization. The compatibility rules are described below.

**Vector Function Pointer Compatibility**

Pointer assignment compatibility is defined as following:

1. If a SIMD-enabled function pointer is assigned the address of a function, the function should be compatible with the pointer in the usual C/C++ sense, it should be SIMD-enabled, and the set of vector variants declared for the function should be a superset of those declared for the pointer. This includes initializations and passing addresses of SIMD-enabled functions as parameters.

2. If a SIMD-enabled function pointer is assigned another function pointer, the source pointer should be compatible with the destination function pointer in the general C/C++ sense, it should be SIMD-enabled, and the set of vector variants declared for the source pointer should be exactly the same as those declared for destination pointer. This includes initializations and passing SIMD-enabled function pointers as parameters.

3. If a regular (non-SIMD-enabled) function pointer is assigned the address of a SIMD-enabled function, the address of a scalar function is assigned. Vector variants cannot be called through the pointer and it cannot be reinterpreted as or converted into a SIMD-enabled function pointer as discussed in rule 2.

4. If a regular (non-SIMD-enabled) function pointer is assigned a SIMD-enabled function pointer matching in the C/C++ sense, the implicit dynamic casting of the right-hand side of the assignment (RHS) is performed by extracting the address of a scalar function and this address is assigned. Vector variants cannot be called through these pointers and it cannot be reinterpreted as or converted into a SIMD-enabled function pointer as discussed in rule 2.
**NOTE** SIMD-enabled function pointers and regular function pointers are binary-incompatible and handled differently. Mixing them may lead to severe unpredictable results. The compiler does its best to check compatibility where it is allowed by C/C++ language standards, but in certain cases it cannot check, such as passing function pointers to undeclared functions or as variable arguments. It is best to refrain from using SIMD-enabled function pointers in these contexts. Additional complexities with respect to the C++ type system are described in the *SIMD-Enabled Function Pointers and the C++ Type System* section below.

**NOTE** A SIMD-enabled function pointer may be assigned to a scalar function pointer with a cast as described in rule 4 above, but a SIMD-enabled function pointer cannot refer to a scalar function pointer.

### Examples of Declarations and Assignments: OpenMP*

```c
// pointer declarations
#pragma omp declare simd
int (*ptr1)(int*, int);
#pragma omp declare simd
int (*ptr1a)(int*, int);

#pragma omp declare simd
#pragma omp declare simd linear(a)
typedef int (*fptr_t2)(int* a, int b);

typedef int (*fptr_t3)(int*, int);
fptr_t2 ptr2, ptr2a;
fptr_t3 ptr3;

// function declarations
#pragma omp declare simd
int func1(int* x, int b);

#pragma omp declare simd
#pragma omp declare simd linear(x)
int func2(int* x, int b);

#pragma omp declare simd
#pragma omp declare simd linear(x)
int func3(float* x, int b);

// allowed assignments
// allowed assignments
ptr1 = func1;  // same prototype and vector spec
ptr2 = func2;  // same prototype and vector spec
ptr1a = ptr1;  // same prototype and vector spec
ptr1a = func2; // same prototype vector spec on function includes all vector spec on pointer

ptr3 = func1; // scalar pointer with same prototype - use scalar func1
ptr3 = func2; // scalar pointer with same prototype - use scalar func2
ptr3 = ptr1;  // scalar pointer with same prototype - implicit conversion from vector to scalar pointer
ptr3 = ptr2;  // scalar pointer with same prototype - implicit conversion from vector to scalar pointer
```
Examples of Declarations and Assignments: OpenMP*

// disallowed assignments
ptr2 = func1; // vector spec on function does not have all specs on pointer
ptr2 = func3; // prototype mismatch although vector spec matched
ptr1 = func3; // prototype mismatch although vector spec matched
ptr3 = func3; // prototype mismatch
ptr1 = ptr2;  // pointers should have the same vector spec
ptr2 = ptr3;  // pointers should have the same vector spec

Call Sequence

Unlike regular function calls, which transfer control to a target function, the call target of an indirect call depends on the dynamic content of the function pointer. In a loop, call targets may be different on different iterations of a vectorized loop or on different lanes of a SIMD-enabled function executing the call. When vectorized, such an indirect call may involve multiple calls to different targets within a single SIMD chunk. This works as follows:

1. If the vector function pointer is uniform (refer to the OpenMP* specification) or if it can be determined to be uniform by the compiler, then multiple calls are not needed. The compiler makes a single indirect call to a matched vector variant accessible by the pointer.
2. If the vector function pointer is not known to be uniform at compile time, all values of the pointer in a SIMD chunk may still be the same. This is checked at run time and a single indirect call to a matched vector variant is invoked.
3. Otherwise, lanes sharing the same function pointer value (call target) are masked-in and a masked vector variant corresponding to the matched one is invoked in the loop for each unique call target. If the masked variant is not provided for the matching vector variant and the function pointer is not proven to be uniform by compiler the match will be rejected and the compiler may serialize the call, or in other words, generate several scalar calls.

Example: OpenMP*

// pointer typedefs
#pragma omp declare simd
typedef int (*fptr_t1)(int*, int);

// function declarations
#pragma omp declare simd
int func1(int* x, int b);

// uses of vector function pointers
fptr_t1 *fptr_array; // array of vector function pointers
void foo(int N, int *x, int y){
    fptr_t1 ptr1 = func1;
    #pragma omp simd
    for (int i = 0; i < N; i++) {
        ptr1(x+i, y); // ptr1 is uniform by OpenMP rule.
        fptr_t1 ptr1a = ptr1;
        ptr1a(x+i, y); // compiler can prove ptr1a is uniform.
        fptr_t1 ptr1b = fptr_array[i];
        ptr1b(x+i,y); // ptr1b may or may not be uniform.
    }
}

734
**SIMD-Enabled Function Pointers and the C++ Type System**

Use caution when using SIMD-enabled function pointers in modern C++: C++ imposes strict requirements on compilation and execution environments which may not compose well with semantically-rich language extensions such as SIMD-enabled function pointers. Vector specifications on SIMD-enabled function pointers are attributes in C++11 sense and so are not part of a pointer type even though they make that pointer binary incompatible with another pointer of the same type but without the attribute. Vector specifications are not bound to a pointer type, but instead are bound to the variable or function argument (which is an instance of a pointer type) itself. For a given function pointer, the type of the pointer is the same with or without SIMD-enabled function pointer decoration. This has the following important implications:

- Vector attributes put on a function argument are not reflected in C++ name mangling, so the functions differ only in the vector attributes of a functional pointer argument (or lack thereof) will have the same name and will be treated the same by the C++ linker. This may result in a parameter of incorrect vectorness (having the vector attribute or not) being passed into the function. In some cases there is no way for the compiler to detect this situation, so you're strongly encouraged to distinctly name functions having SIMD-enabled function pointers as parameters.
- The incorrect interpretation of function pointers is extremely dangerous because it may lead to the execution of unwanted code or non-code. To identify these situations the compiler issues the following warning if a vector function pointer is used as a C++ function parameter: Warning #3757: this use of a vector function type is not fully supported. If you are sure that no ambiguity is possible—for example, the function accepting the vector function pointer has a distinct name and is fully declared before all uses—you may ignore this warning. Otherwise, ensure that no ambiguity is possible.
- Template instantiations having SIMD-enabled pointer types as template parameters won't catch vector attributes. The template will be instantiated a parameter matching the non-SIMD-enabled pointer type. All variables, class members, and function arguments bound to the template argument type will be regular function pointers. The use of such templates with a SIMD-enabled function pointer as a template function parameter, template class method parameter, or RHS of template class member assignment will lead to a dynamic cast to the non-SIMD-enabled function pointer and loss of vectorness.
- There is no way to overload or achieve template specialization by the vector attributes of a functional pointer
- There is no way to write functional traits to capture vector attributes for the sake of template metaprogramming.

**Examples: OpenMP**

```c++
// pointer typedefs and pointer declarations
typedef int
(*fptr_t)(int*, int);

#pragma omp declare simd
typedef int (*fptr_t1)(int*, int);

#pragma omp declare simd
#pragma omp declare simd linear(x)
typedef int (*fptr_t2)(int* a, int b);

fptr_t ptr
fptr_t1 ptr1
fptr_t2 ptr2

// function prototype that only differs in SIMD-enabled function decoration
// All these will have identical mangled names.
void foo(fptr_t);
void foo(fptr_t1);
void foo(fptr_t2);
```
**Examples: OpenMP**

```c++
// template instantiation
template <typename T>
void bar(T);
...
bar(fptr);     // bar<fptr_t>
bar(fptr1);    // bar<fptr_t>
bar(fptr2);    // bar<fptr_t>
```

**Indirect Invocation of a SIMD-Enabled Function with Parallel Context**

Typically, the invocation of a SIMD-enabled function directly or indirectly provides arrays wherever scalar arguments are specified as formal parameters.

The following invocations will give instruction-level parallelism by having the compiler issue special vector instructions.

**Example: OpenMP**

```c++
#pragma omp declare simd
float (**vf_ptr)(float, float);
// operates on the whole extent of the arrays a, b, c
a[:] = vf_ptr[:](b[:],c[:]);
// use the full array notation construct to also specify n
// as an extend and s as a stride
a[0:n:s] = vf_ptr[0:n:s](b[0:n:s],c[0:n:s]);
```

**NOTE** The array notation syntax, as well as calling the SIMD-enabled function from the regular `for` loop, results in invoking the short vector variant in each iteration and utilizing the vector parallelism but the invocation is done in a serial loop, without utilizing multiple cores.

**See Also**
- User-mandated or SIMD Vectorization
- Function Annotations and the SIMD Directive for Vectorization
- SIMD-enabled functions

**Vectorizing a Loop Using the _Simd Keyword**

In this section we introduce the `_Simd` keyword, which provides an alternative to the `simd` pragma. Just like the `simd` pragma, the `_Simd` keyword modifies a serial `for` loop for vectorization. The syntax is as follows:

```c++
_Simd [_Safelen(constant-expression)][_Reduction (reduction-identifier : list)]
```

The `_Simd` keyword and any clauses should come after the `for` keyword as in this example:

```c++
for _Simd (int i=0; i<10; i++){
    // loop body
}
```

Differences between the `simd` pragma and `_Simd` keyword:

- Omission of the `private` and `lastprivate` clauses of the `simd` pragma construct because C and C++ already have variable-scoping rules that allow a programmer to cleanly declare a private variable within the scope of a loop iteration.
• The linear clause is omitted because the ability to increment multiple variables makes it unnecessary. See the following example:

```c
float add_floats(float *a, float *b, int n){
    int i=0;
    int j=0;
    float sum=0;

    for _Simd _Reduction(+;sum) (i=0; i<n; i++, j+=2){
        a[i] = a[i] + b[j];
        sum += a[i];
    }
    return sum;
}
```

To ensure that your loop is vectorized keep the following in mind:

• The countable loop for the _Simd keyword has to conform to the for-loop style of an OpenMP* canonical loop form except that multiple variables may be incremented in the incr-expr (See the OpenMP* specification at www.openmp.org).
• The loop control variable must be a signed integer type.
• The vector values should be signed 8-, 16-, 32-, or 64-bit integers, single or double-precision floating point numbers, or single or double-precision complex numbers.
• You cannot use any control constructs to jump into or out of a SIMD loop. That includes the break, return, goto, and throw constructs.
• A SIMD loop may contain another loop (for, while, do-while) in it, but goto out of such inner loops is not supported. You may use break and continue with the inner loop.
• A SIMD loop performs memory references unconditionally. Therefore, all address computations must result in valid memory addresses, even though such locations may not be accessed if the loop is executed sequentially.

See Also
User-mandated or SIMD Vectorization
simd Enforces vectorization of loops.

Function Annotations and the SIMD Directive for Vectorization

This topic presents specific C++ language features that better help to vectorize code.

**NOTE**
The SIMD vectorization feature is available for both Intel® microprocessors and non-Intel microprocessors. Vectorization may call library routines that can result in additional performance gain on Intel® microprocessors than on non-Intel microprocessors.

The __declspec(align(n)) declaration enables you to overcome hardware alignment constraints. The auto-vectorization hints address the stylistic issues due to lexical scope, data dependency, and ambiguity resolution. The SIMD feature's pragma allows you to enforce vectorization of loops.

You can use the __declspec(vector) __attribute__((vector)) and the __declspec(vector[clauses]) __attribute__((vector(clauses))) declarations to vectorize user-defined functions and loops. For SIMD usage, the vector function is called from a loop that is being vectorized.

The C/C++ extensions for Array Notations map operations can be defined to provide general data parallel semantics, where you do not express the implementation strategy. You can write the same operation regardless of the size of the problem. The implementation uses the construct by combining SIMD, loops and tasking to implement the operation. With these semantics, you can choose more elaborate programming and express a single dimensional operation at two levels. You can use both task constructs and array operations to force a preferred parallel and vector execution.
The usage model of the `vector` declaration takes a small section of code generated for the function (`vectorlength`) of the array and exploits SIMD parallelism. The implementation of task parallelism is done at the call site.

The following table summarizes the language features that help vectorize code.

<table>
<thead>
<tr>
<th>Language Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__declspec(align(n))</code></td>
<td>Directs the compiler to align the variable to an <code>n</code>-byte boundary. Address of the variable is <code>address mod n=0</code>.</td>
</tr>
<tr>
<td><code>__declspec(align(n,off))</code></td>
<td>Directs the compiler to align the variable to an <code>n</code>-byte boundary with offset <code>off</code> within each <code>n</code>-byte boundary. Address of the variable is <code>address mod n=off</code>.</td>
</tr>
<tr>
<td><code>__declspec(vector)</code> (Windows*)</td>
<td>Combines with the <code>map</code> operation at the call site to provide the data parallel semantics. When multiple instances of the vector declaration are invoked in a parallel context, the execution order among them is not sequenced.</td>
</tr>
<tr>
<td><code>__attribute__((vector))</code> (Linux*)</td>
<td>Combines with the <code>map</code> operation at the call site to provide the data parallel semantics. When multiple instances of the vector declaration are invoked in a parallel context, the execution order among them is not sequenced.</td>
</tr>
<tr>
<td><code>restrict</code></td>
<td>Permits the disambiguator flexibility in alias assumptions, which enables more vectorization.</td>
</tr>
<tr>
<td><code>__declspec(vector_variant(clauses))</code> (Windows*)</td>
<td>Provides the ability to vectorize user-defined functions and loops. The <code>clauses</code> are as follows:</td>
</tr>
<tr>
<td><code>__attribute__((vector_variant(clauses)))</code> (Linux*)</td>
<td></td>
</tr>
<tr>
<td><code>__assume_aligned(a,n)</code></td>
<td>Instructs the compiler to assume that array <code>a</code> is aligned on an <code>n</code>-byte boundary; used in cases where the compiler has failed to obtain alignment information.</td>
</tr>
<tr>
<td><code>__assume(cond)</code></td>
<td>Instructs the compiler to assume that the represented condition is true where the keyword appears. Typically used for conveying</td>
</tr>
</tbody>
</table>
Language Feature | Description
--- | ---
properties that the compiler can take advantage of for generating more efficient code, such as alignment information.

Auto-vectorization Hints

```c
#pragma ivdep
```
Instructs the compiler to ignore assumed vector dependencies.

```c
#pragma vector
{aligned|unaligned|always|
temporal|nontemporal}
```
Specifies how to vectorize the loop and indicates that efficiency heuristics should be ignored. Using the `assert` keyword with the `vector {always}` pragma generates an error-level assertion message if the compiler efficiency heuristics indicate that the loop cannot be vectorized. Use `#pragma ivdep!` to ignore the assumed dependencies.

```c
#pragma novector
```
Specifies that the loop should never be vectorized.

### NOTE
Some pragmas are available for both Intel® microprocessors and non-Intel microprocessors, but may perform additional optimizations for Intel® microprocessors than for non-Intel microprocessors.

User-Mandated Pragma

```c
#pragma simd
```
Enforces vectorization of loops.

```c
omp simd
```
Transforms the loop into a loop that will be executed concurrently using SIMD instructions.

### See Also
`__declspec(align)` declaration
`ivdep` pragma
`simd` pragma
`vector` pragma
SIMD-enabled functions
User-mandated or SIMD Vectorization

**Explicit SIMD SYCL® Extension**

*oneAPI provides an Explicit SIMD SYCL extension (ESIMD) for lower-level Intel GPU programming.*

ESIMD provides APIs that are similar to Intel’s GPU Instruction Set Architecture (ISA), but it enables you to write explicitly vectorized device code. This explicit enabling gives you more control over the generated code and allows you to depend less on compiler optimizations.

The specification, documented ESIMD API headers, and working code examples are available on GitHub®.

### NOTE
This extension is under active development and the APIs are subject to change. See the Restrictions section below for details.
ESIMD kernels and functions always require a subgroup size of one, which means that the compiler does not provide vectorization across work items in a subgroup. Instead, you must explicitly express the vectorization in your code. Below is an example that adds the elements of two arrays and writes the results to the third:

```c++
float *A = static_cast<float *>(malloc_shared(Size * sizeof(float), dev, ctxt));
float *B = static_cast<float *>(malloc_shared(Size * sizeof(float), dev, ctxt));
float *C = static_cast<float *>(malloc_shared(Size * sizeof(float), dev, ctxt));
for (unsigned i = 0; i < Size; ++i) {
    A[i] = B[i] = i;
}
// Many work items are needed. Each work item processes Vector Length (VL)
// elements of data.
cl::sycl::range<1> GlobalRange{Size / VL};
// Number of work items in each workgroup.
cl::sycl::range<1> LocalRange{GroupSize};
cl::sycl::nd_range<1> Range(GlobalRange, LocalRange);
auto e = q.submit([&](handler &cgh) {
    cgh.parallel_for<class Test>(Range, [=](nd_item<1> ndi) SYCL_ESIMD_KERNEL {
        using namespace sycl::ext::intel::experimental::esimd;
        int i = ndi.get_global_id(0);
        simd<float, VL> va = block_load<float, VL>(A + i * VL);
        simd<float, VL> vb = block_load<float, VL>(B + i * VL);
        simd<float, VL> vc = va + vb;
        block_store<float, VL>(C + i * VL, vc);
    });
});
```

In the example above, the lambda function passed to the `parallel_for` is marked with a special attribute: `SYCL_ESIMD_KERNEL`. This attribute tells the compiler that the kernel is ESIMD-based and ESIMD APIs can be used inside it. Here the `simd` objects and `block_load/block_store` intrinsics are used. They are available only in the ESIMD extension.

Fully runnable code samples can be found in the GitHub repo.

### Compiling and Running ESIMD Code

Compiling and running code that uses the ESIMD extension is the same as compiler and running code that uses the standard SYCL:

```
$ dpcpp vadd_usm.cpp
```

The resulting executable (`$ ./a.out`) can be run only on Intel GPU hardware, such as Intel® UHD Graphics 600 or later. The DPC++ runtime automatically recognizes ESIMD kernels and dispatches their execution, so no additional setup is needed.

### Restrictions

This section contains lists of the main restrictions that apply when using the ESIMD extension.

**NOTE** Some extensions are not enforced by the compiler, which may lead to undefined program behavior.

- Features not supported with ESIMD:
- Ahead-of-time (AOT) compilation
- C and C++ standard libraries support
- Device library extensions
- A host device (in some cases)
- Unsupported standard SYCL APIs:
  - Local accessors
  - Most image APIs
  - Memory access through a raw pointer returned by `sycl::accessor::get_pointer()`
- Other restrictions:
  - Only Intel GPU devices are supported.

**High-Level Optimization (HLO)**

High-level Optimizations (HLO) exploit the properties of source code constructs (for example, loops and arrays) in applications developed in high-level programming languages. While the default optimization level, option `O2`, performs some high-level optimizations, specifying the `O3` option provides the best chance for performing loop transformations to optimize memory accesses.

**NOTE**

Loop optimizations may result in calls to library routines that can result in additional performance gain on Intel® microprocessors than on non-Intel microprocessors. Additional HLO transformations may be performed for Intel® microprocessors than for non-Intel microprocessors.

Within HLO, loop transformation techniques include:
- Loop Permutation or Interchange
- Loop Distribution
- Loop Fusion
- Loop Unrolling
- Data Prefetching
- Scalar Replacement
- Unroll and Jam
- Loop Blocking or Tiling
- Partial-Sum Optimization
- Predicate Optimization
- Loop Reversal
- Profile-Guided Loop Unrolling
- Loop Peeling
- Data Transformation: Malloc Combining and Memset Combining, Memory Layout Change
- Loop Rerolling
- Memset and Memcpy Recognition
- Statement Sinking for Creating Perfect Loopnests
- Multiversioning: Checks include Dependency of Memory References, and Trip Counts
- Loop Collapsing

**Interprocedural Optimization (IPO)**

Interprocedural Optimization (IPO) is an automatic, multi-step process that allows the compiler to analyze your code to determine where you can benefit from specific optimizations.

The compiler may apply the following optimizations:
• Address-taken analysis
• Array dimension padding
• Alias analysis
• Automatic array transposition
• Automatic memory pool formation
• C++ class hierarchy analysis
• Common block variable coalescing
• Common block splitting
• Constant propagation
• Dead call deletion
• Dead formal argument elimination
• Dead function elimination
• Formal parameter alignment analysis
• Forward substitution
• Indirect call conversion
• Inlining
• Mod/ref analysis
• Partial dead call elimination
• Passing arguments in registers to optimize calls and register usage
• Points-to analysis
• Routine key-attribute propagation
• Specialization
• Stack frame alignment
• Structure splitting and field reordering
• Symbol table data promotion
• Un-referenced variable removal
• Whole program analysis

IPO Compilation Models

IPO supports two compilation models - single-file compilation and multi-file compilation.

The compiler performs some single-file interprocedural optimization at the O2 default optimization level; additionally the compiler may perform some inlining for the O1 optimization level, such as inlining functions marked with inlining pragmas or attributes (GNU C and C++) and C++ class member functions with bodies included in the class declaration.

Multi-file compilation uses the [Q]ipo option, and results in one or more mock object files rather than normal object files. (See the Compilation section below for information about mock object files.) Additionally, the compiler collects information from the individual source files that make up the program. Using this information, the compiler performs optimizations across functions and procedures in different source files.

Compiling with IPO

As each source file is compiled with IPO, the compiler stores an intermediate representation (IR) of the source code in a mock object file. The mock object files contain the IR instead of the normal object code. Mock object files can be ten times or more larger than the size of normal object files.

During the IPO compilation phase only the mock object files are visible.

Linking with IPO

When you link with the [Q]ipo compiler option the compiler is invoked a final time. The compiler performs IPO across all mock object files. The mock objects must be linked with the compiler or by using the Intel® linking tools. While linking with IPO, the compiler and other linking tools compile mock object files as well as invoke the real/true object files linkers provided on the user's platform.
Whole Program Analysis

The compiler supports a large number of IPO optimizations that can be applied or have its effectiveness greatly increased when the whole program condition is satisfied.

During the analysis process, the compiler reads all Intermediate Representation (IR) in the mock file, object files, and library files to determine if all references are resolved and whether or not a given symbol is defined in a mock object file. Symbols that are included in the IR in a mock object file for both data and functions are candidates for manipulation based on the results of whole program analysis.

There are two types of whole program analysis - object reader method and table method. Most optimizations can be applied if either type of whole program analysis determines that the whole program conditions exists; however, some optimizations require the results of the object reader method, and some optimizations require the results of table method.

Object reader method

In the object reader method, the object reader emulates the behavior of the native linker and attempts to resolve the symbols in the application. If all symbols are resolved, the whole program condition is satisfied. This type of whole program analysis is more likely to detect the whole program condition.

Table method

In the table method the compiler analyzes the mock object files and generates a call-graph.

The compiler contains detailed tables about all of the functions for all important language-specific libraries, like libc. In this second method, the compiler constructs a call-graph for the application. The compiler then compares the function table and application call-graph. For each unresolved function in the call-graph, the compiler attempts to resolve the calls by finding an entry for each unresolved function in the compiler tables. If the compiler can resolve the functions call, the whole program condition exists.

See Also

Inline Expansion of Functions

Interprocedural Optimization (IPO) Options

ipo, Qipo

Using IPO

This topic discusses how to use IPO from the command line.

Compiling and Linking Using IPO

To enable IPO, you first compile each source file, then link the resulting source files.

First, compile your source files with [Q]ipo compiler as shown below:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux*</td>
<td>icpx -ipo -c a.cpp b.cpp c.cpp</td>
</tr>
<tr>
<td>Windows*</td>
<td>icx /Qipo /c a.cpp b.cpp c.cpp</td>
</tr>
</tbody>
</table>

The output of the above example command differs according to operating system:
- Linux: The commands produce a.o, b.o, and c.o object files.
- Windows: The commands produce a.obj, b.obj, and c.obj object files.

Use the c compiler option to stop compilation after generating .o or .obj files. The output files contain compiler intermediate representation (IR) corresponding to the compiled source files.

Second, link the resulting files. The following example command will produce an executable named app:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icpx -o app a.o b.o c.o</td>
</tr>
<tr>
<td>Windows</td>
<td>icx /Feapp a.obj b.obj c.obj</td>
</tr>
</tbody>
</table>

The command invokes the compiler on the objects containing IR and creates a new list of objects to be linked. Alternately, you can use the xild (Linux) or xilink (Windows) tool, with the appropriate linking options.

Combining the Steps

The separate compile and link commands demonstrated above can be combined into a single command, as shown in the following examples:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>icpx -ipo -o app a.cpp b.cpp c.cpp</td>
</tr>
<tr>
<td>Windows</td>
<td>icx /Qipo /Feapp a.cpp b.cpp c.cpp</td>
</tr>
</tbody>
</table>

The icx/icpx (for C++) or dpcpp (for DPC++) command, shown in the examples above, calls GCC ld (Linux) or link.exe (Windows only) to link the specified object files and produce the executable application, which is specified by the -o (Linux) or /Fe (Windows) option.

**NOTE**

Linux: Using icpx (for C++) or dpcpp (for DPC++) allows the compiler to use standard C++ libraries automatically; icx will not use the standard C++ libraries automatically.

The Intel linking tools emulate the behavior of compiling at -O0 (Linux) and /Od (Windows) option.

If multiple file IPO is applied to a series of object files, no one which are mock object files, no multi-file IPO is performed. The object files are simply linked with the linker.

**See Also**

c  compiler option

o  compiler option

Fe compiler option

ipo, Qipo  compiler option

O  compiler option

**IPO-Related Performance Issues**

There are some general optimization guidelines for using IPO that you should keep in mind:
Using IPO on very large programs might trigger internal limits of other compiler optimization phases. Applications where the compiler does not have sufficient intermediate representation (IR) coverage to do whole program analysis might not perform as well as those where IR information is complete.

In addition to these general guidelines, there are some practices to avoid while using IPO. The following list summarizes the activities to avoid:

- Do not use the link phase of an IPO compilation using mock object files produced for your application by a different compiler. Intel® compilers cannot inspect mock object files generated by other compilers for optimization opportunities.
- Update make files to call the appropriate Intel linkers when using IPO from scripts. For Linux, replace all instances of `ld` with `xild`; for Windows, replace all instances of `link` with `xilink`.

See Also

IPO for Large Programs

IPO for Large Programs

In most cases, IPO generates a single true object file for the link-time compilation. This behavior is not optimal for very large programs, perhaps even making it impossible to use `[Q]ipo` compiler option on the application.

The compiler provides two methods to avoid this problem. The first method is an automatic size-based heuristic, which causes the compiler to generate multiple true object files for large link-time compilations. The second method is to manually instruct the compiler to perform multi-object IPO.

- Use the `[Q]ipoN` compiler option and pass an integer value in the place of `N`.

Regardless of the method used, it is best to use the compiler defaults first and examine the results. If the defaults do not provide the desired results then experiment with generating a different number of object files.

Using `[Q]ipoN` to Create Multiple Object Files

If you specify `[Q]ipo0`, which is the same as not specifying a value, the compiler uses heuristics to determine whether to create one or more object files based on the expected size of the application. The compiler generates one object file for small applications, and two or more object files for large applications. If you specify any value greater than 0, the compiler generates that number of object files, unless the value you pass a value that exceeds the number of source files. In that case, the compiler creates one object file for each source file then stops generating object files.

The following example commands demonstrate how to use `[Q]ipo2` option to compile large programs.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Example Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows*</td>
<td>dpcpp-cl /Qipo2 /c a.cpp b.cpp</td>
</tr>
<tr>
<td>Linux*</td>
<td>dpcpp -ipo2 -c a.cpp b.cpp</td>
</tr>
</tbody>
</table>

In executing the above commands, the compiler generates object files using an OS-dependent naming convention. On Linux*, the example command results in object files named `ipo_out.o`, `ipo_out1.o`, and `ipo_out2.o`. On Windows*, the file names follow the same convention; however, the file extensions will be `.obj`.

Link the resulting object files as shown in Using IPO.
See Also
ipo, Qipo
compiler option

Understanding Code Layout and Multi-Object IPO

One of the optimizations performed during an IPO compilation is code layout. The analysis performed by the compiler during multi-file IPO determines a layout order for all of the routines for which it has intermediate representation (IR) information. For a multi-object IPO compilation, the compiler must tell the linker about the desired order.

The compiler first puts each routine in a named text section that varies depending on the operating system:

Windows:
• The first routine is placed in .text$00001, the second is placed in .text$00002, and so on.

Linux:
• The first routine is placed in .text00001, the second is placed in .text00002, and so on.

Creating a Library from IPO Objects

Linux*

Libraries are often created using a library manager such as xiar for Linux* or xilib for Windows*. Given a list of objects, the library manager will insert the objects into a named library to be used in subsequent link steps.

Example

```bash
xiar cru user.a a.o b.o
```

The above command creates a library named user.a containing the a.o and b.o objects.

Using xiar is the same as specifying xild -lib.

Windows Only

Create libraries using xilib or xilink -lib to create libraries of IPO mock object files and link them on the command line.

For example, assume that you create three mock object files by using a command similar to the following:

Example

```bash
[invocation] /c /Qipo a.cpp b.cpp c.cpp
```

Where [invocation] is icx for C++ or dpcpp-cl for DPC++.

Further assume a.obj contains the main subprogram. You can enter commands similar to the following to create a library.

Example

```bash
xilib -out:main.lib b.obj c.obj
// or
xilink -lib -out:main.lib b.obj c.obj
```

You can link the library and the main program object file by entering a command similar to the following:
### Inline Expansion of Functions

Inline function expansion does not require that the applications meet the criteria for whole program analysis normally required by IPO; so this optimization is one of the most important optimizations done in Interprocedural Optimization (IPO). For function calls that the compiler believes are frequently executed, the compiler often decides to replace the instructions of the call with code for the function itself.

In the compiler, inline function expansion is performed on relatively small user functions more often than on functions that are relatively large. This optimization improves application performance by performing the following:

- Removing the need to set up parameters for a function call
- Eliminating the function call branch
- Propagating constants

Function inlining can improve execution time by removing the runtime overhead of function calls; however, function inlining can increase code size, code complexity, and compile times. In general, when you instruct the compiler to perform function inlining, the compiler can examine the source code in a much larger context, and the compiler can find more opportunities to apply optimizations.

Specifying the `ipo` compiler option, multi-file IPO, causes the compiler to perform inline function expansion for calls to procedures defined in other files.

**Caution**

Using the `ipo` (Windows*) options can, in some cases, significantly increase compile time and code size.

The compiler does a certain amount of inlining at the default level.

### Selecting Routines for Inlining

The compiler attempts to select the routines whose inline expansions provide the greatest benefit to program performance. The selection is done using default heuristics.

When you use PGO with `ipo`, the compiler uses the following guidelines for applying heuristics:

- The default heuristic focuses on the most frequently executed call sites, based on the profile information gathered for the program.
- The default heuristic always inlines very small functions that meet the minimum inline criteria.

**Using IPO with PGO**

Combining IPO and PGO typically produces better results than using IPO alone. PGO produces dynamic profiling information that can usually provide better optimization opportunities than the static profiling information used in IPO.

The compiler uses characteristics of the source code to estimate which function calls are executed most frequently. It applies these estimates to the PGO-based guidelines described above. The estimation of frequency, based on static characteristics of the source, is not always accurate.

---

**Example**

```
xilink -out:result.exe a.obj main.lib
```
Inline Expansion of Library Functions

By default, the compiler automatically inlines (expands) a number of standard and math library functions at the point of the call to that function, which usually results in faster computation.

Many routines in the libirc, libm, or the svml library are more highly optimized for Intel microprocessors than for non-Intel microprocessors.

The -fno-builtin (Linux*) or the /Qno-builtin-<name> and /Oi- (Windows*) options disable inlining for intrinsic functions and disable the by-name recognition support of intrinsic functions and the resulting optimizations. The /Qno-builtin-<name> option provides the ability to disable inlining for intrinsic functions, fine-tuning the functionality of the /Oi- option, which disables almost all intrinsic functions when used. Use these options if you redefine standard library routines with your own version and your version of the routine has the same name as the standard library routine.

Inlining and Function Preemption (Linux)

You must specify fpic to use function preemption. By default the compiler does not generate the position-independent code needed for preemption.

See Also
fbuiltin, Oi
fpic
ipo, Qipo

Compiler Directed Inline Expansion of Functions

Without directions from the user, the compiler attempts to estimate what functions should be inlined to optimize application performance. See Inline Expansion of Functions for more information.

The following options are useful in situations where an application can benefit from user function inlining but does not need specific direction about inlining limits.

<table>
<thead>
<tr>
<th>Option</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>fno-builtin (Linux*) or Oi- (Windows)</td>
<td>Disables inlining for intrinsic functions. Disables the by-name recognition support of intrinsic functions and the resulting optimizations. Use this option if you redefine standard library routines with your own version and your version of the routine has the same name as the standard library routine. By default, the compiler automatically inlines (expands) a number of standard and math library functions at the point of the call to that function, which usually results in faster computation. Many routines in the libirc, libm, or svml library are more highly optimized for Intel microprocessors than for non-Intel microprocessors. Indicates that the source position information for an inlined function should be retained, rather than replaced, by that of the call which is being inlined.</td>
</tr>
<tr>
<td>setting inline-debug-info for the debug option</td>
<td>Indicates that the source position information for an inlined function should be retained, rather than replaced, by that of the call which is being inlined.</td>
</tr>
</tbody>
</table>

See Also
depth (Linux*)
depth (Windows*)
Developed Directed Inline Expansion of User Functions

In addition to the options that support compiler directed inline expansion of user functions, the compiler also provides compiler options and pragmas that allow you to more precisely direct when and if inline function expansion should occur.

The compiler measures the relative size of a routine in an abstract value of intermediate language units, which is approximately equivalent to the number of instructions that will be generated. The compiler uses the intermediate language unit estimates to classify routines and functions as relatively small, medium, or large functions. The compiler then uses the estimates to determine when to inline a function; if the minimum criteria for inlining is met and all other things are equal, the compiler has an affinity for inlining relatively small functions and not inlining relative large functions.

Typically, the compiler targets functions that have been marked for inlining based on the following:

- **Inlining keywords:** Tells the compiler to inline the specified function. For example, `__inline`, `__forceinline`.
- **Procedure-specific inlining pragmas:** Tells the compiler to inline calls within the targeted procedure if it is legal to do so. For example, `#pragma inline` or `#pragma forceinline`.
- **GCC function attributes for inlining:** Tells the compiler to inline the function even when no optimization level is specified. For example, `__attribute__((always_inline))`.

If your code hits an inlining limit, the compiler issues a warning at the highest warning level. The warning specifies which of the inlining limits have been hit, and the compiler option and/or pragmas needed to get a full report.

Messages in the report refer directly to the command line options or pragmas that can be used to overcome the limits.

**Methods to Optimize Code Size**

This section provides some guidance on how to achieve smaller object and smaller executable size when using the optimizing features of Intel compilers.

To begin, there are two compiler options that are designed to prioritize code size over performance:

<table>
<thead>
<tr>
<th>Favors size over speed</th>
<th>Linux*: -Os</th>
<th>Windows*: /Os</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This option enables optimizations that do not increase code size; it produces smaller code size than option O2.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Option Os disables some optimizations that may increase code size for a small speed benefit.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Minimizes code size</th>
<th>Linux*: -O1</th>
<th>Windows*: /O1</th>
</tr>
</thead>
<tbody>
<tr>
<td>This option enables optimizations that do not increase code size; it produces smaller code size than option O2.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compared to option Os, option O1 disables even more optimizations that are generally known to increase code size. Specifying option O1 implies option Os.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>As an intermediate step in reducing code size, you can replace option O3 with option O2 before specifying option O1.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Option O1 may improve performance for applications with very large code size, many branches, and execution time not dominated by code within loops.

For more information about the above options, see their full descriptions in the Compiler Reference.

The rest of this section briefly discusses methods that may help you further improve code size even when compared to the default behaviors of options Os and O1.

The following table shows the methods for code size optimization that are summarized in this section.

| Disable or Decrease the Amount of Inlining |
| Strip Symbols from Your Binaries |
| Dynamically Link Intel-Provided Libraries |
| Exclude Unused Code and Data from the Executable |
| Disable Recognition and Expansion of Intrinsic Functions |
| Optimize Exception Handling Data on Linux Systems |
| Disable Loop Unrolling |
| Disable Automatic Vectorization |
| Avoid References to Compiler-Specific Libraries |
| Use Inter-Procedural Optimization (IPO) |

For more information about compiler options that are mentioned in the above topics, see their full descriptions in the Compiler Reference.

Things to remember:

- Some of these methods may already be applied by default when options Os and O1 are specified. All the methods mentioned in subsequent topics can be applied at higher optimization levels.
- Some of the options referred to in these topics will not necessarily cause code size reduction, and they may provide varying results (good, bad, or neutral) based on the characteristics of the target code. Still, these are the recommended things to try to see if they cause your binaries to become smaller while maintaining acceptable performance.

**See Also**

○ compiler option

Os compiler option

---

**Disable or Decrease the Amount of Inlining**

Inlining replaces a call to a function with the body of the function. This lets the compiler optimize the code for the inlined function in the context of its caller, usually yielding more specialized and better performing code. This also removes the overhead of calling the function at run-time.

However, replacing a call to a function by the code for that function usually increases code size. The code size increase can be substantial. To eliminate this code size increase, at the cost of the potential performance improvement, inlining can be disabled.

As an alternative to completely disabling inlining, the default amount of inlining can be decreased by using an inline factor less than the default value of 100. It corresponds to scaling the default values of the main inlining parameters by \( n \% \).

**Options to specify:**
To disable inlining:

Linux*: -fno-inline
Windows*: /Ob0

For further details about the compiler options, see the compiler option descriptions.

Advantages of this method:
Disabling or reducing this optimization can reduce code size.

Disadvantages of this method:
Performance is likely to be sacrificed by disabling or reducing inlining especially for applications with many small functions.

**Strip Symbols from Your Binaries**

You can specify a compiler option to omit debugging and symbol information from the executable without sacrificing its operability.

**Options to specify:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux*:</td>
<td>-Wl, --strip-all</td>
</tr>
<tr>
<td>Windows*:</td>
<td>None</td>
</tr>
</tbody>
</table>

Advantages of this method:
This method noticeably reduces the size of the binary.

Disadvantages of this method:
It may be very difficult to debug a stripped application.

**Dynamically Link Intel-Provided Libraries**

This content is specific to C++; it does not apply to DPC++.

By default, some of the Intel support and performance libraries are linked statically into an executable. As a result, the library codes are linked into every executable being built. This means that codes are duplicated.

It may be more profitable to link them dynamically.

**Options to specify:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux*:</td>
<td>-shared-intel</td>
</tr>
<tr>
<td>Windows*:</td>
<td>/MD</td>
</tr>
</tbody>
</table>

**NOTE** Option /MD affects all libraries, not only the Intel-provided ones.

Advantages of this method:
- Performance of the resulting executable is normally not significantly affected.
- Library codes that are otherwise linked in statically into every executable will not contribute to the code size of each executable with this option. These codes will be shared between all executables using them, and will be available independent of those executables.

Disadvantages of this method:
- The libraries on which the resulting executable depends must be re-distributed with the executable in order for it to work properly.
When libraries are linked statically, only library content that is actually used is linked into the executable. Dynamic libraries, on the other hand, contain all the library content. Therefore, it may not be beneficial to use this option if you only need to build and/or distribute a single executable.

The executable itself may be much smaller when linked dynamically, compared to a statically linked executable. However, the total size of the executable plus shared libraries or DLLs may be much larger than the size of the statically linked executable.

### Exclude Unused Code and Data from the Executable

Programs often contain dead code or data that is not used during their execution. Even if no expensive whole-program inter-procedural analysis is made at compile time to identify dead code, there are compiler options you can specify to eliminate unused functions and data at link time.

This method is often referred to as function-level linking.

#### Options to specify:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux*</td>
<td>-fdata-sections -ffunction-sections -Wl,--gc-sections</td>
</tr>
<tr>
<td>Windows*</td>
<td>/Gy /link /OPT:REF</td>
</tr>
</tbody>
</table>

Some of the options in the above specifications are passed to the linker.

#### Advantages of this method:

Only the code that is referenced remains in an executable. Dead functions and data are stripped from the executable.

#### Disadvantages of this method:

- The object codes may become slightly larger because each function or datum is put into a separate section. The overhead is eliminated at the linking stage.
- This method requires linker support to strip unused sections.
- This method can slightly increase linking time.

### Disable Recognition and Expansion of Intrinsic Functions

This content is specific to C++; it does not apply to DPC++.

When recognized, intrinsic functions can get expanded inline or their faster implementation in a library may be assumed and linked in. By default, Inline expansion of intrinsic functions is enabled.

In some cases, disabling this behavior may noticeably improve the size of the produced object or binary.

#### Options to specify:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux*</td>
<td>-fno-builtin</td>
</tr>
<tr>
<td>Windows*</td>
<td>/Oi-</td>
</tr>
</tbody>
</table>

#### Advantages of this method:

Both the size of the object files and the size of library codes brought into an executable can be reduced.
Disadvantages of this method:

- This method can prevent various performance optimizations from happening. Slower standard library implementation will be used.
- The size of the final executable can be increased in cases when code pulled in statically from a library for an otherwise inlined intrinsic is large.

Notes:

- This option is already the default if you specify option O1.
- You can specify option -nolib-inline to disable inline expansion of standard library or intrinsic functions.
- Depending on code characteristics, this option can sometimes increase binary size.

Optimize Exception Handling Data (Linux*)

This content is specific to C++; it does not apply to DPC++.

If a program requires support for exception handling, the compiler creates a special section containing DWARF directives that are used by the Linux* run-time to unwind and catch an exception.

This information is located in the .eh_frame section and may be shrunk using the compiler options listed below.

Options to specify:

<table>
<thead>
<tr>
<th></th>
<th>Linux*</th>
<th>Windows*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-fno-exceptions</td>
<td>-fno-asynchronous-unwind-tables</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Advantages of this method:

- These options may shrink the size of the object or binary file by up to 15%, though the amount of the reduction depends on the target platform.
- These options control whether unwind information is precise at an instruction boundary or at a call boundary. For example, option -fno-asynchronous-unwind-tables can be used for programs that may only throw or catch exceptions.

Disadvantages of this method:

Both options may change the program's behavior:

- Do not use option -fno-exceptions for programs that require standard C++ handling for objects of classes with destructors.
- Do not use option -fno-asynchronous-unwind-tables for functions compiled with option -fexceptions that contain calls to other functions that might throw exceptions or for C++ functions that declare objects with destructors.

Please read the compiler option descriptions, which explain what the defaults and behavior are for each target platform.

Disable Loop Unrolling

Unrolling a loop increases the size of the loop proportionally to the unroll factor.

Disabling (or limiting) this optimization may help reduce code size at the expense of performance.

Options to specify:
Disable Automatic Vectorization

This content is specific to C++; it does not apply to DPC++.

The compiler finds possibilities to use SIMD (SSE/AVX) instructions to improve performance of applications. This optimization is called automatic vectorization.

In most cases, this optimization involves transformation of loops and increases code size, in some cases significantly.

Disabling this optimization may help reduce code size at the expense of performance.

Options to specify:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux*</td>
<td>-no-vec</td>
</tr>
<tr>
<td>Windows* (C++ only)</td>
<td>/Qvec-</td>
</tr>
</tbody>
</table>

Advantages of this method:
Compile-time is also improved significantly.

Disadvantages of this method:
Performance of otherwise vectorized loops may suffer significantly. If you care about the performance of your application, you should use this option selectively to suppress vectorization on everything except performance-critical parts.

Notes:
Depending on code characteristics, this option can sometimes increase binary size.

Avoid References to Compiler-Specific Libraries

While compiler-specific libraries are intended to improve the performance of your application, they increase the size of your binaries.

Certain compiler options may improve the code size.

Options to specify:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux*</td>
<td>-ffreestanding</td>
</tr>
<tr>
<td>Windows* (C++ only)</td>
<td>/Qfreestanding</td>
</tr>
</tbody>
</table>

Advantages of this method:
The compiler will not assume the presence of compiler-specific libraries. It will generate only calls that appear in the source code.

Disadvantages of this method:
This method may sacrifice performance if the library codes were in hotspots. Also, because we cannot assume any libraries, some compiler optimizations will be suppressed.

Notes:
• This option implies option \texttt{-fno-builtin}; you can override that default by explicitly specifying option \texttt{-fbuiltin}.
• Depending on code characteristics, this option can sometimes increase binary size.

**Use Inter-Procedural Optimization (IPO)**

Using IPO may reduce code size because it enables dead code elimination and suppresses generation of code for functions always inlined or proven never to be called during execution.

**Options to specify:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux*:</td>
<td>-ipo</td>
</tr>
<tr>
<td>Windows*:</td>
<td>/Qipo</td>
</tr>
</tbody>
</table>

| Advantages of this method: | Depending on the code characteristics, this optimization can reduce executable size and improve performance. |
| Disadvantages of this method: | Binary size can increase depending on code/application. |

This method is not recommended if you plan to ship object files as part of a final product.

**Intel® Math Library**

The Intel® oneAPI DPC++/C++ Compiler includes a mathematical software library containing highly optimized and very accurate mathematical functions. These functions are commonly used in scientific or graphic applications, as well as other programs that rely heavily on floating-point computations. To include support for C99 \_Complex data types, use the \[Q\]std=c99 compiler option.

Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The mathimf.h header file includes prototypes for Intel® Math Library functions.

**NOTE**

Intel's math.h header file is compatible with the GCC* Math Library libm, but it does not cause the GCC Math Library to be linked. The source can be built with gcc or icx. Intel Math Library is linked when you build with icx. The header file for the Intel Math Library mathimf.h contains additional functions that are found only in the Intel Math Library. The source can only be built using the Intel oneAPI DPC++/C++ Compiler and libraries.

The long double functions, such as \texttt{expl} or \texttt{logl}, in the Intel Math Library are ABI incompatible with the Microsoft* libraries. The Intel compiler and libraries support the 80-bit long double data type (see the description of the Qlong-double option). For maximum compatibility, use math.h or mathimf.h header files along with the Intel Math Library.

**Intel Math Libraries for Linux***

The math library linked to an application depends on the compilation or linkage options specified.

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libimf.a</td>
<td>Default static math library.</td>
</tr>
<tr>
<td>libimf.so</td>
<td>Default shared math library.</td>
</tr>
</tbody>
</table>
The Intel Compiler Math Libraries contain performance-optimized implementations for various Intel platforms. By default, the best implementation for the underlying hardware is selected at runtime. The library dispatch of multi-threaded code may lead to apparent data races, which may be detected by certain software analysis tools. However, as long as the threads are running on cores with the same CPUID, these data races are harmless and are not a cause for concern.

**Intel Math Libraries for Windows**

The math library linked to an application depends on the compilation or linkage options specified.

<table>
<thead>
<tr>
<th>Library</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libm.lib</td>
<td></td>
<td>Default static math library.</td>
</tr>
<tr>
<td>libmmt.lib</td>
<td>/MT</td>
<td>Multi-threaded static math library.</td>
</tr>
<tr>
<td>libmmd.lib</td>
<td>/MD</td>
<td>Dynamically linked math library.</td>
</tr>
<tr>
<td>libmmd.lib</td>
<td>/MDd</td>
<td>Dynamically linked debug math library.</td>
</tr>
<tr>
<td>libmmd.lib</td>
<td></td>
<td>Static version compiled with /MD option.</td>
</tr>
</tbody>
</table>

**oneAPI and OpenCL™ Considerations**

Currently, oneAPI uses the OpenCL Specification to determine the ULP accuracy for OpenCL mathematical functions. Details about their precision and accuracy, including tables for single and double precision functions, are available from the Khronos* OpenCL Specification’s section, Relative Error as ULPs.

Mathematical functions have different accuracy levels on different devices. The OpenCL specification sets a limit on the maximum ULP error (where applicable), but individual devices may provide a more accurate implementation. If the OpenCL implementation is optimized for CPU usage, using the same code may not work on a GPU device.

**See Also**

Function List
Qlong-double  compiler option
MD  compiler option
MT  compiler option
std, Qstd  compiler option
Using the Intel® Math Library

**Using the Intel® Math Library**

Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The mathimf.h header file includes prototypes for Intel® Math Library functions.

To use the Intel Math Library, include the header file, mathimf.h, in your program. If the Intel® oneAPI DPC++/C++ Compiler is used for linking, then the Intel Math Library is used by default.

**Example: Using Real Functions**

The following examples demonstrate how to use the Intel Math Library with the Intel oneAPI DPC++/C++ Compiler. After you compile this example and run the program, the program will display the sine value of $x$. 
// real_math.c
#include <stdio.h>
#include <mathimf.h>

int main() {
    float fp32bits;
    double fp64bits;
    long double fp80bits;
    long double pi_by_four = 3.141592653589793238/4.0;

    // pi/4 radians is about 45 degrees
    fp32bits = (float) pi_by_four; // float approximation to pi/4
    fp64bits = (double) pi_by_four; // double approximation to pi/4
    fp80bits = pi_by_four; // long double (extended) approximation to pi/4

    // The sin(pi/4) is known to be 1/sqrt(2) or approximately .7071067
    printf("When x = %8.8f, sinf(x) = %8.8f \n", fp32bits, sinf(fp32bits));
    printf("When x = %16.16f, sin(x) = %16.16f \n", fp64bits, sin(fp64bits));
    printf("When x = %20.20Lf, sinl(x) = %20.20Lf \n", fp80bits, sinl(fp80bits));
    return 0;
}

Use the following command to compile the example code on Linux platforms:

[invocation] real_math.c

The invocation is icx for C, icpx for C++, or dpcpp for DPC++.

Windows*

// real_math.c
#include <stdio.h>
#include <mathimf.h>

int main() {
    float fp32bits;
    double fp64bits;

    // /Qlong-double compiler option required because, without it,
    // long double types are mapped to doubles.
    long double fp80bits;
    long double pi_by_four = 3.141592653589793238/4.0;

    // pi/4 radians is about 45 degrees
    fp32bits = (float) pi_by_four;

    // float approximation to pi/4
    fp64bits = (double) pi_by_four;

    // double approximation to pi/4
    fp80bits = pi_by_four;

    // long double (extended) approximation to pi/4
    // The sin(pi/4) is known to be 1/sqrt(2) or approximately .7071067
    printf("When x = %8.8f, sinf(x) = %8.8f \n", fp32bits, sinf(fp32bits));
    printf("When x = %16.16f, sin(x) = %16.16f \n", fp64bits, sin(fp64bits));
    printf("When x = %20.20Lf, sinl(x) = %20.20Lf \n", fp80bits, sinl(fp80bits));

    return 0;
}
fp64bits, sin(fp64bits));
printf("When x = %20.20f, sinl(x) = %20.20f \n",
(double) fp80bits, (double) sinl(fp80bits));

// printf() does not support the printing of long doubles
// on Microsoft* Windows*, so fp80bits is cast to double in this example.
return 0;
}

This content is specific to C++; it does not apply to DPC++.

Since the real_math.c program includes the long double data type, use the /Qlong-double and /Qpc80 compiler options in the command line:

[invocation] /Qlong-double /Qpc80 real_math.c

The invocation is icx for C/C++ or dpcpp-cl for DPC++.

Example Using Complex Functions

After you compile this example and run the program, you should get the following results:

When z = 1.0000000 + 0.7853982 i, cexpf(z) = 1.9221154 + 1.9221156 i
When z = 1.0000000000000 + 0.785398163397 i, cexp(z) = 1.922115514080 + 1.922115514080 i

Linux and Windows

// complex_math.c
#include <stdio.h>
#include <complex.h>

int main() {
    float _Complex c32in,c32out;
    double _Complex c64in,c64out;
    double pi_by_four= 3.141592653589793238/4.0;
    c64in = 1.0 + I* pi_by_four;

    // Create the double precision complex number 1 + (pi/4) * i
    // where I is the imaginary unit.
    c32in = (float _Complex) c64in;

    // Create the float complex value from the double complex value.
    c64out = cexp(c64in);
    c32out = cexpf(c32in);

    // Call the complex exponential,
    // cexpf(z) = cexp(x+iy) = e^ (x + i y) = e^x * (cos(y) + i sin(y))
    printf("When z = %7.7f + %7.7f i, cexpf(z) = %7.7f + %7.7f i \n",
        crealf(c32in),cimagf(c32in),crealf(c32out),cimagf(c32out));
    printf("When z = %12.12f + %12.12f i, cexp(z) = %12.12f + %12.12f i \n",
        creal(c64in),cimag(c64in),creal(c64out),cimagf(c64out));

    return 0;
}

Since this example program includes the _Complex data type, be sure to include the [Q]std=c99 compiler option in the command line.
To compile this example code in Linux use the following command:

```
[invocation] -std=c99 complex_math.c
```

The `invocation` is `icx` for C, `icpx` for C++, or `dpcpp` for DPC++.

To compile this example code in Windows, use the following command:

```
[invocation] /Qstd=c99 complex_math.c
```

The `invocation` is `icx` for C/C++ or `dpcpp-cl` for DPC++.

**NOTE** Complex data types are supported in C but not in C++ programs.

### Exception Conditions

If you call a math function using argument(s) that may produce undefined results, an error number is assigned to the system variable `errno`. Math function errors are usually domain errors or range errors.

**Domain errors** result from arguments that are outside the domain of the function. For example, `acos` is defined only for arguments between -1 and +1 inclusive. Attempting to evaluate `acos(-2)` or `acos(3)` results in a domain error, where the return value is `QNaN`.

**Range errors** occur when a mathematically valid argument results in a function value that exceeds the range of representable values for the floating-point data type. Attempting to evaluate `exp(1000)` results in a range error, where the return value is `INF`.

When domain or range error occurs, the following values are assigned to `errno`:

- **domain error (EDOM)**: `errno = 33`
- **range error (ERANGE)**: `errno = 34`

The following example shows how to read the `errno` value for an EDOM and ERANGE error.

```c
// errno.c
#include <errno.h>
#include <mathimf.h>
#include <stdio.h>

int main(void) {
  double neg_one=-1.0;
  double zero=0.0;

  // The natural log of a negative number is considered a domain error - EDOM
  printf("log(%e) = %e and errno(EDOM) = %d \n",neg_one,log(neg_one),errno);

  // The natural log of zero is considered a range error - ERANGE
  printf("log(%e) = %e and errno(ERANGE) = %d \n",zero,log(zero),errno);
}
```

The output of `errno.c` will look like this:

```
log(-1.000000e+00) = nan and errno(EDOM) = 33
log(0.000000e+00) = -inf and errno(ERANGE) = 34
```

For the math functions in this section, a corresponding value for `errno` is listed when applicable.

### Other Considerations
Some math functions are inlined automatically by the compiler. The functions actually inlined may vary and may depend on any vectorization or processor-specific compilation options used. You can disable automatic inline expansion of all functions by compiling your program with the -fno-builtin option (Linux) or the /Oi- option (Windows).

It is strongly recommended to use the default rounding mode (round-to-nearest-even) when calling math library transcendental functions and compiling with default optimization or higher. Faster implementations—in terms of latency and/or throughput—of these functions are validated under the default round-to-nearest-even mode. Using other rounding modes may make results generated by these faster implementations less accurate, or set unexpected floating-point status flags. This behavior may be avoided by using the -fp-model strict option (Linux) or /fp: strict option (Windows). This option warns the compiler not to assume default settings for the floating-point environment.

**NOTE** 64-bit decimal transcendental functions rely on binary double extended precision arithmetic. To obtain accurate results, user applications that call 64-bit decimal transcendentals should ensure that the x87 unit is operating in 80-bit precision (64-bit binary significands). In an environment where the default x87 precision is not 80 bits, such as Windows, it can be set to 80 bits by compiling the application source files with the /Qpc80 option.

A change of the default precision control or rounding mode may affect the results returned by some of the mathematical functions.

The following are important compiler options when using certain data types in IA-32 (for C++ only) and Intel® 64 architectures running Windows operating systems:

- /Qlong-double: Use this option when compiling programs that require support for the long double data type (80-bit floating-point). Without this option, compilation will be successful, but long double data types will be mapped to double data types.
- /Qstd=c99: Use this option when compiling programs that require support for _Complex data types.

See Also

fbuiltin, Oi compiler option
Overview: Tuning Performance
Qlong-double compiler option
std, Qstd compiler option

Math Functions

This section provides information about math functions.

Function List

Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The mathimf.h header file includes prototypes for Intel® Math Library functions.

The Intel® Math Library functions are listed here by function type.

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigonometric Functions</td>
<td>acos</td>
</tr>
<tr>
<td></td>
<td>acosd</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>acospi</td>
</tr>
<tr>
<td></td>
<td>asin</td>
</tr>
<tr>
<td></td>
<td>asind</td>
</tr>
<tr>
<td></td>
<td>asinpi</td>
</tr>
<tr>
<td></td>
<td>atan</td>
</tr>
<tr>
<td></td>
<td>atan2</td>
</tr>
<tr>
<td></td>
<td>atan2pi</td>
</tr>
<tr>
<td></td>
<td>atand</td>
</tr>
<tr>
<td></td>
<td>atand2</td>
</tr>
<tr>
<td></td>
<td>atanpi</td>
</tr>
<tr>
<td></td>
<td>cos</td>
</tr>
<tr>
<td></td>
<td>cosd</td>
</tr>
<tr>
<td></td>
<td>cospi</td>
</tr>
<tr>
<td></td>
<td>cot</td>
</tr>
<tr>
<td></td>
<td>cotd</td>
</tr>
<tr>
<td></td>
<td>sin</td>
</tr>
<tr>
<td></td>
<td>sincos</td>
</tr>
<tr>
<td></td>
<td>sincosd</td>
</tr>
<tr>
<td></td>
<td>sind</td>
</tr>
<tr>
<td></td>
<td>sinpi</td>
</tr>
<tr>
<td></td>
<td>tan</td>
</tr>
<tr>
<td></td>
<td>tand</td>
</tr>
<tr>
<td></td>
<td>tanpi</td>
</tr>
<tr>
<td>Hyperbolic Functions</td>
<td>acosh</td>
</tr>
<tr>
<td></td>
<td>asinh</td>
</tr>
<tr>
<td></td>
<td>atanh</td>
</tr>
<tr>
<td></td>
<td>cosh</td>
</tr>
<tr>
<td></td>
<td>sinh</td>
</tr>
<tr>
<td></td>
<td>sinhchosh</td>
</tr>
<tr>
<td></td>
<td>tanh</td>
</tr>
<tr>
<td>Exponential Functions</td>
<td>cbrt</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>exp</td>
</tr>
<tr>
<td></td>
<td>exp10</td>
</tr>
<tr>
<td></td>
<td>exp2</td>
</tr>
<tr>
<td></td>
<td>expml</td>
</tr>
<tr>
<td></td>
<td>frexp</td>
</tr>
<tr>
<td></td>
<td>hypot</td>
</tr>
<tr>
<td></td>
<td>invsqrt</td>
</tr>
<tr>
<td></td>
<td>ilogb</td>
</tr>
<tr>
<td></td>
<td>ldexp</td>
</tr>
<tr>
<td></td>
<td>log</td>
</tr>
<tr>
<td></td>
<td>log10</td>
</tr>
<tr>
<td></td>
<td>log1p</td>
</tr>
<tr>
<td></td>
<td>log2</td>
</tr>
<tr>
<td></td>
<td>logb</td>
</tr>
<tr>
<td></td>
<td>pow</td>
</tr>
<tr>
<td></td>
<td>pow2o3</td>
</tr>
<tr>
<td></td>
<td>pow3o2</td>
</tr>
<tr>
<td></td>
<td>powr</td>
</tr>
<tr>
<td></td>
<td>scalb</td>
</tr>
<tr>
<td></td>
<td>scalbln</td>
</tr>
<tr>
<td></td>
<td>scalbn</td>
</tr>
<tr>
<td></td>
<td>sqrt</td>
</tr>
<tr>
<td></td>
<td>annuity</td>
</tr>
<tr>
<td></td>
<td>compound</td>
</tr>
<tr>
<td></td>
<td>erf</td>
</tr>
<tr>
<td></td>
<td>erfcx</td>
</tr>
<tr>
<td></td>
<td>erfc</td>
</tr>
<tr>
<td></td>
<td>erfinv</td>
</tr>
<tr>
<td></td>
<td>gamma</td>
</tr>
<tr>
<td></td>
<td>gamma_r</td>
</tr>
<tr>
<td></td>
<td>j0</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>j1</td>
</tr>
<tr>
<td></td>
<td>jn</td>
</tr>
<tr>
<td></td>
<td>lgamma</td>
</tr>
<tr>
<td></td>
<td>lgamma_r</td>
</tr>
<tr>
<td></td>
<td>tgamma</td>
</tr>
<tr>
<td></td>
<td>y0</td>
</tr>
<tr>
<td></td>
<td>y1</td>
</tr>
<tr>
<td></td>
<td>yn</td>
</tr>
<tr>
<td>Nearest Integer Functions</td>
<td>ceil</td>
</tr>
<tr>
<td></td>
<td>floor</td>
</tr>
<tr>
<td></td>
<td>llrint</td>
</tr>
<tr>
<td></td>
<td>llround</td>
</tr>
<tr>
<td></td>
<td>lrint</td>
</tr>
<tr>
<td></td>
<td>lround</td>
</tr>
<tr>
<td></td>
<td>modf</td>
</tr>
<tr>
<td></td>
<td>nearbyint</td>
</tr>
<tr>
<td></td>
<td>rint</td>
</tr>
<tr>
<td></td>
<td>round</td>
</tr>
<tr>
<td></td>
<td>trunc</td>
</tr>
<tr>
<td>Remainder Functions</td>
<td>fmod</td>
</tr>
<tr>
<td></td>
<td>remainder</td>
</tr>
<tr>
<td></td>
<td>remquo</td>
</tr>
<tr>
<td>Miscellaneous Functions</td>
<td>copysign</td>
</tr>
<tr>
<td></td>
<td>fabs</td>
</tr>
<tr>
<td></td>
<td>fdim</td>
</tr>
<tr>
<td></td>
<td>finite</td>
</tr>
<tr>
<td></td>
<td>fma</td>
</tr>
<tr>
<td></td>
<td>fmax</td>
</tr>
<tr>
<td></td>
<td>fmin</td>
</tr>
<tr>
<td></td>
<td>fpclassify</td>
</tr>
<tr>
<td></td>
<td>isfinite</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td></td>
<td>isgreater</td>
</tr>
<tr>
<td></td>
<td>isgreaterequal</td>
</tr>
<tr>
<td></td>
<td>isninf</td>
</tr>
<tr>
<td></td>
<td>isless</td>
</tr>
<tr>
<td></td>
<td>islessequal</td>
</tr>
<tr>
<td></td>
<td>islessgreater</td>
</tr>
<tr>
<td></td>
<td>isnan</td>
</tr>
<tr>
<td></td>
<td>isnormal</td>
</tr>
<tr>
<td></td>
<td>isnunordered</td>
</tr>
<tr>
<td></td>
<td>maxmag</td>
</tr>
<tr>
<td></td>
<td>minmag</td>
</tr>
<tr>
<td></td>
<td>nextafter</td>
</tr>
<tr>
<td></td>
<td>nexttoward</td>
</tr>
<tr>
<td></td>
<td>signbit</td>
</tr>
<tr>
<td></td>
<td>significand</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>cabs</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>cacos</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>cacosh</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>carg</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>casin</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>casinh</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>catan</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>catanh</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>ccos</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>cexp</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>cexp2</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>cimag</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>cis</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>clog</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>clog10</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>conj</td>
</tr>
</tbody>
</table>
Trigonometric Functions

Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The mathimf.h header file includes prototypes for Intel® Math Library functions.

The Intel® Math Library supports the following trigonometric functions:

acos

Description: The acos function returns the principal value of the inverse cosine of x in the range [0, pi] radians for x in the interval [-1,1].

errno: EDOM, for |x| > 1

Calling interface:

double acos(double x);
long double acosl(long double x);
float acosf(float x);

acosd

Description: The acosd function returns the principal value of the inverse cosine of x in the range [0,180] degrees for x in the interval [-1,1].

errno: EDOM, for |x| > 1

Calling interface:

double acosd(double x);
long double acosl(long double x);
float acosdf(float x);

acospi

Description: The acospi function returns the principal value of the inverse cosine of x, divided by pi, in the range [0,1] for x in the interval [-1,1].

errno: EDOM, for |x| > 1

Calling interface:
double acospi(double x);
float acospif(float x);

**asin**

**Description:** The `asin` function returns the principal value of the inverse sine of `x` in the range \([-\pi/2, +\pi/2]\) radians for `x` in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\)

**Calling interface:**

double asin(double x);
long double asinl(long double x);
float asinf(float x);

**asinpi**

**Description:** The `asinpi` function returns the principal value of the inverse sine of `x`, divided by pi, in the range \([-1/2,1/2]\) degrees for `x` in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\) divided by pi

**Calling interface:**

double asinpi(double x);
float asinpif(float x);

**asind**

**Description:** The `asind` function returns the principal value of the inverse sine of `x` in the range \([-90,90]\) degrees for `x` in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\)

**Calling interface:**

double asind(double x);
long double asindl(long double x);
float asindf(float x);

**atan**

**Description:** The `atan` function returns the principal value of the inverse tangent of `x` in the range \([-\pi/2, +\pi/2]\) radians.

**Calling interface:**

double atan(double x);
long double atanl(long double x);
float atanf(float x);

**atan2**

**Description:** The `atan2` function returns the principal value of the inverse tangent of `y/x` in the range \([-\pi, +\pi]\) radians.

**errno:** EDOM, for `x = 0` and `y = 0`

**Calling interface:**

double atan2(double y, double x);
long double atan2l(long double y, long double x);
float atan2f(float y, float x);
atan2pi

Description: The atan2pi function returns the principal value of the inverse tangent of \( y/x \), divided by pi, in the range \([-1, +1]\).

errno: EDOM, for \( x = 0 \) and \( y = 0 \)

Calling interface:
double atan2pi(double y, double x);
float atan2pif(float y, float x);

atan
d

Description: The atand function returns the principal value of the inverse tangent of \( x \) in the range \([-90,90]\) degrees.

Calling interface:
double atand(double x);
long double atandl(long double x);
float atandf(float x);

atan2d

Description: The atan2d function returns the principal value of the inverse tangent of \( y/x \) in the range \([-180, +180]\) degrees.

errno: EDOM, for \( x = 0 \) and \( y = 0 \).

Calling interface:
double atan2d(double x, double y);
long double atan2dl(long double x, long double y);
float atan2df(float x, float y);

atanpi

Description: The atanpi function returns the principal value of the inverse tangent of \( x \), divided by pi, in the range \([-1/2, +1/2]\).

Calling interface:
double atanpi(double x);
float atanpif(float x);

cos

Description: The cos function returns the cosine of \( x \) measured in radians.

Calling interface:
double cos(double x);
long double cosl(long double x);
float cosf(float x);

cosd

Description: The cosd function returns the cosine of \( x \) measured in degrees.

Calling interface:
double cosd(double x);
long double cosdl(long double x);
float cosdf(float x);
**cospi**

**Description:** The `cospi` function returns the cosine of `x` multiplied by pi, \( \cos(x\pi) \).

**Calling interface:**
- `double cospi(double x);`
- `float cospif(float x);`

**cot**

**Description:** The `cot` function returns the cotangent of `x` measured in radians.

**errno:** ERANGE, for overflow conditions at `x = 0`.

**Calling interface:**
- `double cot(double x);`
- `long double cotl(long double x);`
- `float cotf(float x);`

**cotd**

**Description:** The `cotd` function returns the cotangent of `x` measured in degrees.

**errno:** ERANGE, for overflow conditions at `x = 0`.

**Calling interface:**
- `double cotd(double x);`
- `long double cotdl(long double x);`
- `float cotdf(float x);`

**sin**

**Description:** The `sin` function returns the sine of `x` measured in radians.

**Calling interface:**
- `double sin(double x);`
- `long double sinl(long double x);`
- `float sinf(float x);`

**sincos**

**Description:** The `sincos` function returns both the sine and cosine of `x` measured in radians.

**Calling interface:**
- `void sincos(double x, double *sinval, double *cosval);`
- `void sincosl(long double x, long double *sinval, long double *cosval);`
- `void sincosf(float x, float *sinval, float *cosval);`

**sincosd**

**Description:** The `sincosd` function returns both the sine and cosine of `x` measured in degrees.

**Calling interface:**
- `void sincosd(double x, double *sinval, double *cosval);`
- `void sincosdl(long double x, long double *sinval, long double *cosval);`
- `void sincosdf(float x, float *sinval, float *cosval);`

**sind**

**Description:** The `sind` function computes the sine of `x` measured in degrees.
**Calling interface:**
```c
double sind(double x);
long double sindl(long double x);
float sindf(float x);
```

**sinpi**

**Description:** The `sinpi` function returns the sine of x multiplied by pi, \(\sin(x \cdot \pi)\).

**Calling interface:**
```c
double sinpi(double x);
float sinpif(float x);
```

**tan**

**Description:** The `tan` function returns the tangent of x measured in radians.

**Calling interface:**
```c
double tan(double x);
long double tanl(long double x);
float tanf(float x);
```

**tand**

**Description:** The `tand` function returns the tangent of x measured in degrees.

**errno:** `ERANGE`, for overflow conditions

**Calling interface:**
```c
double tand(double x);
long double tandl(long double x);
float tandf(float x);
```

**tanpi**

**Description:** The `tanpi` function returns the tangent of x multiplied by pi, \(\tan(x \cdot \pi)\).

**Calling interface:**
```c
double tanpi(double x);
float tanpif(float x);
```

**Hyperbolic Functions**

Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The `mathimf.h` header file includes prototypes for Intel® Math Library functions.

The Intel® Math Library supports the following hyperbolic functions:

**acosh**

**Description:** The `acosh` function returns the inverse hyperbolic cosine of x.

**errno:** `EDOM`, for \(x < 1\)

**Calling interface:**
```c
double acosh(double x);
long double acoshl(long double x);
float acoshf(float x);
```
asinh
Description: The asinh function returns the inverse hyperbolic sine of x.
Calling interface:
double asinh(double x);
long double asinhl(long double x);
float asinhf(float x);

atanh
Description: The atanh function returns the inverse hyperbolic tangent of x.
errno:
EDOM, for |x| > 1
ERANGE, for x = 1
Calling interface:
double atanh(double x);
long double atanhl(long double x);
float atanhf(float x);

cosh
Description: The cosh function returns the hyperbolic cosine of x, \((e^x + e^{-x})/2\).
errno: ERANGE, for overflow conditions
Calling interface:
double cosh(double x);
long double coshl(long double x);
float coshf(float x);

sinh
Description: The sinh function returns the hyperbolic sine of x, \((e^x - e^{-x})/2\).
errno: ERANGE, for overflow conditions
Calling interface:
double sinh(double x);
long double sinhlg(long double x);
float sinhf(float x);

sinhcosh
Description: The sinhcosh function returns both the hyperbolic sine and hyperbolic cosine of x.
errno: ERANGE, for overflow conditions
Calling interface:
void sinhcosh(double x, double *sinval, double *cosval);
void sinhcoshl(long double x, long double *sinval, long double *cosval);
void sinhcoshf(float x, float *sinval, float *cosval);

tanh
Description: The tanh function returns the hyperbolic tangent of x, \((e^x - e^{-x}) / (e^x + e^{-x})\).
Calling interface:
double tanh(double x);
long double tanhl(long double x);
float tanhf(float x);

**Exponential Functions**

Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The `mathimf.h` header file includes prototypes for Intel® Math Library functions.

The Intel® Math Library supports the following exponential functions:

**cbrt**

**Description:** The `cbrt` function returns the cube root of `x`.

**Calling interface:**
```c
double cbrt(double x);
long double cbrtl(long double x);
float cbrtf(float x);
```

**exp**

**Description:** The `exp` function returns `e` raised to the `x` power, `e^x`.

**errno:** `ERANGE`, for underflow and overflow conditions

**Calling interface:**
```c
double exp(double x);
long double expl(long double x);
float expf(float x);
```

**exp10**

**Description:** The `exp10` function returns `10` raised to the `x` power, `10^x`.

**errno:** `ERANGE`, for underflow and overflow conditions

**Calling interface:**
```c
double exp10(double x);
long double exp10l(long double x);
float exp10f(float x);
```

**exp2**

**Description:** The `exp2` function returns `2` raised to the `x` power, `2^x`.

**errno:** `ERANGE`, for underflow and overflow conditions

**Calling interface:**
```c
double exp2(double x);
long double exp2l(long double x);
float exp2f(float x);
```

**expm1**

**Description:** The `expm1` function returns `e` raised to the `x` power, minus `1`, `e^x - 1`.

**errno:** `ERANGE`, for overflow conditions
**Calling interface:**

double expm1(double x);
long double expm1l(long double x);
float expmlf(float x);

**frexp**

**Description:** The `frexp` function converts a floating-point number \( x \) into signed normalized fraction in \([1/2, 1)\) multiplied by an integral power of two. The signed normalized fraction is returned, and the integer exponent stored at location `exp`.

**Calling interface:**
double frexp(double x, int *exp);
long double frexpl(long double x, int *exp);
float frexpf(float x, int *exp);

**hypot**

**Description:** The `hypot` function returns the square root of \( (x^2 + y^2) \).

**errno:** `ERANGE`, for overflow conditions

**Calling interface:**
double hypot(double x, double y);
long double hypotl(long double x, long double y);
float hypotf(float x, float y);

**ilogb**

**Description:** The `ilogb` function returns the exponent of \( x \) base two as a signed int value.

**errno:** `ERANGE`, for \( x = 0 \)

**Calling interface:**
int ilogb(double x);
int ilogbl(long double x);
int ilogbf(float x);

**invsqrt**

**Description:** The `invsqrt` function returns the inverse square root.

**Calling interface:**
double invsqrt(double x);
long double invsqrtrl(long double x);
float invsqrtf(float x);

**ldexp**

**Description:** The `ldexp` function returns \( x \cdot 2^{\text{exp}} \), where `exp` is an integer value.

**errno:** `ERANGE`, for underflow and overflow conditions

**Calling interface:**
double ldexp(double x, int exp);
long double ldexpl(long double x, int exp);
float ldexpf(float x, int exp);
**log**

**Description:** The log function returns the natural log of \( x \), \( \ln(x) \).

**errno:** EDOM, for \( x < 0 \)  
**errno:** ERANGE, for \( x = 0 \)

**Calling interface:**
- `double log(double x);`
- `long double logl(long double x);`
- `float logf(float x);`

**log10**

**Description:** The log10 function returns the base-10 log of \( x \), \( \log_{10}(x) \).

**errno:** EDOM, for \( x < 0 \)  
**errno:** ERANGE, for \( x = 0 \)

**Calling interface:**
- `double log10(double x);`
- `long double log10l(long double x);`
- `float log10f(float x);`

**log1p**

**Description:** The log1p function returns the natural log of \( (x+1) \), \( \ln(x + 1) \).

**errno:** EDOM, for \( x < -1 \)  
**errno:** ERANGE, for \( x = -1 \)

**Calling interface:**
- `double log1p(double x);`
- `long double log1pl(long double x);`
- `float log1pf(float x);`

**log2**

**Description:** The log2 function returns the base-2 log of \( x \), \( \log_{2}(x) \).

**errno:** EDOM, for \( x < 0 \)  
**errno:** ERANGE, for \( x = 0 \)

**Calling interface:**
- `double log2(double x);`
- `long double log2l(long double x);`
- `float log2f(float x);`

**logb**

**Description:** The logb function returns the signed exponent of \( x \).

**errno:** EDOM, for \( x = 0 \)

**Calling interface:**
- `double logb(double x);`
- `long double logbl(long double x);`
- `float logbf(float x);`
**pow**

**Description:** The `pow` function returns \( x \) raised to the power of \( y \), \( x^y \).

**errno:** EDOM, for \( x = 0 \) and \( y < 0 \)

**errno:** EDOM, for \( x < 0 \) and \( y \) is a non-integer

**errno:** ERANGE, for overflow and underflow conditions

**Calling interface:**
```c
double pow(double x, double y);
long double powl(double x, double y);
float powf(float x, float y);
```

**pow2o3**

**Description:** The `pow2o3` function returns the cube root of \( x \) squared, \( \text{cbrt}(x^2) \).

**Calling interface:**
```c
double pow2o3(double x);
float pow2o3f(float x);
```

**pow3o2**

**Description:** The `pow3o2` function returns the square root of the cube of \( x \), \( \text{sqrt}(x^3) \).

**errno:** EDOM, for \( x < 0 \)

**errno:** ERANGE, for overflow and underflow conditions

**Calling interface:**
```c
double pow3o2(double x);
float pow3o2f(float x);
```

**powr**

**Description:** The `powr` function returns \( x \) raised to the power of \( y \), \( x^y \), where \( x \geq 0 \).

**errno:** EDOM, for \( x < 0 \)

**errno:** ERANGE, for overflow and underflow conditions

**Calling interface:**
```c
double powr(double x, double y);
float powrf(float x, float y);
```

**scalb**

**Description:** The `scalb` function returns \( x \times 2^y \), where \( y \) is a floating-point value.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**
```c
double scalb(double x, double y);
long double scalbl(long double x, long double y);
float scalbf(float x, float y);
```

**scalbn**

**Description:** The `scalbn` function returns \( x \times 2^n \), where \( n \) is an integer value.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**
```c
double scalbn(double x, int n);
```
long double scalbnl (long double x, int n);  
float scalbnf(float x, int n);

scalbln
Description: The scalbln function returns \( x \times 2^n \), where \( n \) is a long integer value.
errno: ERANGE, for underflow and overflow conditions
Calling interface:
double scalbln(double x, long int n);
long double scalblnl (long double x, long int n);
float scalblnf(float x, long int n);

sqrt
Description: The sqrt function returns the correctly rounded square root.
errno: EDOM, for \( x < 0 \)
Calling interface:
double sqrt(double x);
long double sqrtl(long double x);
float sqrtf(float x);

Special Functions
Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.
The mathimf.h header file includes prototypes for Intel® Math Library functions.
The Intel® Math Library supports the following special functions:

annuity
Description: The annuity function computes the present value factor for an annuity, \( \frac{(1 - \frac{1+x}{y})}{x} \), where \( x \) is a rate and \( y \) is a period.
errno: ERANGE, for underflow and overflow conditions
Calling interface:
double annuity(double x, double y);
long double annuityl(long double x, long double y);
float annuityf(float x, float y);

cdfnorminv
Description: The cdfnorminv function returns the inverse cumulative normal distribution function value.
errno:
EDOM, for finite or infinite \( (x > 1) \) || \( (x < 0) \)
ERANGE, for \( x = 0 \) or \( x = 1 \)
Calling interface:
double cdfnorminv(double x);
float cdfnorminvf (float x);
**compound**

**Description:** The `compound` function computes the compound interest factor, \((1+x)^y\), where \(x\) is a rate and \(y\) is a period.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**

```c
double compound(double x, double y);
long double compoundl(long double x, long double y);
float compoundf(float x, float y);
```

**erf**

**Description:** The `erf` function returns the error function value.

**Calling interface:**

```c
double erf(double x);
long double erfl(long double x);
float erff(float x);
```

**erfc**

**Description:** The `erfc` function returns the complementary error function value.

**errno:** ERANGE, for underflow conditions

**Calling interface:**

```c
double erfc(double x);
long double erfcl(long double x);
float erfcf(float x);
```

**erfcx**

**Description:** The `erfcx` function returns the scaled complementary error function value.

**errno:** ERANGE, for overflow conditions

**Calling interface:**

```c
double erfcx(double x);
float erfcxf(float x);
```

**erfinv**

**Description:** The `erfinv` function returns the value of the inverse error function of \(x\).

**errno:** EDOM, for finite or infinite \(|x| > 1\)

**Calling interface:**

```c
double erfinv(double x);
long double erfinvl(long double x);
float erfinvf(float x);
```

**gamma**

**Description:** The `gamma` function returns the value of the logarithm of the absolute value of gamma.

**errno:** ERANGE, for overflow conditions when \(x\) is a negative integer.

**Calling interface:**

```c
double gamma(double x);
long double gammal(long double x);
```
float gammaf(float x);

**gamma_r**

**Description:** The `gamma_r` function returns the value of the logarithm of the absolute value of gamma. The sign of the gamma function is returned in the integer `signgam`.

**Calling interface:**
```
double gamma_r(double x, int *signgam);
long double gammal_r(long double x, int *signgam);
float gammaf_r(float x, int *signgam);
```

**j0**

**Description:** Computes the Bessel function (of the first kind) of `x` with order 0.

**Calling interface:**
```
double j0(double x);
long double j0l(long double x);
float j0f(float x);
```

**j1**

**Description:** Computes the Bessel function (of the first kind) of `x` with order 1.

**Calling interface:**
```
double j1(double x);
long double j1l(long double x);
float j1f(float x);
```

**jn**

**Description:** Computes the Bessel function (of the first kind) of `x` with order `n`.

**Calling interface:**
```
double jn(int n, double x);
long double jnl(int n, long double x);
float jnf(int n, float x);
```

**lgamma**

**Description:** The `lgamma` function returns the value of the logarithm of the absolute value of gamma.

**errno:** `ERANGE`, for overflow conditions, `x=0` or negative integers.

**Calling interface:**
```
double lgamma(double x);
long double lgammal(long double x);
float lgammaf(float x);
```

**lgamma_r**

**Description:** The `lgamma_r` function returns the value of the logarithm of the absolute value of gamma. The sign of the gamma function is returned in the integer `signgam`.

**errno:** `ERANGE`, for overflow conditions, `x=0` or negative integers.

**Calling interface:**
```
double lgamma_r(double x, int *signgam);
long double lgammal_r(long double x, int *signgam);
```
float lgammaf_r(float x, int *signgam);

tgamma
Description: The tgamma function computes the gamma function of x.
errno:
EDOM, for x=0 or negative integers.
ERANGE, for overflow conditions.
Calling interface:
double tgamma(double x);
long double tgammal(long double x);
float tgammaf(float x);

y0
Description: Computes the Bessel function (of the second kind) of x with order 0.
errno: EDOM, for x <= 0
Calling interface:
double y0(double x);
long double y0l(long double x);
float y0f(float x);

y1
Description: Computes the Bessel function (of the second kind) of x with order 1.
errno: EDOM, for x <= 0
Calling interface:
double y1(double x);
long double y1l(long double x);
float y1f(float x);

yn
Description: Computes the Bessel function (of the second kind) of x with order n.
errno: EDOM, for x <= 0
Calling interface:
double yn(int n, double x);
long double ynl(int n, long double x);
float ynf(int n, float x);

Nearest Integer Functions
Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.
The mathimf.h header file includes prototypes for Intel® Math Library functions.
The Intel® Math Library supports the following nearest integer functions:
ceil
Description: The ceil function returns the smallest integral value not less than \( x \) as a floating-point number.

Calling interface:

\[
\begin{align*}
\text{double } & \text{ ceil(double } x)\; ; \\
\text{long double } & \text{ ceill(long double } x)\; ; \\
\text{float } & \text{ ceilf(float } x)\; ;
\end{align*}
\]

floor
Description: The floor function returns the largest integral value not greater than \( x \) as a floating-point value.

Calling interface:

\[
\begin{align*}
\text{double } & \text{ floor(double } x)\; ; \\
\text{long double } & \text{ floorl(long double } x)\; ; \\
\text{float } & \text{ floorf(float } x)\; ;
\end{align*}
\]

llrint
Description: The llrint function returns the rounded integer value (according to the current rounding direction) as a long long int.

errno: ERANGE, for values too large

Calling interface:

\[
\begin{align*}
\text{long long int } & \text{ llrint(double } x)\; ; \\
\text{long long int } & \text{ llrintl(long double } x)\; ; \\
\text{long long int } & \text{ llrintf(float } x)\; ;
\end{align*}
\]

llround
Description: The llround function returns the rounded integer value as a long long int.

errno: ERANGE, for values too large

Calling interface:

\[
\begin{align*}
\text{long long int } & \text{ llround(double } x)\; ; \\
\text{long long int } & \text{ llroundl(long double } x)\; ; \\
\text{long long int } & \text{ llroundf(float } x)\; ;
\end{align*}
\]

lrint
Description: The lrint function returns the rounded integer value (according to the current rounding direction) as a long int.

errno: ERANGE, for values too large

Calling interface:

\[
\begin{align*}
\text{long int } & \text{ lrint(double } x)\; ; \\
\text{long int } & \text{ lrintl(long double } x)\; ; \\
\text{long int } & \text{ lrintf(float } x)\; ;
\end{align*}
\]

lround
Description: The lround function returns the rounded integer value as a long int. Halfway cases are rounded away from zero.

errno: ERANGE, for values too large
**Calling interface:**
long int lround(double x);
long int lroundl(long double x);
long int lroundf(float x);

**modf**

**Description:** The `modf` function returns the value of the signed fractional part of `x` and stores the integral part at `*iptr` as a floating-point number.

**Calling interface:**
double modf(double x, double *iptr);
long double modfl(long double x, long double *iptr);
float modff(float x, float *iptr);

**nearbyint**

**Description:** The `nearbyint` function returns the rounded integral value as a floating-point number, using the current rounding direction.

**Calling interface:**
double nearbyint(double x);
long double nearbyintl(long double x);
float nearbyintf(float x);

**rint**

**Description:** The `rint` function returns the rounded integral value as a floating-point number, using the current rounding direction.

**Calling interface:**
double rint(double x);
long double rintl(long double x);
float rintf(float x);

**round**

**Description:** The `round` function returns the nearest integral value as a floating-point number. Halfway cases are rounded away from zero.

**Calling interface:**
double round(double x);
long double roundl(long double x);
float roundf(float x);

**trunc**

**Description:** The `trunc` function returns the truncated integral value as a floating-point number.

**Calling interface:**
double trunc(double x);
long double truncl(long double x);
float truncf(float x);

**Remainder Functions**

Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.
The mathimf.h header file includes prototypes for Intel® Math Library functions.

The Intel® Math Library supports the following remainder functions:

**fmod**

**Description:** The fmod function returns the value \( x - n \times y \) for integer \( n \) such that if \( y \) is nonzero, the result has the same sign as \( x \) and magnitude less than the magnitude of \( y \).

**errno:** EDOM, for \( y = 0 \)

**Calling interface:**
- double fmod(double x, double y);
- long double fmodl(long double x, long double y);
- float fmodf(float x, float y);

**remainder**

**Description:** The remainder function returns the value of \( x \ REM y \) as required by the IEEE standard.

**errno:** EDOM, for \( y = 0 \)

**Calling interface:**
- double remainder(double x, double y);
- long double remainderl(long double x, long double y);
- float remainderf(float x, float y);

**remquo**

**Description:** The remquo function returns the value of \( x \ REM y \). In the object pointed to by \( \text{quo} \) the function stores a value whose sign is the sign of \( x/y \) and whose magnitude is congruent modulo \( 2^n \) of the integral quotient of \( x/y \). \( N \) is an implementation-defined integer. For all systems, \( N \) is equal to 31.

**errno:** EDOM, for \( y = 0 \)

**Calling interface:**
- double remquo(double x, double y, int *quo);
- long double remquol(long double x, long double y, int *quo);
- float remquof(float x, float y, int *quo);

**Miscellaneous Functions**

Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The mathimf.h header file includes prototypes for Intel® Math Library functions.

The Intel® Math Library supports the following miscellaneous functions:

**copysign**

**Description:** The copysign function returns the value with the magnitude of \( x \) and the sign of \( y \).

**Calling interface:**
- double copysign(double x, double y);
- long double copysignl(long double x, long double y);
- float copysignf(float x, float y);

**fabs**

**Description:** The fabs function returns the absolute value of \( x \).
Calling interface:

double fabs(double x);
long double fabsl(long double x);
float fabsf(float x);

defdim

Description: The fdim function returns the positive difference value, \( x - y \) (for \( x > y \)) or +0 (for \( x \leq y \)).

ererrno: ERANGE, for overflow conditions

Calling interface:

double fdim(double x, double y);
long double fdiml(long double x, long double y);
float fdimf(float x, float y);

finite

Description: The finite function returns 1 if \( x \) is not a NaN or +/- infinity. Otherwise 0 is returned.

Calling interface:

int finite(double x);
int finitel(long double x);
int finitef(float x);

fma

Description: The fma functions return \( (x*y)+z \).

Calling interface:

double fma(double x, double y, double z);
long double fmal(long double x, long double y, long double z);
float fmaf(float x, float y, float z);

fmax

Description: The fmax function returns the maximum numeric value of its arguments.

Calling interface:

double fmax(double x, double y);
long double fmaxl(long double x, long double y);
float fmaxf(float x, float y);

fmin

Description: The fmin function returns the minimum numeric value of its arguments.

Calling interface:

double fmin(double x, double y);
long double fminl(long double x, long double y);
float fminf(float x, float y);

fpclassify

Description: The fpclassify function returns the value of the number classification macro appropriate to the value of its argument.
<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (NaN)</td>
</tr>
<tr>
<td>1 (Infinity)</td>
</tr>
<tr>
<td>2 (Zero)</td>
</tr>
<tr>
<td>3 (Subnormal)</td>
</tr>
<tr>
<td>4 (Finite)</td>
</tr>
</tbody>
</table>

**Calling interface:**
```c
int fpclassify(double x);
int fpclassifyl(long double x);
int fpclassifyf(float x);
```

**isle**

**Description:** The `isle` function returns 1 if `x` is not a NaN or +/- infinity. Otherwise 0 is returned.

**Calling interface:**
```c
int isle(double x);
int islel(long double x);
int islef(float x);
```

**isgreaterequal**

**Description:** The `isgreaterequal` function returns 1 if `x` is greater than or equal to `y`. This function does not raise the invalid floating-point exception.

**Calling interface:**
```c
int isgreaterequal(double x, double y);
int isgreaterequall(long double x, long double y);
int isgreaterequalf(float x, float y);
```

**isinf**

**Description:** The `isinf` function returns a non-zero value if and only if its argument has an infinite value.

**Calling interface:**
```c
int isinf(double x);
int isinfl(long double x);
int isinff(float x);
```

**isless**

**Description:** The `isless` function returns 1 if `x` is less than `y`. This function does not raise the invalid floating-point exception.
Calling interface:
int isless(double x, double y);
int islessl(long double x, long double y);
int islesseff(float x, float y);

islessequal

Description: The islessequal function returns 1 if \( x \) is less than or equal to \( y \). This function does not raise the invalid floating-point exception.

Calling interface:
int islessequal(double x, double y);
int islessequall(long double x, long double y);
int islessequalf(float x, float y);

islessgreater

Description: The islessgreater function returns 1 if \( x \) is less than or greater than \( y \). This function does not raise the invalid floating-point exception.

Calling interface:
int islessgreater(double x, double y);
int islessgreaterl(long double x, long double y);
int islessgreaterf(float x, float y);

isnan

Description: The isnan function returns a non-zero value, if and only if \( x \) has a NaN value.

Calling interface:
int isnan(double x);
int isnanl(long double x);
int isnanf(float x);

isnormal

Description: The isnormal function returns a non-zero value, if and only if \( x \) is normal.

Calling interface:
int isnormal(double x);
int isnormall(long double x);
int isnormalf(float x);

isunordered

Description: The isunordered function returns 1 if either \( x \) or \( y \) is a NaN. This function does not raise the invalid floating-point exception.

Calling interface:
int isunordered(double x, double y);
int isunorderedl(long double x, long double y);
int isunorderedf(float x, float y);

maxmag

Description: The maxmag function returns the value of larger magnitude from among its two arguments, \( x \) and \( y \). If \(|x| > |y|\) it returns \( x \); if \(|y| > |x|\) it returns \( y \); otherwise it behaves like \( \text{fmax}(x, y) \).
Calling interface:
double maxmag(double x, double y);
float maxmagf(float x, float y);

minmag
Description: The minmag function returns the value of smaller magnitude from among its two arguments, \( x \) and \( y \). If \(|x| < |y|\) it returns \( x \); if \(|y| < |x|\) it returns \( y \); otherwise it behaves like \( \text{fmin}(x, y) \).

Calling interface:
double minmag(double x, double y);
float minmagf(float x, float y);

nan
Description: The nan function returns a quiet NaN, with content indicated through tagp.

Calling interface:
double nan(const char *tagp);
long double nanl(const char *tagp);
float nanf(const char *tagp);

nextafter
Description: The nextafter function returns the next representable value in the specified format after \( x \) in the direction of \( y \).

errno: ERANGE, for overflow and underflow conditions

Calling interface:
double nextafter(double x, double y);
long double nextafterl(long double x, long double y);
float nextafterf(float x, float y);

nexttoward
Description: The nexttoward function returns the next representable value in the specified format after \( x \) in the direction of \( y \). If \( x \) equals \( y \), then the function returns \( y \) converted to the type of the function. Use the Qlong-double option (for C++ only) on Windows* operating systems for accurate results.

errno: ERANGE, for overflow and underflow conditions

Calling interface:
double nexttoward(double x, long double y);
long double nexttowardl(long double x, long double y);
float nexttowardf(float x, long double y);

signbit
Description: The signbit function returns a non-zero value, if and only if the sign of \( x \) is negative.

Calling interface:
int signbit(double x);
int signbitl(long double x);
int signbitf(float x);
**significand**

**Description:** The significand function returns the significand of \( x \) in the interval \([1,2)\). For \( x \) equal to zero, NaN, or +/- infinity, the original \( x \) is returned.

**Calling interface:**
```c
double significand(double x);
long double significandl(long double x);
float significandf(float x);
```

**Complex Functions**

Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

The mathimf.h header file includes prototypes for Intel® Math Library functions.

The Intel® Math Library supports the following complex functions:

**cabs**

**Description:** The cabs function returns the complex absolute value of \( z \).

**Calling interface:**
```c
double cabs(double _Complex z);
long double cabsl(long double _Complex z);
float cabsf(float _Complex z);
```

**cacos**

**Description:** The cacos function returns the complex inverse cosine of \( z \).

**Calling interface:**
```c
double _Complex cacos(double _Complex z);
long double _Complex cacosl(long double _Complex z);
float _Complex cacosf(float _Complex z);
```

**cacosh**

**Description:** The cacosh function returns the complex inverse hyperbolic cosine of \( z \).

**Calling interface:**
```c
double _Complex cacosh(double _Complex z);
long double _Complex cacoshl(long double _Complex z);
float _Complex cacoshf(float _Complex z);
```

**carg**

**Description:** The carg function returns the value of the argument in the interval \([-\pi, +\pi]\).

**Calling interface:**
```c
double carg(double _Complex z);
long double cargl(long double _Complex z);
float cargf(float _Complex z);
```

**casin**

**Description:** The casin function returns the complex inverse sine of \( z \).

**Calling interface:**
double _Complex casin(double _Complex z);
long double _Complex casinl(long double _Complex z);
float _Complex casinf(float _Complex z);

**casinh**

**Description:** The **casinh** function returns the complex inverse hyperbolic sine of **z**.

**Calling interface:**
double _Complex casinh(double _Complex z);
long double _Complex casinhl(long double _Complex z);
float _Complex casinhf(float _Complex z);

catan

**Description:** The **catan** function returns the complex inverse tangent of **z**.

**Calling interface:**
double _Complex catan(double _Complex z);
long double _Complex catanl(long double _Complex z);
float _Complex catanf(float _Complex z);

catanh

**Description:** The **catanh** function returns the complex inverse hyperbolic tangent of **z**.

**Calling interface:**
double _Complex catanh(double _Complex z);
long double _Complex catanhl(long double _Complex z);
float _Complex catanhf(float _Complex z);

cos

**Description:** The **ccos** function returns the complex cosine of **z**.

**Calling interface:**
double _Complex ccos(double _Complex z);
long double _Complex ccosl(long double _Complex z);
float _Complex ccosf(float _Complex z);

cosh

**Description:** The **ccosh** function returns the complex hyperbolic cosine of **z**.

**Calling interface:**
double _Complex ccosh(double _Complex z);
long double _Complex ccoshl(long double _Complex z);
float _Complex ccoshf(float _Complex z);

cexp

**Description:** The **cexp** function returns \( e^z \) (e raised to the power **z**).

**Calling interface:**
double _Complex cexp(double _Complex z);
long double _Complex cexpl(long double _Complex z);
float _Complex cexpf(float _Complex z);
cexp2
Description: The cexp function returns \(2^z\) (2 raised to the power \(z\)).

Calling interface:
```c
double _Complex cexp2(double _Complex z);
long double _Complex cexp2l(long double _Complex z);
float _Complex cexp2f(float _Complex z);
```

cexp10
Description: The cexp10 function returns \(10^z\) (10 raised to the power \(z\)).

Calling interface:
```c
double _Complex cexp10(double _Complex z);
long double _Complex cexp10l(long double _Complex z);
float _Complex cexp10f(float _Complex z);
```

cimag
Description: The cimag function returns the imaginary part value of \(z\).

Calling interface:
```c
double cimag(double _Complex z);
long double cimagl(long double _Complex z);
float cimagf(float _Complex z);
```

cis
Description: The cis function returns the cosine and sine (as a complex value) of \(z\) measured in radians.

Calling interface:
```c
double _Complex cis(double x);
long double _Complex cisl(long double z);
float _Complex cisf(float z);
```

cisd
Description: The cisd function returns the cosine and sine (as a complex value) of \(z\) measured in degrees.

Calling interface:
```c
double _Complex cisd(double x);
long double _Complex cisdl(long double z);
float _Complex cisdf(float z);
```

clog
Description: The clog function returns the complex natural logarithm of \(z\).

Calling interface:
```c
double _Complex clog(double _Complex z);
long double _Complex clogl(long double _Complex z);
float _Complex clogf(float _Complex z);
```

clog2
Description: The clog2 function returns the complex logarithm base 2 of \(z\).

Calling interface:
```c
double _Complex clog2(double _Complex z);
```
long double _Complex clog2l(long double _Complex z);
float _Complex clog2f(float _Complex z);

clog10
Description: The clog10 function returns the complex logarithm base 10 of z.
Calling interface:
double _Complex clog10(double _Complex z);
long double _Complex clog10l(long double _Complex z);
float _Complex clog10f(float _Complex z);

conj
Description: The conj function returns the complex conjugate of z by reversing the sign of its imaginary part.
Calling interface:
double _Complex conj(double _Complex z);
long double _Complex conjl(long double _Complex z);
float _Complex conjf(float _Complex z);

cpow
Description: The cpow function returns the complex power function, \(x^y\).
Calling interface:
double _Complex cpow(double _Complex x, double _Complex y);
long double _Complex cpowl(long double _Complex x, long double _Complex y);
float _Complex cpowf(float _Complex x, float _Complex y);

cproj
Description: The cproj function returns a projection of z onto the Riemann sphere.
Calling interface:
double _Complex cproj(double _Complex z);
long double _Complex cprojl(long double _Complex z);
float _Complex cprojf(float _Complex z);

creal
Description: The creal function returns the real part of z.
Calling interface:
double creal(double _Complex z);
long double creall(long double _Complex z);
float crealf(float _Complex z);

csin
Description: The csin function returns the complex sine of z.
Calling interface:
double _Complex csin(double _Complex z);
long double _Complex csinl(long double _Complex z);
float _Complex csinf(float _Complex z);
csinh
Description: The csinh function returns the complex hyperbolic sine of z.

Calling interface:
double _Complex csinh(double _Complex z);
long double _Complex csinhl(long double _Complex z);
float _Complex csinhf(float _Complex z);

csqrt
Description: The csqrt function returns the complex square root of z.

Calling interface:
double _Complex csqrt(double _Complex z);
long double _Complex csqrtl(long double _Complex z);
float _Complex csqrtf(float _Complex z);

ctan
Description: The ctan function returns the complex tangent of z.

Calling interface:
double _Complex ctan(double _Complex z);
long double _Complex ctanl(long double _Complex z);
float _Complex ctanf(float _Complex z);

ctanhl
Description: The ctan function returns the complex hyperbolic tangent of z.

Calling interface:
double _Complex ctanh(double _Complex z);
long double _Complex ctanhl(long double _Complex z);
float _Complex ctanhf(float _Complex z);

C99 Macros
Many routines in the Intel® Math Library are more optimized for Intel® microprocessors than for non-Intel microprocessors.
The mathimf.h header file includes prototypes for Intel® Math Library functions.
The Intel® Math Library and mathimf.h header file support the following C99 macros:

int fpclassify(x);
int isfinite(x);
int isgreater(x, y);
int isgreaterequal(x, y);
int isinf(x);
int isless(x, y);
int islessequal(x, y);
int islessgreater(x, y);
int isnan(x);
int isnormal(x);
int isunordered(x, y);
int signbit(x);
See Also
Miscellaneous Functions
Compatibility and Portability

This section contains information about conformance to language standards, language compatibility, and portability.

Conformance to the C/C++/DPC++ Standards

The Intel® oneAPI DPC++/C++ Compiler conforms to the following standards:

- C++14 standard (ISO/IEC 14882:2014)
- C++11 standard (ISO/IEC 14882:2011)

**NOTE** For SYCL* specific standards, visit the SYCL 2020 specification.

GCC* Compatibility and Interoperability

This topic applies to Linux*.

The Intel® oneAPI DPC++/C++ Compiler is compatible with most versions of the GNU* Compiler Collection (GCC*). The release notes contains a list of compatible versions.

C language object files created with the compiler are binary compatible with the GCC and C/C++ language library. You can use the Intel® oneAPI DPC++/C++ Compiler or the GCC compiler to pass object files to the linker.

**NOTE** When using an Intel software development product that includes an Intel® oneAPI DPC++/C++ Compiler with a Clang front-end, you can also use icx or icpx.

The Intel® oneAPI DPC++/C++ Compiler supports many of the language extensions provided by the GNU compilers. See http://www.gnu.org for more information.
NOTE Statement expressions are supported, except that the following are prohibited inside them:

- dynamically-initialized local static variables
- local non-POD class definitions
- try/catch
- variable length arrays

Branching out of a statement expression and statement expressions in constructor initializers are not allowed. Variable-length arrays are no longer allowed in statement expressions.

NOTE The Intel® oneAPI DPC++/C++ Compiler supports GCC-style inline ASM if the assembler code uses AT&T* System V/386 syntax.

GCC Interoperability

C++ compilers are interoperable if they can link object files and libraries generated by one compiler with object files and libraries generated by the second compiler, and the resulting executable runs successfully. The Intel® oneAPI DPC++/C++ Compiler is highly compatible with the GNU compilers.

The Intel® oneAPI DPC++/C++ Compiler and GCC support the following predefined macros:

- __GNUC__
- __GNUG__
- __GNUC_MINOR__
- __GNUC_PATCHLEVEL__

Caution Not defining these macros results in different paths through system header files. These alternate paths may be poorly tested or otherwise incompatible.

How the Compiler Uses GCC

The Intel® oneAPI DPC++/C++ Compiler uses the GNU tools on the system, such as the GNU header files, including stdio.h, and the GNU linker and libraries. So the compiler has to be compatible with the version of GCC or G++* you have on your system.

By default, the compiler determines which version of GCC or G++ you have installed from the PATH environment variable.

If you want use a version of GCC or G++ other than the default version on your system, you need to use the -gcc-toolchain compiler option to specify the location of the base toolchain. For example:

- You want to build something that cannot be compiled by the default version of the system compiler, so you need to use a legacy version for compatibility, such as if you want to use third party libraries that are not compatible with the default version of the system compiler.
- You want to use a later version of GCC or G++ than the default system compiler.

The Intel® oneAPI DPC++/C++ Compiler driver uses the default version of GCC/G++, or the version you specify, to extract the location of the headers and libraries.

Compatibility with Open Source Tools

The Intel® oneAPI DPC++/C++ Compiler includes improved support for the following open source tools:

- GNU Libtool – a script that allows package developers to provide generic shared library support.
- Valgrind – a flexible system for debugging and profiling executables running on x86 processors.
- GNU Automake – a tool for automatically generating Makefile.ins from files called Makefile.am.
Microsoft* Compatibility

This content is specific to C++; it does not apply to DPC++.

The Intel® oneAPI DPC++/C++ Compiler is fully source- and binary-compatible (native code only) with Microsoft Visual C++* (MSVC). You can debug binaries built with the Intel oneAPI DPC++/C++ Compiler from within the Microsoft Visual Studio* environment.

The compiler supports security checks with the /GS option. You can control this option in the Microsoft Visual Studio IDE by using C/C++ > Code Generation > Buffer Security Check.

Microsoft Visual Studio Integration

The compiler is compatible with Microsoft Visual Studio 2017 and 2019 projects.

NOTE Support for Microsoft Visual Studio 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

The compiler only supports native C++ project types provided by Microsoft Visual Studio development environment. The project types with .NET* attributes such as the ones below, cannot be converted to an Intel C++ project:

- Empty Project (.NET)
- Class Library (.NET)
- Console Application (.NET)
- Windows Control Library (.NET)
- Windows Forms Application (.NET)
- Windows Service (.NET)

Unsupported Major Features

- COM Attributes
- C++ Accelerated Massive Parallelism (C++ AMP)
- Managed extensions for C++ (new pragmas, keywords, and command-line options)
- Event handling (new keywords)
- __abstract keyword
- __box keyword
- __delegate keyword
- __gc keyword
- __identifier keyword
- __no_gc keyword
- __pin keyword
- __property keyword
- __seal keyword
- __try_cast keyword
- __w64 keyword

Unsupported Preprocessor Features

- #import directive changes for attributed code
- #using directive
- managed, unmanaged pragmas
• _MANAGED macro
• runtime_checks pragma

Mixing Managed and Unmanaged Code
If you use the managed extensions to the C++ language in Microsoft Visual Studio .NET, you can use the compiler for your non-managed code for better application performance. Make sure managed keywords do not appear in your non-managed code.

For information on how to mix managed and unmanaged code, refer to the article, An Overview of Managed/Unmanaged Code Interoperability, on the Microsoft Web site.

See Also
/ GS compiler option

Precompiled Header Support
There are some differences in how precompiled header (PCH) files are supported between the Intel oneAPI DPC++/C++ Compiler and the Microsoft Visual C++* Compiler. These differences include the following:

• The PCH information generated by the Intel oneAPI DPC++/C++ Compiler is not compatible with the PCH information generated by the Microsoft Visual Studio Compiler.
• The Intel oneAPI DPC++/C++ Compiler does not support PCH generation and use in the same translation unit.

Compilation and Execution Differences
While the Intel oneAPI DPC++/C++ Compiler is compatible with the Microsoft Visual C++* Compiler, some differences can prevent successful compilation. Also there can be some incompatible generated-code behavior of some source files with the Intel oneAPI DPC++/C++ Compiler. In most cases, a modification of the user source file enables successful compilation with both the Intel oneAPI DPC++/C++ Compiler and the Microsoft Visual C++ Compiler. The differences between the compilers are listed as follows:

Inline Assembly Target Labels (IA-32 Architecture Only)
This content is specific to C++; it does not apply to DPC++.

For compilations targeted for IA-32 architecture, inline assembly target labels of goto statements are case sensitive. The Microsoft Visual C++ compiler treats these labels in a case insensitive manner. For example, the Intel oneAPI DPC++/C++ Compiler issues an error when compiling the following code:

```cpp
int func(int x) {
    goto LAB2;
    // error: label "LAB2" was referenced but not defined
    __asm lab2: mov x, 1
    return x;
}
```

However, the Microsoft Visual C++ Compiler accepts the preceding code. As a work-around for the Intel oneAPI DPC++/C++ Compiler, when a goto statement refers to a label defined in inline assembly, you must match the label reference with the label definition in both name and case.

Inlining Functions Marked for dllimport
The Intel oneAPI DPC++/C++ Compiler will attempt to inline any functions that are marked dllimport but Microsoft* will not. Therefore, any calls or variables used inside a dllimport routine needs to be available at link time or the result will be an unresolved symbol.
Example

The following example contains two files: header.h and bug.cpp.

header.h

```cpp
#ifndef _HEADER_H
#define _HEADER_H

namespace Foo_NS {

    class Foo2 {
        public:
            Foo2(){};
            ~Foo2();
            static int test(int m_i);
        }
    }

#define _HEADER_H
```

bug.cpp

```cpp
#include "header.h"

struct Foo2 {
    static void test();
};

struct __declspec(dllimport) Foo {
    void getI() { Foo2::test(); }
};

struct C {
    virtual void test();
};

void C::test() { Foo* p; p->getI(); }

int main() {
    return 0;
}
```

Enum Bit-Field Signedness

The Intel® oneAPI DPC++/C++ Compiler and Microsoft® Visual C++* differ in how they attribute signedness to bit fields declared with an `enum` type. Microsoft Visual C++ always considers `enum` bit fields to be signed, even if not all values of the `enum` type can be represented by the bit field.

The Intel oneAPI DPC++/C++ Compiler considers an `enum` bit field to be unsigned, unless the `enum` type has at least one `enum` constant with a negative value. In any case, the Intel oneAPI DPC++/C++ Compiler produces a warning if the bit field is declared with too few bits to represent all the values of the `enum` type.

Portability

This section contains information about porting from the Microsoft® Compiler or from GCC* to the Intel® oneAPI DPC++/C++ Compiler.
Porting from Microsoft* Visual C++* to the Intel® oneAPI DPC++/C++ Compiler

This section describes a basic approach to porting applications from Microsoft* Visual C++* for Windows* to the Intel® oneAPI DPC++/C++ Compiler for Windows.

If you build your applications from the Windows command line, you can port applications from Microsoft Visual C++ to the Intel® oneAPI DPC++/C++ Compiler by modifying your makefile to invoke the Intel® oneAPI DPC++/C++ Compiler instead of Microsoft Visual C++.

The Intel® oneAPI DPC++/C++ Compiler integration with Microsoft Visual Studio provides a conversion path to the Intel® oneAPI DPC++/C++ Compiler that allows you to build your Visual C++ projects with the Intel® oneAPI DPC++/C++ Compiler. This version of the Intel® oneAPI DPC++/C++ Compiler supports:

- Microsoft Visual Studio 2019
- Microsoft Visual Studio 2017

**NOTE** Support for Microsoft Visual Studio 2017 is deprecated as of the Intel® oneAPI 2022.1 release, and will be removed in a future release.

See the appropriate section in this documentation for details on using the Intel® oneAPI DPC++/C++ Compiler with Microsoft Visual Studio.

The Intel® oneAPI DPC++/C++ Compiler also supports many of the same compiler options, macros, and environment variables you already use in your Microsoft work.

One challenge in porting applications from one compiler to another is making sure there is support for the compiler options you use to build your application. The Compiler Options reference lists compiler options that are supported by both the Intel® oneAPI DPC++/C++ Compiler and Microsoft C++.

**See Also**

Other Considerations
Modifying Your Makefile

### Modifying Your makefile

If you use makefiles to build your Microsoft* application, you need to change the value for the compiler variable to use the Intel® oneAPI DPC++/C++ Compiler. You may also want to review the options specified by **CPPFLAGS**. A simple example follows:

**Microsoft makefile Example**

```makefile
# name of the program
PROGRAM = area.exe

# names of source files
CPPSOURCES = area_main.cpp area_functions.cpp

# names of object files
CPPOBJECTS = area_main.obj area_functions.obj

# Microsoft(R) compiler options
CPPFLAGS = /RTC1 /EHsc

# Use Microsoft C++(R)
CPP = cl
```
**Microsoft makefile Example**

```
# link objects
$(PROGRAM): $(CPPOBJECTS)
    link.exe /out:$@ $(CPPOBJECTS)

# build objects
area_main.obj: area_main.cpp area_headers.h
area_functions.obj: area_functions.cpp area_headers.h

# clean
clean:  del  $(CPPOBJECTS) $(PROGRAM)
```

**Modified makefile for the Intel® oneAPI DPC++/C++ Compiler**

Before you can run `nmake` with the Intel® oneAPI DPC++/C++ Compiler, you need to set the proper environment. In this example, only the name of the compiler changed:

```
# name of the program
PROGRAM = area.exe

# names of source files
CPPSOURCES = area_main.cpp area_functions.cpp

# names of object files
CPPOBJECTS = area_main.obj area_functions.obj

# Intel(R) C/C++/DPC++ Compiler options
CPPFLAGS = /RTC1 /EHsc

# Use the Intel(R) C/C++/DPC++ Compiler
CPP = [invocation]

# link objects
$(PROGRAM): $(CPPOBJECTS)
    link.exe /out:$@ $(CPPOBJECTS)

# build objects
area_main.obj: area_main.cpp area_headers.h
area_functions.obj: area_functions.cpp area_headers.h

# clean
clean:  del  $(CPPOBJECTS) $(PROGRAM)
```

With the modified makefile, the output of `nmake` is similar to the following:

```
Microsoft (R) Program Maintenance Utility Version 8.00.50727.42
Copyright (C) Microsoft Corporation. All rights reserved.
    icx /RTC1 /EHsc /c area_main.cpp area_functions.cpp

Intel(R) Compiler for applications running on IA-32 or IA-64
Copyright (C) 1985-2006 Intel Corporation. All rights reserved.
    area_main.cpp
```
area_functions.cpp
  link.exe /out:area.exe area_main.obj area_functions.obj

Microsoft (R) Incremental Linker Version 8.00.50727.42
Copyright (C) Microsoft Corporation. All rights reserved.

Using IPO in makefiles

By default, IPO generates dummy object files containing interprocedural information used by the compiler. To link or create static libraries with these object files requires specific Intel-provided tools. To use them in your makefile, replace references to `link` with `xilink` and references to `lib` with `xilib`:

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| # name of the program
  PROGRAM = area.exe

# names of source files
  CPPSOURCES = area_main.cpp area_functions.cpp

# names of object files
  CPPOBJECTS = area_main.obj area_functions.obj

# Intel C/C++/DPC++ Compiler options
  CPPFLAGS = /RTC1 /EHsc /Qipo

# Use the Intel C/C++/DPC++ Compiler
  CPP = [invocation]

# link objects
  $(PROGRAM): $(CPPOBJECTS)
    xilink.exe /out:$@ $(CPPOBJECTS)

# build objects
  area_main.obj: area_main.cpp area_headers.h
  area_functions.obj: area_functions.cpp area_headers.h

# clean
  clean:  del $(CPPOBJECTS) $(PROGRAM)

Where [invocation] is icx for C++ or dpcpp-cl for DPC++.

Other Considerations

There are some notable differences between the Intel® oneAPI DPC++/C++ Compiler and the Microsoft* Compiler. Consider the following as you begin compiling your code with the Intel® oneAPI DPC++/C++ Compiler.

Setting the Environment

The compiler installation provides a batch file, setvars.bat, that sets the proper environment for the Intel® oneAPI DPC++/C++ Compiler. For information on running setvars.bat, see Specifying the Location of Compiler Components.
Using Optimization

The Intel® oneAPI DPC++/C++ Compiler is an optimizing compiler that begins with the assumption that you want improved performance from your application when it is executed on Intel® architecture. Consequently, certain optimizations, such as option O2, are part of the default invocation of the compiler. By default, Microsoft turns off optimization, which is the equivalent of compiling with options Od or O0. Other forms of the O[n] option compare as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Intel® oneAPI DPC++/C++ Compiler</th>
<th>Microsoft Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>/od</td>
<td>Turns off all optimization. Same as O0.</td>
<td>Default. Turns off all optimization.</td>
</tr>
<tr>
<td>/o1</td>
<td>Decreases code size with some increase in speed.</td>
<td>Optimizes code for minimum size.</td>
</tr>
<tr>
<td>/o2</td>
<td>Default. Favors speed optimization with some increase in code size. Intrinsics, loop unrolling, and inlining are performed.</td>
<td>Optimizes code for maximum speed.</td>
</tr>
<tr>
<td>/o3</td>
<td>Enables -O2 optimizations plus more aggressive optimizations, such as prefetching, scalar replacement, and loop and memory access transformations.</td>
<td>Not supported.</td>
</tr>
</tbody>
</table>

Modifying Your Configuration

The Intel® oneAPI DPC++/C++ Compiler lets you maintain configuration and response files that are part of compilation. Options stored in the configuration file apply to every compilation, while options stored in response files apply only where they are added on the command line. If you have several options in your makefile that apply to every build, you may find it easier to move these options to the configuration file (.\bin\icx.cfg).

In a multi-user, networked environment, options listed in the icx.cfg file are generally intended for everyone who uses the compiler. If you need a separate configuration, you can use the ICXCFG environment variable to specify the name and location of your own .cfg file, such as \my_code\my_config.cfg. Anytime you instruct the compiler to use a different configuration file, the icx.cfg system configuration file is ignored.

Using the Intel Libraries

The Intel® oneAPI DPC++/C++ Compiler supplies additional libraries that contain optimized implementations of many commonly used functions. Some of these functions are implemented using CPU dispatch. This means that different code may be executed when run on different processors.

Supplied libraries include the Intel® Math Library (libm), the Short Vector Math Library (svml_disp), libirc, as well as others. These libraries are linked in by default when the compiler sees that references to them have been generated. Some library functions, such as sin or memset, may not require a call to the library, since the compiler may inline the code for the function.

Intel Math Library (libm)

With the Intel® oneAPI DPC++/C++ Compiler, the Intel Math Library, libm, is linked by default when calling math functions that require the library. Some functions, such as sin, may not require a call to the library, since the compiler already knows how to compute the sin function. The Intel Math Library also includes some functions not found in the standard math library.

**NOTE**
You cannot make calls to the Intel Math Library with the Microsoft Compiler.
Many routines in the *libimf* library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

**Short Vector Math Library (svml Disp)**

When vectorization is in progress, the compiler may translate some calls to the *libm* math library functions into calls to *svml Disp* functions. These functions implement the same basic operations as the Intel Math Library, but operate on short vectors of operands. This results in greater efficiency. In some cases, the *svml Disp* functions are slightly less precise than the equivalent *libm* functions.

Many routines in the Short Vector Math Library (SVML) are more optimized for Intel® microprocessors than for non-Intel microprocessors.

**libirc**

*libirc* contains optimized implementations of some commonly used string and memory functions. For example, it contains functions that are optimized versions of *memcpy* and *memset*. The compiler will automatically generate calls to these functions when it sees calls to *memcpy* and *memset*. The compiler may also transform loops that are equivalent to *memcpy* or *memset* into calls to these functions.

Many routines in the *libirc* library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

**See Also**
- compiler option
- Differences in PCH Support
- Using Configuration Files
- Using Response Files
- Specifying the Location of Compiler Components

**Porting from GCC* to the Intel® oneAPI DPC++/C++ Compiler**

This section describes a basic approach to porting applications from the (GNU Compiler Collection*) GCC C/C++ compilers to the Intel® oneAPI DPC++/C++ Compiler. These compilers correspond to each other as follows:

<table>
<thead>
<tr>
<th>Language</th>
<th>Intel® Compiler</th>
<th>GCC Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>icx for C++ or dpcpp for DPC++.</td>
<td>gcc</td>
</tr>
<tr>
<td>C++</td>
<td>icpx for C++ or dpcpp for DPC++.</td>
<td>g++</td>
</tr>
</tbody>
</table>

**NOTE** Unless otherwise indicated, the term "gcc" refers to both GCC and G++* compilers from the GCC.
Advantages to Using the Intel® oneAPI DPC++/C++ Compiler

In many cases, porting applications from gcc to the Intel® oneAPI DPC++/C++ Compiler can be as easy as modifying your makefile to invoke the Intel® oneAPI DPC++/C++ Compiler (icx for C++ or dpcpp for DPC++) instead of gcc. Using the Intel® oneAPI DPC++/C++ Compiler typically improves the performance of your application, especially for those that run on Intel processors. In many cases, your application’s performance may also show improvement when running on non-Intel processors. When you compile your application with the Intel® oneAPI DPC++/C++ Compiler, you have access to:

- Compiler options that optimize your code for the latest Intel® architecture processors.
- Advanced profiling tools (PGO) similar to the GNU profiler gprof.
- High-level optimizations (HLO).
- Interprocedural optimization (IPO).
- Intel intrinsic functions that the compiler uses to inline instructions, including various versions of Intel® Streaming SIMD Extensions and Intel® Advanced Vector Extensions.
- Highly-optimized Intel® Math Library for improved accuracy.

Because the Intel® oneAPI DPC++/C++ Compiler is compatible and interoperable with gcc, porting your gcc application to the Intel® oneAPI DPC++/C++ Compiler includes the benefits of binary compatibility. As a result, you should not have to re-build libraries from your gcc applications. The Intel® oneAPI DPC++/C++ Compiler also supports many of the same compiler options, macros, and environment variables you already use in your gcc work.

<table>
<thead>
<tr>
<th>Product and Performance Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance varies by use, configuration and other factors. Learn more at <a href="http://www.Intel.com/PerformanceIndex">www.Intel.com/PerformanceIndex</a>.</td>
</tr>
<tr>
<td>Notice revision #20201201</td>
</tr>
</tbody>
</table>

See Also
Modifying Your makefile
Supported Environment Variables

Modifying Your makefile

If you use makefiles to build your GCC* application, you need to change the value for the GCC compiler variable to use the Intel® oneAPI DPC++/C++ Compiler. You may also want to review the options specified by CFLAGS. For example:

<table>
<thead>
<tr>
<th>Sample GCC makefile</th>
</tr>
</thead>
<tbody>
<tr>
<td># Use gcc compiler</td>
</tr>
<tr>
<td>CC = gcc</td>
</tr>
<tr>
<td># Compile-time flags</td>
</tr>
<tr>
<td>CFLAGS = -O2 -std=c99</td>
</tr>
<tr>
<td>all: area_app</td>
</tr>
<tr>
<td>area_app: area_main.o area_functions.o</td>
</tr>
<tr>
<td>$(CC) area_main.o area_functions.o -o area</td>
</tr>
<tr>
<td>area_main.o: area_main.c</td>
</tr>
<tr>
<td>$(CC) -c $(CFLAGS) area_main.c</td>
</tr>
</tbody>
</table>
### Sample GCC makefile

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area_functions.o: area_functions.c</td>
<td>$(CC) -c -fno-asm $(CFLAGS) area_functions.c</td>
</tr>
<tr>
<td>clean: rm -rf *o area</td>
<td></td>
</tr>
</tbody>
</table>

### Sample makefile modified for the Intel® oneAPI DPC++/C++ Compiler

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td># Use Intel C/C++/DPC++ Compiler</td>
<td>CC = [invocation]</td>
</tr>
<tr>
<td># Compile-time flags</td>
<td>CFLAGS = -std=c99</td>
</tr>
<tr>
<td>all: area_app</td>
<td></td>
</tr>
<tr>
<td>area_app: area_main.o area_functions.o</td>
<td>$(CC) area_main.o area_functions.o -o area</td>
</tr>
<tr>
<td>area_main.o: area_main.c</td>
<td>$(CC) -c $(CFLAGS) area_main.c</td>
</tr>
<tr>
<td>area_functions.o: area_functions.c</td>
<td>$(CC) -c -fno-asm $(CFLAGS) area_functions.o</td>
</tr>
<tr>
<td>clean: rm -rf *o area</td>
<td></td>
</tr>
</tbody>
</table>

Where [invocation] is *icpx* (for C++) or *dpcpp* (for DPC++).

If your GCC code includes features that are not supported with the Intel® oneAPI DPC++/C++ Compiler (compiler options, language extensions, macros, pragmas, and so on), you can compile those sources separately with GCC if necessary.

In the above makefile, *area_functions.c* is an example of a source file that includes features unique to GCC. Because the Intel® oneAPI DPC++/C++ Compiler uses the O2 option by default and GCC uses option O0 as the default, we instruct GCC to compile at option O2. We also include the -fno-asm switch from the original makefile because this switch is not supported with the Intel® oneAPI DPC++/C++ Compiler.

### Sample makefile modified for using the Intel® oneAPI DPC++/C++ Compiler and GCC together

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td># Use Intel C/C++/DPC++ Compiler</td>
<td>CC = [invocation]</td>
</tr>
<tr>
<td># Use gcc for files that cannot be compiled by [invocation]</td>
<td>GCC = gcc</td>
</tr>
<tr>
<td># Compile-time flags</td>
<td>CFLAGS = -std=c99</td>
</tr>
<tr>
<td>all: area_app</td>
<td></td>
</tr>
<tr>
<td>area_app: area_main.o area_functions.o</td>
<td>$(CC) area_main.o area_functions.o -o area</td>
</tr>
<tr>
<td>area_main.o: area_main.c</td>
<td>$(CC) -c $(CFLAGS) area_main.c</td>
</tr>
<tr>
<td>area_functions.o: area_functions.c</td>
<td>$(CC) -c -fno-asm $(CFLAGS) area_functions.o</td>
</tr>
</tbody>
</table>
Sample makefile modified for using the Intel® oneAPI DPC++/C++ Compiler and GCC together

```
$(GCC) -c -O2 -fno-asm $(CFLAGS) area_functions.c
```

```
clean: rm -rf *o area
```

Where `[invocation]` is `icpx` (for C++) or `dpcpp` (for DPC++).

Output of `make` using a modified makefile

```
[invocation] -c -std=c99 area_main.c
gcc -c -O2 -fno-asm -std=c99 area_functions.c
[invocation] area_main.o area_functions.o -o area
```

Where `[invocation]` is `icpx` (for C++) or `dpcpp` (for DPC++).

**Using IPO in Makefiles**

By default, IPO generates "dummy" object files containing Interprocedural information used by the compiler. To link or create static libraries with these object files requires special Intel®-provided tools. To use them in your makefile, simply replace references to "ld" with "xild" and references to "ar" with "xiar", or use `icx` or `icpx` (for C++) or `dpcpp` (for DPC++) to link as shown below.

Sample makefile modified for the Intel® oneAPI DPC++/C++ Compiler with IPO

```
# Use Intel C/C++/DPC++ Compiler
CC = [invocation]
# Compile-time flags
CFLAGS = -std=c99 -ipo
all: area_app

area_app: area_main.o area_functions.o
   $(CC) area_main.o area_functions.o -o area

area_main.o: area_main.c
   $(CC) -c $(CFLAGS) area_main.c

area_functions.o: area_functions.c
   $(CC) -c $(CFLAGS) area_functions.c

clean: rm -rf *o area
```

Where `[invocation]` is `icpx` (for C++) or `dpcpp` (for DPC++).

**Equivalent Macros**

The Intel® oneAPI DPC++/C++ Compiler is compatible with the predefined GNU* macros. See http://gcc.gnu.org for a list of compatible predefined macros.

**See Also**

Additional Predefined Macros

**Other Considerations**

There are some notable differences between the Intel® oneAPI DPC++/C++ Compiler and GCC*. Consider the following as you begin compiling your source code with the Intel® oneAPI DPC++/C++ Compiler.
Setting the Environment

The Intel® oneAPI DPC++/C++ Compiler relies on environment variables for the location of compiler binaries, libraries, man pages, and license files. In some cases these are different from the environment variables that GCC uses. Another difference is that these variables are not set by default after installing the Intel® oneAPI DPC++/C++ Compiler. The following environment variables can be set prior to running the Intel® oneAPI DPC++/C++ Compiler:

- **PATH**: Add the location of the compiler binaries to PATH.
- **LD_LIBRARY_PATH**: Sets the location where the generated executable picks up the runtime libraries (*.so files).
- **MANPATH**: add the location of the compiler man pages (icx or icpx for C++ or dpcpp for DPC++) to MANPATH.

To set these environment variables, you can source the `setvars.sh` script (e.g., `source setvars.sh`).

**NOTE**
Setting these environment variables with `setvars.sh` does not impose a conflict with GCC. You should be able to use both compilers in the same shell.

Using Optimization

The Intel® oneAPI DPC++/C++ Compiler is an optimizing compiler that begins with the assumption that you want improved performance from your application when it is executed on Intel® architecture. Consequently, certain optimizations, such as option `O2`, are part of the default invocation of the Intel® oneAPI DPC++/C++ Compiler. Optimization is turned off in GCC by default, the equivalent of compiling with option `O0`. Other forms of the `O<n>` option compare as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Intel® oneAPI DPC++/C++ Compiler</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>-00</td>
<td>Turns off optimization.</td>
<td>Default. Turns off optimization.</td>
</tr>
<tr>
<td>-01</td>
<td>Decreases code size with some increase in speed.</td>
<td>Decreases code size with some increase in speed.</td>
</tr>
<tr>
<td>-02</td>
<td><strong>Default.</strong> Favors speed optimization with some increase in code size. Same as option <code>O</code>. Intrinsics, loop unrolling, and inlining are performed.</td>
<td>Optimizes for speed as long as there is not an increase in code size. Loop unrolling and function inlining, for example, are not performed.</td>
</tr>
<tr>
<td>-03</td>
<td>Enables option <code>O2</code> optimizations plus more aggressive optimizations, such as prefetching, scalar replacement, and loop and memory access transformations.</td>
<td>Optimizes for speed while generating larger code size. Includes option <code>O2</code> optimizations plus loop unrolling and inlining.</td>
</tr>
</tbody>
</table>

Targeting Intel® Processors

While many of the same options that target specific processors are supported with both compilers, Intel includes options that utilize processor-specific instruction scheduling to target the latest Intel® processors.
Modifying Your Configuration

The Intel® oneAPI DPC++/C++ Compiler lets you maintain configuration and response files that are part of compilation. Options stored in the configuration file apply to every compilation, while options stored in response files apply only where they are added on the command line. If you have several options in your makefile that apply to every build, you may find it easier to move these options to the configuration file (icx.cfg or icpx.cfg for C++ or dpcpp.cfg for DPC++).

In a multi-user, networked environment, options listed in the icx.cfg or icpx.cfg for C++ or dpcpp.cfg for DPC++ files are generally intended for everyone who uses the compiler. If you need a separate configuration, you can use the ICXCFG or ICPXCFG for C++ or DPCPPCFG for DPC++ environment variable to specify the name and location of your own .cfg file, such as /my_code/my_config.cfg. Anytime you instruct the compiler to use a different configuration file, the system configuration files (icx.cfg or icpx.cfg for C++ or dpcpp.cfg for DPC++) are ignored.

Using the Intel Libraries

The Intel® oneAPI DPC++/C++ Compiler supplies additional libraries that contain optimized implementations of many commonly used functions. Some of these functions are implemented using CPU dispatch. This means that different code may be executed when run on different processors.

Supplied libraries include the Intel® Math Library (libimf), the Short Vector Math Library (libsvml), libirc, as well as others. These libraries are linked in by default. Some library functions, such as sin or memset, may not require a call to the library, since the compiler may inline the code for the function.

**NOTE** The Intel Compiler Math Libraries contain performance-optimized implementations for various Intel platforms. By default, the best implementation for the underlying hardware is selected at runtime. The library dispatch of multi-threaded code may lead to apparent data races, which may be detected by certain software analysis tools. However, as long as the threads are running on cores with the same CPUID, these data races are harmless and are not a cause for concern.

Intel Math Library (libimf)

With the Intel® Compiler, the Intel Math Library, libimf, is linked by default. Some functions, such as sin, may not require a call to the library, since the compiler already knows how to compute the sin function. The Intel Math Library also includes some functions not found in the standard math library.

**NOTE** You cannot make calls to the Intel Math Library with GCC.

Many routines in the libimf library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

Short Vector Math Library (libsvml)

When vectorization is being done, the compiler may translate some calls to the libimf math library functions into calls to libsvml functions. These functions implement the same basic operations as the Intel Math Library, but operate on short vectors of operands. This results in greater efficiency. In some cases, the libsvml functions are slightly less precise than the equivalent libimf functions.

Many routines in the libimf library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

libirc
`libirc` contains optimized implementations of some commonly used string and memory functions. For example, it contains functions that are optimized versions of `memcpy` and `memset`. The compiler will automatically generate calls to these functions when it sees calls to `memcpy` and `memset`. The compiler may also transform loops that are equivalent to `memcpy` or `memset` into calls to these functions.

Many routines in the `libirc` library are more optimized for Intel® microprocessors than for non-Intel microprocessors.

### Product and Performance Information

| Performance varies by use, configuration and other factors. Learn more at [www.Intel.com/PerformanceIndex](http://www.Intel.com/PerformanceIndex). |
| Notice revision #20201201 |

### See Also

- Invoking the Compiler
  - `march`
  - `-o` compiler option
- Using Configuration Files
- Using Response Files
absolute error  
   option defining for math library function results 181
access_by 388
adding a source file 38
adding files 44
adding the compiler
   in Eclipse* 37
align
   attribute 361
align_value
   attribute 361
aligned
   attribute 361
aligned_offset 419
alloc_section 551
alternate compiler options 346
alternate tools and locations 617
ANSI/ISO standard 792
aos1d_container 377, 379, 385, 389, 392, 394–396, 420, 424–426
aos1d_container::accessor 396, 399, 400, 403, 405, 406, 408
aos1d_container::const_accessor 407
applications
   deploying 371
   option specifying code optimization for 85
ar tool 368
assembler
   option passing options to 329
assembler output file
   option specifying a dialect for 107
assembly files
   naming 35
assembly listing file
   option specifying generation of 207
Asynchronous I/O async_class methods
   clear_queue() 507
   get_error_operation_id() 506
   get_last_error() 506
   get_last_operation_id() 505
   get_status() 506
   resume_queue() 507
   stop_queue() 507
   wait() 505
Asynchronous I/O Extensions
   introduction 489
   library 490
   template class 504
Asynchronous I/O library functions
   aio_cancel() 499
   aio_error() 496
   aio_fsync() 498
   aio_read() 490
   aio_return() 497
   aio_suspend() 495
   aio_write() 491
   errno macro 503
Error Handling 503
   examples
      aio_cancel() 499
      aio_error() 497
      aio_read() 491
      aio_write() 492
      lio_listio() 502
   lio_listio() 501
Asynchronous I/O template class
   async_class 504
   thread_control 504
attribute
   align 361
      align_value 361
      aligned 361
   attribute 361
   aligned
      attribute 361
associative memory 118
associative prefetching 118
asynchronous I/O async_class methods
   clear_queue() 507
   get_error_operation_id() 506
   get_last_error() 506
   get_last_operation_id() 505
   get_status() 506
   resume_queue() 507
   stop_queue() 507
   wait() 505
asynchronous I/O Extensions
   introduction 489
   library 490
   template class 504
asynchronous I/O library functions
   aio_cancel() 499
   aio_error() 496
   aio_fsync() 498
   aio_read() 490
   aio_return() 497
   aio_suspend() 495
   aio_write() 491
   errno macro 503
Error Handling 503
   examples
      aio_cancel() 499
      aio_error() 497
      aio_read() 491
      aio_write() 492
      lio_listio() 502
   lio_listio() 501
Asynchronous I/O template class
   async_class 504
   thread_control 504
attribute
   align 361
      align_value 361
      aligned 361
   attribute 361
   aligned
      attribute 361
associative memory 118
associative prefetching 118
auto-vectorization auto-vectorization hints 737
auto-vectorization of innermost loops 358
auto-vectorizer
   AVX 700
   SSE 700
   SSE2 700
   SSE3 700
auto-vectorizer (continued)
SSSE3 700
using 705
avoid
inefficient data types 358
mixed arithmetic expressions 358

B
base platform toolset 48
bit fields and signs 796
block_loop 552
building a project
with Eclipse* 40
building multiple projects 50
building with Intel® C++ 46
builds
parallel project 50

c
C++0x
option enabling support of 261
C++11
option enabling support of 261
c99
option enabling support of 261
calling conventions 54
capturing IPO output 743
changing number of threads
summary table of 642
Class Libraries
C++ classes and SIMD operations 436
capabilities of C++ SIMD classes 439
conventions 441
floating-point vector classes
arithmetic operators 464
cacheability support operators 477
compare operators 471
conditional select operators 474
constructors and initialization 463
conversions 463
data alignment 463
debug operators 478
load operators 479
logical operators 470
minimum and maximum operators 469
move mask operators 479
notation conventions 462
overview 461
store operators 479
unpack operators 479
integer vector classes
addition operators
subtraction operators 446
assignment operator 444
clear MMX(TM) state operators 460
comparison operators 450
conditional select operators 452
conversions between fvec and ivec 460
debug operators
element access operator 453
element assignment operators 453
functions for SSE 460
ivec classes 440
logical operators 444
multiplication operators 448
debug operators (continued)
debug operators (continued)
pack operators 459
rules for operators 442
shift operators 449
unpack operators 456
Quick reference 480
syntax 441
terms 441
Classes
programming example 486
code
methods to optimize size of 749
mixing managed and unmanaged 794
option generating feature-specific 102
option generating feature-specific for Windows* OS 91
option generating for specified CPU 105
option generating specialized 122
option generating specialized and optimized 119
code layout 746
code size
methods to optimize 749
option affecting inlining 750
option disabling expansion of functions 752
option disabling loop unrolling 753
option dynamically linking libraries 751
option excluding data 752
option for certain exception handling 753
option stripping symbols 751, 754
option to avoid library references 754
using IPO 755
code_align 554
command line 27
command-line window
setting up 27
compatibility
with Microsoft* Visual Studio* 794
compilation phases 615
compilation units 749
compiler
compilation phases 615
overview 19
compiler command-line options
option recording 218
compiler differences
between Intel® C++ and Microsoft* Visual C++* 795
compiler directives
for vectorization 700, 718
compiler information
saving in your executable 621
compiler installation
option specifying root directory for 252
compiler operation
input files 28
invoking from the command line 25
compiler options
alphabetical list of 60
alternate 346
command-line syntax 31
deprecated and removed 73
for optimization 799, 804
for portability 347
for visibility 621
gcc-compatible warning 353
general rules for 79
how to display informational lists 78
linker-related 615
option categories 31
compiler options (continued)
  overview of descriptions of 80
  using 31
compiler selection
  in Visual Studio* 46
compiler setup
compilers
  using multiple versions 37
compilervars environment script 25
compilervars.bat 799
compiling
  compiling considerations 799
  gcc* code with Intel® C++ Compiler 804
compiling considerations 799
compiling large programs 745
compiling with IPO 743
concurrency_safe
  attribute 364
conditional parallel region execution
  inline expansion 748
configuration files 618
configurations
  debug and release 47
console
  option displaying information to 345
const
  attribute 364
conventions
  in the documentation 20
converting to Intel® C++ Compiler project system 794
coprocessorThread allocation on processor 641
correct usage of countable loop 713
COS
  correct usage of 713
CPU
  option generating code for specified 105
CPU time
  for inline function expansion 747
cpu_dispatch
  attribute 365
cpu_specific
  attribute 365
create libraries using IPO 746
creating
  projects 44
  a new project in Eclipse* 38

\[D\]
data format
  prefetching 741
type 700, 718
data types
  efficiency 359
DAZ flag 357
debug information
  in program database file 209
  option generating full 227
  option generating in DWARF 2 format 216
  option generating in DWARF 3 format 216
  option generating in DWARF 4 format 216
  option generating levels of 215
debugging
  option affecting information generated 203, 205
  option specifying settings to enhance 203, 205
denormal exceptions 358
denormal numbers (IEEE*)
  NaN values 359
denormals 356
deploying applications 371
deprecated compiler options 73
diagnostics 587
dialog boxes
  Intel® Performance Libraries 52
  Options: Compilers 51
  Options: Converter 53
  Options: Intel® Performance Libraries 52
  Use Intel C++ 52
difference operators 693
directory
  option adding to start of include path 239
  option specifying for executables 228
  option specifying for includes and libraries 228
directory paths
  in Microsoft Visual Studio* 48
disabling
  inlining 748
distribute_point 554
distributing applications 371
DO constructs 713
documentation
  conventions for 20
driver tool commands
  option specifying to show and execute 328
  option specifying to show but not execute 333
dual core thread affinity 668
DWARF debug information
  option creating object file containing 218
dynamic information
  threads 648, 686
dynamic linker
  option specifying an alternate 310
dynamic shared object
  option producing a 322
dynamic-link libraries (DLLs)
  option searching for unresolved references in 316
dynamic-linking of libraries
  option enabling 307

\[E\]
ebp register
  option determining use in optimizations 96
Eclipse*
  adding a source file 38
  cheat sheets 37
  creating a new project 38
  Eclipse* integration
  excluding source files from build 39
  exporting makefiles 41
  error parser 41
  including source files from build 39
  exporting makefiles in Eclipse* 41
  global symbols 620
  integration
  adding the compiler 37
  building a project 40
  cheat sheets 37
  creating a new project 38
  excluding source files from build 39
  exporting makefiles 41
  global symbols 620
  Intel® C/C++ Error Parser 41
running a project (continued)
integration (continued)
makfiles 41
multi-version compiler support 37
running a project 40
setting options 39
visibility declaration attribute 621
integration overview 36
Intel® C/C++ Error Parser 41
projects
multi-version compiler support 37
running a project 40
in Eclipse* 40
using Intel® Performance Libraries 42
visibility declaration attribute 621
Eclipse* integration
building a project 40
makfiles 41
efficiency 358
efficient
inlining 748
efficient data types 359
endian data
and OpenMP* extension routines 658
loop constructs 713
routines overriding 648, 686
using OpenMP* 693
Enter index keyword 15, 16, 30, 143, 362, 548, 622, 625, 694
enums 796
environment variables
LD_LIBRARY_PATH 370
Linux* 589
run-time 589
setting 27
setting with setvars file 23
Windows* 589
error messages 587
error parser 41
examples
aio_cancel() 499
aio_error() 497
aio_return() 497
aio_suspend() 495
lio_listio() 502
exception handling
option generating table of 95
execution environment routines 648, 686
execution mode 658
explicit vector programming
array notations 718
elemental functions 718
smid 718
extensions 627

F
feature requirements 17
feature-specific code
option generating and optimizing 119
fixed_offset 419
floating-point array operation 357
Floating-point array: Handling 357
floating-point calculations
option controlling semantics of 197
option enabling consistent results 197
floating-point exceptions
denormal exceptions 358
floating-point numbers
special values 359
floating-point operations
option controlling semantics of 197
Floating-point Operations
programming tradeoffs 354
FMA instructions
option enabling 196
forceinline 556
format function security problems
option issuing warning for 295
frame pointer
option affecting leaf functions 111
FTZ flag 357
Function annotations
__declspec(align) 737
__declspec(vector) 737
function expansion 749
function pointers
SIMD-enabled 730
function preemption 747
fused multiply-add instructions
option enabling 196
G
g++* language extensions 792
gather and scatter type vector memory references
option enabling optimization for 136
gcc C++ run-time libraries
include file path
option adding a directory to second 237
option removing standard directories from 251
gcc-compatible warning options 353
gcc* compatibility 792
gcc* considerations 804
gcc* interoperability 792
gcc* language extensions 792
general compiler directives
for inlining functions 747
for vectorization 700
global function symbols
option binding references to shared library
definitions 309
global symbols
option binding references to shared library
definitions 309
GNU C++ compatibility 792
H
help
using in Microsoft Visual Studio* 18
high performance programming
applications for 741
high-level optimizer 741
HLO 741
I
IA-32 architecture based applications
HLO 741
ICV 691
IEEE Standard for Floating-Point Arithmetic, IEEE 754-2008 359
IEEE*
IEEE* (continued)
floating-point values 359
include files 34
inline 556
inlining
compiler directed 748
developer directed 749
preemption 747
input files 28
integrating Intel® C++ with Microsoft® Visual Studio® 794
intel_omp_task 557
intel_omp_taskq 558
Intel-provided libraries
option linking dynamically 323
option linking statically 325
Intel's C++ asynchronous I/O template class
Usage Example 508
Intel's Memory Allocator Library 373
Intel's Numeric String Conversion Library
libstroconv 535, 537
Intel(R) 64 architecture based applications
HLO 741
Intel(R) IPP libraries
option letting you choose the library to link to 128
option letting you link to 131
Intel(R) libraries
option disabling linking to 319
Intel(R) linking tools 741
Intel(R) MKL
option letting you link to libraries 133
Intel(R) TBB libraries
option letting you link to 137
Intel® C/C++ Error Parser 41
Intel® C++
command-line environment 27
Intel® C++ Class Libraries
overview 435
Intel® C++ Compiler command prompt window 27
Intel® C++ Compiler extension routines 658
Intel® extension environment variables 589
Intel® IEEE 754-2008 Binary Floating-Point Conformance
Library
formatOf general-computational operations
add 518
binary32_to_binary64 518
binary64_to_binary32 518
div 518
fma 518
from_hexstring 518
from_int32 518
from_int64 518
from_string 518
from_uint32 518
from_uint64 518
mul 518
sqrt 518
sub 518
to_hexstring 518
to_int32 ceil 518
to_int32 floor 518
to_int32 int 518
to_int32 rnint 518
to_int32 xceil 518
to_int32 xfloor 518
to_int32 xint 518
to_int32 xrnint 518
to_int32 xrninta 518
signaling-computational operations (continued)
formatOf general-computational operations (continued)
to_int64 ceil 518
to_int64 floor 518
to_int64 int 518
to_int64 rnint 518
to_int64 xceil 518
to_int64 xfloor 518
to_int64 xint 518
to_int64 xrnint 518
to_int64 xrninta 518
to_string 518
to_uint32 ceil 518
to_uint32 floor 518
to_uint32 int 518
to_uint32 rnint 518
to_uint32 xrnint 518
to_uint32 xceil 518
to_uint32 xfloor 518
to_uint32 xint 518
to_uint32 xrnint 518
to_uint32 xrninta 518
to_uint64 ceil 518
to_uint64 floor 518
to_uint64 int 518
to_uint64 rnint 518
to_uint64 xceil 518
to_uint64 xfloor 518
to_uint64 xint 518
to_uint64 xrnint 518
to_uint64 xrninta 518
to_uint64 xint 518
to_uint64 xrninta 518
to_uint64 xrninta 518
to_uint64 xint 518
to_uint64 xrninta 518
to_uint64 xrninta 518
to_uint64 xrninta 518
to_uint64 xrninta 518
to_uint64 xrninta 518
to_uint64 xrninta 518
to_uint64 xrninta 518
to_uint64 xrninta 518
homogeneous general-computational operations
ilogb 516
maxnum 516
maxnum_mag 516
minnum 516
minnum_mag 516
next_down 516
next_up 516
rem 516
round_integral_exact 516
round_integral_nearest_away 516
round_integral_nearest_even 516
round_integral_negative 516
round_integral_positive 516
round_integral_zero 516
scalbn 516
non-computational operations
class 530
defaultMode 530
getBinaryRoundingDirection 530
is754version1985 530
is754version2008 530
isCanonical 530
isFinite 530
isInfinite 530
isNaN 530
isNormal 530
isSignaling 530
isSignMinus 530
isSubnormal 530
isZero 530
lowerFlags 530
radix 530
raiseFlags 530
restoreFlags 530
signaling-computational operations (continued)  
non-computational operations (continued)  
restoreModes 530  
saveFlags 530  
setBinaryRoundingDirections saveModes 530  
testFlags 530  
testSavedFlags 530  
totalOrder 530  
totalOrderMag 530  
nonhomogeneous general-computational operations 512  
quiet-computational operations  
copy 524  
copysign 524  
negate 524  
signaling-computational operations  
quiet_equal 525  
quiet_greater 525  
quiet_greater_equal 525  
quiet_greater_unordered 525  
quiet_less 525  
quiet_less_equal 525  
quiet_less_unordered 525  
quiet_not_equal 525  
quiet_not_greater 525  
quiet_not_less 525  
quiet_ordered 525  
quiet_unordered 525  
signaling_equal 525  
signaling_greater 525  
signaling_greater_equal 525  
signaling_greater_unordered 525  
signaling_less 525  
signaling_less_equal 525  
signaling_less_unordered 525  
signaling_not_equal 525  
signaling_not_greater 525  
signaling_not_less 525  
   
using the library 511  
   
Intel® Integrated Performance Primitives 49, 50  
Intel® Math Kernel Library 49, 50  
Intel® Math Library  
C99 macros  
fcclassify 790  
isfinite 790  
isgreater 790  
isgreaterequal 790  
isinf 790  
isless 790  
islessequal 790  
islessgreater 790  
isnan 790  
isnormal 790  
isunordered 790  
signbit 790  
   
Intel® Performance Libraries  
Intel® Integrated Performance Primitives (Intel® IPP) 49, 50  
Intel® Math Kernel Library (Intel® MKL) 49, 50  
Intel® Threading Building Blocks (Intel® TBB) 49, 50  
Intel® Streaming SIMD Extensions (Intel® SSE) 700  
   
intermediate files  
option saving during compilation 339  
intermediate representation (IR) 741, 743  
interoperability  
with g++* 792  
with gcc* 792  
interprocedural optimizations  
capturing intermediate output 743  
   
code layout 746  
compilation 741  
compiling 743  
considerations 745  
creating libraries 746  
issues 744  
large programs 745  
linking 741, 743  
option enabling between files 125  
option enabling for single file compilation 201  
   
overview 741  
performance 744  
using 743  
whole program analysis 741  
xiar 746  
xild 746  
xilibtool 746  
intrinsics  
about 367  
invoking Intel® C++ Compiler 25  
IR 743  
ivdep 560  
IVDEP  
effect when tuning applications 741  
   
K  
   
KMP_AFFINITY  
modifier 668  
offset 668  
permute 668  
type 668  
KMP_LIBRARY 662  
KMP_TOPOLOGY_METHOD 668  
KMP_TOPOLOGY_METHOD environment variable 668  
   
L  
   
language extensions  
g++* 792  
gcc* 792  
LD_LIBRARY_PATH 370  
level zero 697  
Level Zero 695  
LIB environment variable 370  
libgcc library  
option linking dynamically 323  
option linking statically 326  
libistrcnv Library  
Intel's Numeric String Conversion functions 537  
Numeric String Conversion 535  
Numeric String Conversion Functions 535  
libm 799  
libqkmalloc Library 373  
libraries  
-c compiler option 368  
-ffPIC compiler option 368  
-shared compiler option 368  
creating 368  
creating your own 368  
LD_LIBRARY_PATH 370  
managing 370  
OpenMP* run-time routines 648, 658, 686  
option enabling dynamic linking of 307  
option enabling static linking of 308  

816
libraries (continued)
  option letting you link to Intel(R) DAAL 130
  option letting you link to the AC data types libraries for FPGA 129
  option preventing linking with shared 324
  option preventing use of standard 318
  option printing location of system 221
  redistributing 371
  shared 368, 369
  specifying 370
  static 368
library
  option searching in specified directory for 314
  option to search for 313
Library extensions
  valarray implementation 487
library functions
  Intel extension 658
  OpenMP* run-time routines 648, 686
library math functions
  option testing errno after calls to 272
libstdc++ library
  option linking statically 327
linear_index 420
  option passing linker option to 331
  option passing options to 315
linker
  option specifying maximum times to unroll 138
linking
  option preventing use of startup files and libraries
  when 320
  option preventing use of startup files when 319
  option suppressing 202
linking debug information 622
linking tools
  xild 741, 744, 746
  xilibtool 746
  xilink 741, 744
linking tools IR 741
linking with IPO 743
Linux* compiler options
  c 35
  I 34
  o 35
  Qlocation 617
  Qoption 617
  S 35
  X 34
lock routines 648, 686
loop unrolling
  using the HLO optimizer 741
loop_count 561
loops
  constructs 713
  distribution 741
  interchange 741
  option specifying maximum times to unroll 138
  parallelization 711
  transformations 741
  vectorization 711, 736

M

macro names
  option associating with an optional value 230
macros 542, 792, 804
maintainability (continued)
  allocation 658
makefiles
  modifying 797, 802
makefiles, using 29
managed and unmanaged code 794
Math library
  Complex Functions
    cabs library function 786
    cacos library function 786
    cacosh library function 786
    carg library function 786
    casin library function 786
    csin library function 786
    csinh library function 786
    csqrt library function 786
    ctan library function 786
    ctanh library function 786
    ccos library function 786
    ccosh library function 786
    cexp library function 786
    cexp10 library function 786
    cimag library function 786
    cis library function 786
    clog library function 786
    clog2 library function 786
    clog10 library function 786
    conj library function 786
    cpow library function 786
    cproj library function 786
    creal library function 786
    csin library function 786
    csinh library function 786
    csqrt library function 786
    ctan library function 786
    ctanh library function 786
  Exponential Functions
    cbrt library function 771
    exp library function 771
    exp10 library function 771
    exp2 library function 771
    expm1 library function 771
  Hyperbolic Functions
    acosh library function 769
    asinh library function 769
    atanh library function 769
    csinh library function 769
    sinh library function 769
    sinhcsinh library function 769
    tanh library function 769
  Miscellaneous Functions
    copy sign library function 781
    fabs library function 781
    fdim library function 781
    finite library function 781
    fmax library function 781
    fmin library function 781

Index
Trigonometric Functions (continued)
  Miscellaneous Functions (continued)
  Miscellaneous Functions 781
  nextafter library function 781
  Nearest Integer Functions
    ceil library function 778
    floor library function 778
    llrint library function 778
    llround library function 778
    lrint library function 778
    lround library function 778
    modf library function 778
    nearbyint library function 778
    rint library function 778
    round library function 778
    trunc library function 778
  Remainder Functions
    fmod library function 780
    remainder library function 780
    remquo library function 780
  Special Functions
    annuity library function 775
    compound library function 775
    erf library function 775
    erfc library function 775
    gamma library function 775
    gamma_r library function 775
    j0 library function 775
    j1 library function 775
    jn library function 775
    lgamma library function 775
    lgamma_r library function 775
    tgamma library function 775
    y0 library function 775
    y1 library function 775
    yn library function 775
  Trigonometric Functions
    acos library function 765
    acosl library function 765
    asin library function 765
    asind library function 765
    atan library function 765
    atand library function 765
    atan2 library function 765
    atand2 library function 765
    cos library function 765
    cosd library function 765
    cot library function 765
    cotd library function 765
    sin library function 765
    sinc library function 765
    sincos library function 765
    sincosd library function 765
    sinl library function 765
    tgamma library function 765
    tan library function 765
    tanh library function 765

Math Library
  code examples 756
  function list
    Complex Functions 760
    Exponential Functions 760
    Hyperbolic Functions 760
    Miscellaneous Functions 760
    Nearest Integer Functions 760
    Remainder Functions 760
    Special Functions 760
    Trigonometric Functions 760
  using 756
  math library functions

math library functions (continued)
  option indicating domain for input arguments 186
  option producing consistent results 184
  option specifying a level of accuracy for 192
  memory model
    option specifying large 284
    option specifying small or medium 284
    option to use specific 284
  Message Fabric Interface (MPI) support 50
  Microsoft Visual Studio*
    getting started with 45
    Intel® Performance Libraries 49
    property pages 49
      target platform
        for projects 47
        for solutions 47
  Microsoft® Visual Studio®
    compatibility 794
    integration 794
    min_val 427
    mixing vectorizable types in a loop 700
    mock object files 743
    MPI support 50
    mpx
      attribute 366
      multiple processes
        option creating 338
      multithreading
        option to use specific 338
      MXCSR register 357

N
  noblock_loop 552
  nofusion 562
  noinline 556
  noparallel 568
  noprefetch 570
  normalized floating-point number 359
  Not-a-Number (NaN) 359
  nounroll 578
  nounroll_and_jam 579
  novector 563

O
  object files
    specifying 35
  omp simd early exit 564
  omp simdoff 576
  omp target variant dispatch
    pragma 577
  omp target variant dispatch pragma 577
  OMP_STACKSIZE environment variable 628
  Open Source tools 794
  OpenMP
    support overview 627
    openmp_version 648, 686
  OpenMP*
    advanced issues 689
    C/C++ interoperability 689
    combined construct 642
    compatibility libraries 662
    composite construct 642
    debugging 689
    environment variables 668
    examples of 693
    extensions for Intel® Compiler 658
Index
Pragmas (continued)
  Microsoft* compatible 582
  overview 550
Pragmas: Intel-specific 550
precompiled header files 795
predefined macros 542
preempting functions 747
prefetch 570
processor
  option optimizing for specific 113
processor features
  option telling which to target 119
program loops
  parallel processing model 629
programs
  option maximizing speed in 81
projects
  adding files 44
  creating 44
in Microsoft Visual Studio* 44
property pages in Microsoft Visual Studio* 49
Proxy 412, 414

R
redistributable package 371
redistributing libraries 371
references to global function symbols
  option binding to shared library definitions 309
references to global symbols
  option binding to shared library definitions 309
relative error
  option defining for math library function results 183
  option defining maximum for math library function results 191
release configuration 47
remarks
  option changing to errors 294
removed compiler options 73
report generation
  Intel® Compiler extensions 658
  OpenMP* run-time routines 648, 686
  timing 648, 686
response files 619
run-time environment variables 589
run-time performance
  improving 357
runtime dispatch
  option using in calls to math functions 189

S
SDLT
  accessors 396, 406
  example programs 427, 434
  indexes 420
  number representation 415
  proxy objects 412
  SDLT_DEBUG 426
  SDLT_INLINE 426
SDLT Layouts
  sdlt layout namespace 391
setting options
  in Eclipse* 39
setvars.bat 23
setvars.csh 23
setvars.sh 23
shared libraries 368
shared object
  option producing a dynamic 322
shared scalars 693
short vector math library
  option specifying for math library functions 194
signed infinity 359
signed zero 359
simd
  vectorization
    function annotations 572
SIMD-enabled functions
  pointers to 730
soa1d_container 383
soa1d_container::accessor 396, 399, 400, 403, 405, 406, 408
soa1d_container::const_accessor 407
  specifying file names
    for assembly files 35
    for object files 35
stack
  option specifying reserve amount 311
  stack checking routine
  option controlling threshold for call of 282
stack variables
  option initializing to NaN 213
standard directories
  option removing from include search path 251
standards conformance 792
static libraries 368
subnormal numbers 356
subroutines in the OpenMP* run-time library
  for OpenMP* 662
  supported tools 792
symbol visibility
  option specifying 278
synchronization
  parallel processing model for 629
  thread sleep time 658

T
thread affinity 668
threads 49, 50
threshold control for auto-parallelization
  OpenMP* routines for 648, 686
  reordering 700
throughput optimization
  option determining 136
  to Microsoft Visual Studio* projects 44
tools
  option passing options to 254
  option specifying directory for supporting 253
topology maps 668

U
unroll
  n 578
unroll_and_jam
  n 579
unwind information
  option determining where precision occurs 95
user functions
  dynamic libraries 648, 686
  OpenMP* 693
using 618, 619
using Intel® Performance Libraries
  in Eclipse* 42
Using OpenMP* 628
using property pages in Microsoft Visual Studio* 49

**V**

valarray implementation
  compiling code 487
  using in code 487
variables
  option placing explicitly zero-initialized in DATA section 280
  option saving always 271
vector
  pragma 580
vector copy
  non-vectorizable copy 700
  programming guidelines 700
vector pragma 580
vectorization
  compiler options 705
  compiler pragmas 705
  keywords 705
  obstacles 705
  option disabling 140
  option setting threshold for loops 141
  speed-up 705
  what is 705
Vectorization
  auto-parallelization
    reordering threshold control 700
  general compiler directives 700
  Intel® Streaming SIMD Extensions 700
  language support 737
  loop unrolling 700
  pragma 737
  pragma simd 572
  SIMD 718
  user-mandated 718
    vector copy
      non-vectorizable copy 700
      programming guidelines 700
vectorizing
  loops 713
visibility declaration attribute 621
Visual Studio*
  build configuration 47
  build options 47
  building multiple projects 50
  building parallel projects 50
  building with Intel® C++ 46
  changing directory paths 48
  compiler selection 46
  converting projects 35
  debug configuration 47
    dialog boxes
      Compilers 51
      Converter 53
      Intel® Performance Libraries 52
      Use Intel C++ 52
    Intel® Performance Libraries 50
    MPI support 50
    release configuration 47
    selecting the compiler 46
    selecting the Visual C++* compiler 46

**W**

warnings
  gcc-compatible 353
  option changing to errors 293, 294
  warnings and errors 587
  whole program analysis 741
Windows* compiler options
  Fa 35
  Fo 35
  I 34
  Qlocation 617
  Qoption 617
  X 34
  worker thread 662
  worksharing 642

**X**

xiar 744, 746
xild 741, 744, 746
xilib 746
xilibtool 746
xilink 741, 744, 746