Getting the most out of your compiler with the new Optimization Reports

Martyn Corden

Technical Consulting Engineer

Intel Corporation

Objectives of the Webinar

Learn to use the consolidated and enhanced compiler optimization report in Intel® Composer XE 2015

(part of Intel® Parallel Studio XE 2015
and Intel® Cluster Studio XE 2015):

Control the information provided

Understand what optimizations the compiler performed

Use the information in the report to guide further tuning for improved performance

Why a new Optimization Report?

We heard your feedback……

- Vectorization and other reports were hard to understand
- Messages too cryptic
- Loop optimization messages split between multiple reports
- Confusion due to multiple versions of one source loop created by the compiler
- Hard to relate messages to source when code was inlined
- One huge report stream hard to navigate; unsuitable for parallel builds

Report Goals for the 15.0 Compiler

Improve the user experience

- Make the report easier to read and understand
  - A single, unified report
  - Loop-based reporting
- Focus on user actionable information
- Make it easy to select the desired information
- Expand the range of output modes

Make the report's information accessible

- In a text file
- In the assembly listing
- Through the Microsoft* Visual Studio* IDE
- Through other Intel software tools

General

Applicable to Intel® Compiler version 15.0 (currently in beta)

- for C, C++ and Fortran
- for Windows*, Linux* and OS X*

(For readability, options may not be repeated for each OS where spellings are similar. Options apply to all three OS unless otherwise stated.)

Main switches:

/Qopt-report[:N] (Windows), -opt-report[=N] (Linux and OS X)

N = 1-5 for increasing levels of detail, (default N=2)

/Qopt-report-phase:str[,str1,...]

str = loop, par, vec, openmp, ipo, pgo, cg, offload, tcollect, all

/Qopt-report-file:stdout | stderr | filename

Report Output

Output goes to a text **file** by default, no longer stderr

- File extension is .optrpt, root is same as object file
- One report file per object file, in object directory
- created from scratch or overwritten (no appending)

/Qopt-report-file:stderr gives old behavior (everything to stderr)

:filename to change default file name

/Qopt-report-format:vs format suitable for Visual Studio* IDE

For debug builds, (-g on Linux* or OS X*, /Zi on Windows*), assembly code and object files contain loop optimization info

- /Qopt-report-embed to enable this for non-debug builds

Filtering Report Output

The optimization report can be large

Filtering can restrict the content to the most performance-critical parts of an application

/Qopt-report-routine:<function1>[,<function2>,...]

"function1" can be a substring of function name
or a regular expression

can also restrict to a particular range of line numbers, e.g.:

icl /Qopt-report-filter="test.cpp,100-300" test.cpp

Also select the optimization phase(s) of interest with -opt-report-phase

Loop, Vectorization and Parallelization Phases

Hierarchical display of loop nest

- Easier to read and understand
- For loops for which the compiler generates multiple versions, each version gets its own set of messages

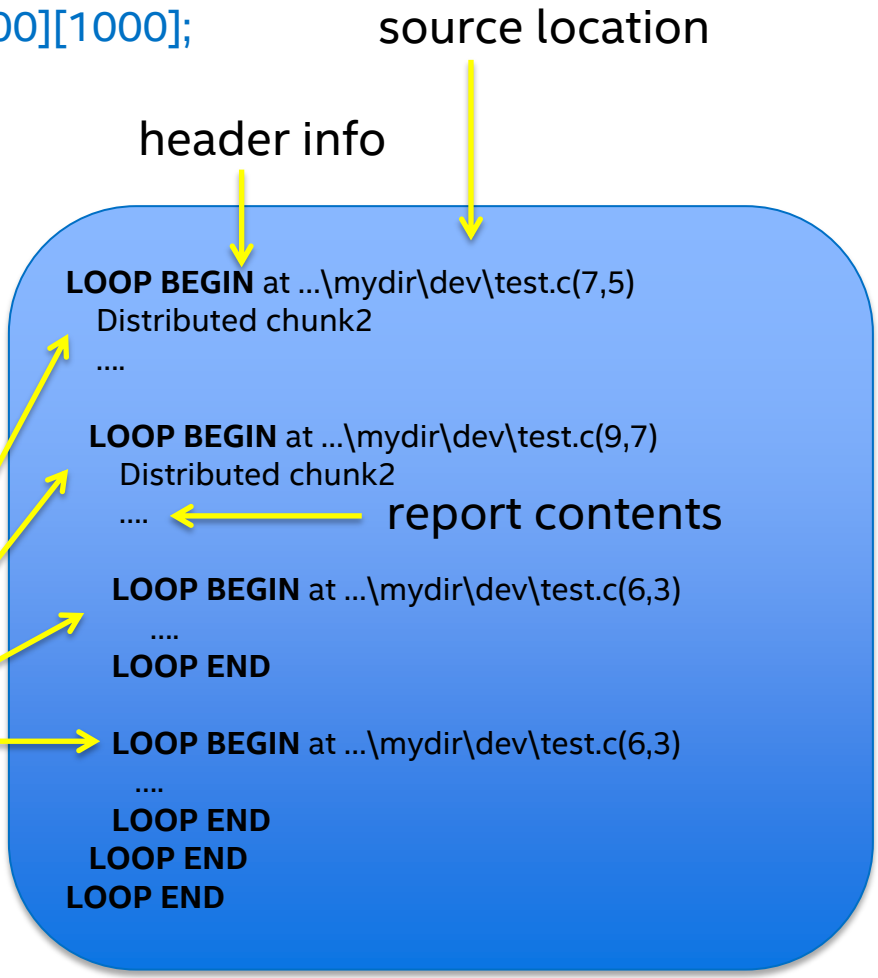
Where code has been inlined, caller/callee info available

The “Loop” (formerly hlo) phase includes messages about memory and cache optimizations, such as blocking, unrolling and prefetching

- Now integrated with vectorization & parallelization reports

Hierarchically Presented Loop Optimization Report

```
1  double a[1000][1000],b[1000][1000],c[1000][1000];
2
3  void foo() {
4    int i,j,k;
5
6    for( i=0; i<1000; i++) {
7      for( j=0; j< 1000; j++) {
8        c[j][i] = 0.0;
9        for( k=0; k<1000; k++) {
10         c[j][i] = c[j][i] + a[k][i] * b[j][k];
11       }
12     }
13   }
14 }
```



14.0 vs 15.0 loop optimization reports

```
HPO VECTORIZER REPORT (foo) LOG OPENED ON Mon Feb 24 12:10:15 2014

<d:\dev\test.c;-1:-1;hpo_vectorization;foo;0>
HPO Vectorizer Report (foo)

d:\dev\test.c(7:5-7:5):VEC:foo:  loop was not vectorized:
loop was transformed to memset or memcpy
d:\dev\test.c(6:3-6:3):VEC:foo:  PERMUTED LOOP WAS VECTORIZED
d:\dev\test.c(9:7-9:7):VEC:foo:  loop was not vectorized: not inner loop
d:\dev\test.c(7:5-7:5):VEC:foo:  loop was not vectorized: not inner loop

HLO REPORT LOG OPENED ON Mon Feb 24 12:10:15 2014

<test.c;-1:-1;hlo;foo;0>
High Level Optimizer Report (foo)

<d:\dev\test.c;7:7;hlo_distribution;in foo;0>
LOOP DISTRIBUTION in foo at line 7

<d:\dev\test.c;6:6;hlo_linear_trans;foo;0>
LOOP INTERCHANGE in loops at line: 6 7
Loopnest permutation ( 1 2 ) --> ( 2 1 )
LOOP INTERCHANGE in loops at line: 6 7 9
Loopnest permutation ( 1 2 3 ) --> ( 2 3 1 )

<d:\dev\test.c;7:7;hlo_reroll;foo;0>
Loop at line:7       memset generated

Block, Unroll, Jam Report:
(loop line numbers, unroll factors and type of transformation)

<d:\dev\test.c;6:6;hlo_unroll;foo;0>
Loop at line 6 completely unrolled by 8

Loop Collapsing Report:

<d:\dev\test.c;7:7;hlo_loop_collapsing;foo;0>
Loops at line:7     and line:6     collapsed
```

```
 Report from: Loop nest, Vector & Auto-parallelization optimizations   [loop,
vec, par]

LOOP BEGIN at d:\dev\test.c(7,5)
Distributed chunk1
  remark #25430: LOOP DISTRIBUTION (2 way)
  remark #25448: Loopnest Interchanged : ( 1 2 ) --> ( 2 1 )
  remark #25424: Collapsed with loop at line 6
  remark #25412: memset generated
  remark #15144: loop was not vectorized: loop was transformed to memset
or memcpy
LOOP END

LOOP BEGIN at d:\dev\test.c(7,5)
Distributed chunk2
  remark #25448: Loopnest Interchanged : ( 1 2 3 ) --> ( 2 3 1 )
  remark #15018: loop was not vectorized: not inner loop

  LOOP BEGIN at d:\dev\test.c(9,7)
  Distributed chunk2
    remark #15018: loop was not vectorized: not inner loop

    LOOP BEGIN at d:\dev\test.c(6,3)
      remark #15145: vectorization support: unroll factor set to 4
      remark #15003: PERMUTED LOOP WAS VECTORIZED
    LOOP END

    LOOP BEGIN at d:\dev\test.c(6,3)
      remark #15003: REMAINDER LOOP WAS VECTORIZED
    LOOP END
  LOOP END
LOOP END
```

Ordered Reporting of Transformations

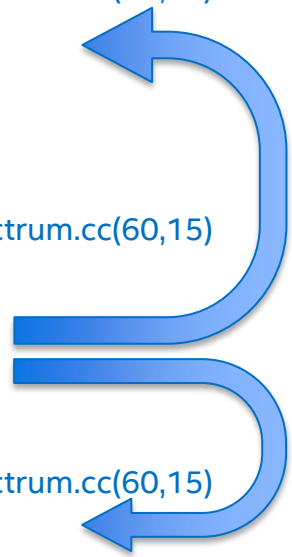
Multi-versioned loops: Peel loop, remainder loop and kernel

```
LOOP BEGIN at ggFineSpectrum.cc(124,5) inlined into ggFineSpectrum.cc(56,7)
   remark #15018: loop was not vectorized: not inner loop

   LOOP BEGIN at ggFineSpectrum.cc(138,5) inlined into ggFineSpectrum.cc(60,15)
   Peeled
      remark #25460: Loop was not optimized
   LOOP END

   LOOP BEGIN at ggFineSpectrum.cc(138,5) inlined into ggFineSpectrum.cc(60,15)
      remark #15145: vectorization support: unroll factor set to 4
      remark #15002: LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at ggFineSpectrum.cc(138,5) inlined into ggFineSpectrum.cc(60,15)
   Remainder
      remark #15003: REMAINDER LOOP WAS VECTORIZED
   LOOP END
LOOP END
```



Vectorized with Peeling and Remainder

Annotated Assembly Listings

```
.L11:        # optimization report
             # LOOP WAS INTERCHANGED
             # loop was not vectorized: not inner loop
      xorl     %edi, %edi                          #38.3
      movsd    b.279.0.2(%rax,%rsi,8), %xmm0       #41.32
      unpcklpd %xmm0, %xmm0                        #41.32
                    # LOE rax rcx rbx rsi rdi r12 r13 r14 r15 edx xmm0
..B1.11:              # Preds ..B1.11 ..B1.10
..L12:       # optimization report
             # LOOP WAS INTERCHANGED
             # LOOP WAS VECTORIZED
             # VECTORIZATION HAS UNALIGNED MEMORY REFERENCES
             # VECTORIZATION SPEEDUP COEFFECIENT 2.250000
      movaps   a.279.0.2(%rcx,%rdi,8), %xmm1       #41.22
      movaps   16+a.279.0.2(%rcx,%rdi,8), %xmm2    #41.22
      movaps   32+a.279.0.2(%rcx,%rdi,8), %xmm3    #41.22
      movaps   48+a.279.0.2(%rcx,%rdi,8), %xmm4    #41.22
      mulpd    %xmm0, %xmm1                        #41.32
      mulpd    %xmm0, %xmm2                        #41.32
<...>
```

```
L4::         ; optimization report
             ; PEELED LOOP FOR VECTORIZATION
$LN36:
$LN37:
      vaddss   xmm1, xmm0, DWORD PTR [r8+r10*4]    ;4.5

snip snip snip

L5::         ; optimization report
             ; LOOP WAS VECTORIZED
             ; VECTORIZATION HAS UNALIGNED MEMORY REFERENCES
             ; VECTORIZATION SPEEDUP COEFFECIENT 8.398438
$LN46:
      vaddps   ymm1, ymm0, YMMWORD PTR [r8+r9*4]   ;4.5

snip snip snip

L6::         ; optimization report
             ; LOOP WAS VECTORIZED
             ; REMAINDER LOOP FOR VECTORIATION
             ; VECTORIZATION HAS UNALIGNED MEMORY REFERENCES
             ; VECTORIZATION SPEEDUP COEFFECIENT 2.449219
$LN78:
      add      r10, 4                              ;3.3

snip snip snip

L7::         ; optimization report
             ; REMAINDER LOOP FOR VECTORIATION
$LN93:
      inc      rax                                 ;3.3
```

Vectorization – report levels

/Qopt-report-phase:vec /Qopt-report:N

N specifies the level of detail; default N=2 if N omitted

Level 0: No vectorization report

Level 1: Reports when vectorization has occurred.

Level 2: Adds diagnostics why vectorization did not occur.

Level 3: Adds vectorization loop summary diagnostics.

Level 4: Additional detail, e.g. on data alignment

Level 5: Adds detailed data dependency information

Actionable Messages Example (1)

```
$ icc -c -opt-report=4 -opt-report-phase=loop,vec -opt-report-file=stderr foo.c

Begin optimization report for: foo

    Report from: Loop nest & Vector optimizations [loop, vec]

LOOP BEGIN at foo.c(4,3)
Multiversioned v1
   remark #25231: Loop multiversioned for Data Dependence
   remark #15135: vectorization support: reference theta has unaligned access
   remark #15135: vectorization support: reference sth has unaligned access
   remark #15127: vectorization support: unaligned access used inside loop body
   remark #15145: vectorization support: unroll factor set to 2
   remark #15164: vectorization support: number of FP up converts: single to double precision 1
   remark #15165: vectorization support: number of FP down converts: double to single precision 1
   remark #15002: LOOP WAS VECTORIZED
   remark #36066: unmasked unaligned unit stride loads: 1
   remark #36067: unmasked unaligned unit stride stores: 1
   ....   (loop cost summary)  ....
   remark #25018: Estimate of max trip count of loop=32
LOOP END

LOOP BEGIN at foo.c(4,3)
Multiversioned v2
   remark #15006: loop was not vectorized: non-vectorizable loop instance from multiversioning
LOOP END
================================================================
```

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 128; i++)
      sth[i] = sin(theta[i]+3.1415927);
}
```

Actionable Messages: Example (2)

$ icc -c -opt-report=4 -opt-report-phase=loop,vec -opt-report-file=stderr **-fargument-noalias** foo.c
Begin optimization report for: foo (/Qalias-args- on Windows*)
Report from: Loop nest & Vector optimizations [loop, vec]

LOOP BEGIN at foo.c(4,3)
remark #15135: vectorization support: reference theta has unaligned access
remark #15135: vectorization support: reference sth has unaligned access
remark #15127: vectorization support: unaligned access used inside loop body
remark #15145: vectorization support: unroll factor set to 2
remark #15164: vectorization support: number of **FP up converts: single to double precision 1**
remark #15165: vectorization support: number of **FP down converts: double to single precision 1**
remark #15002: LOOP WAS VECTORIZED
remark #36066: unmasked unaligned unit stride loads: 1
remark #36067: unmasked unaligned unit stride stores: 1
remark #36091: --- begin **vector loop cost summary** ---
remark #36092: **scalar loop cost: 114**
remark #36093: **vector loop cost: 55.750**
remark #36094: **estimated potential speedup: 2.040**
remark #36095: lightweight vector operations: 10
remark #36096: medium-overhead vector operations: 1
remark #36098: vectorized math library calls: 1
remark #36103: **type converts: 2**
remark #36104: --- end vector loop cost summary ---
remark #25018: Estimate of max trip count of loop=32
LOOP END

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 128; i++)
      sth[i] = sin(theta[i]+3.1415927);
}
```

Actionable Messages: Example (3)

$ icc -c -opt-report=4 -opt-report-phase=loop,vec -opt-report-file=stderr -fargument-noalias foo.c

Begin optimization report for: foo

Report from: Loop nest & Vector optimizations [loop, vec]

LOOP BEGIN at foo.c(4,3)
remark #15135: vectorization support: reference theta has unaligned access
remark #15135: vectorization support: reference sth has unaligned access
remark #15127: vectorization support: unaligned access used inside loop body
remark #15002: LOOP WAS VECTORIZED
remark #36066: unmasked unaligned unit stride loads: 1
remark #36067: unmasked unaligned unit stride stores: 1
remark #36091: --- begin vector loop cost summary ---
remark #36092: scalar loop cost: 111
remark #36093: vector loop cost: 28.000
remark #36094: **estimated potential speedup: 3.950**
remark #36095: lightweight vector operations: 9
remark #36098: vectorized math library calls: 1
remark #36104: --- end vector loop cost summary ---
remark #25018: **Estimate of max trip count of loop=32**
LOOP END

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 128; i++)
    sth[i] = sinf(theta[i]+3.1415927f);
}
```

Actionable Messages: Example (4)

$ icc -c -opt-report=4 -opt-report-phase=loop,vec -opt-report-file=stderr -fargument-noalias **-xavx** foo.c

Begin optimization report for: foo

Report from: Loop nest & Vector optimizations [loop, vec]

LOOP BEGIN at foo.c(4,3)
remark #15135: vectorization support: **reference theta has unaligned access**
remark #15135: vectorization support: **reference sth has unaligned access**
remark #15127: vectorization support: **unaligned access used inside loop body**
remark #15002: LOOP WAS VECTORIZED
remark #36066: **unmasked unaligned unit stride loads: 1**
remark #36067: **unmasked unaligned unit stride stores: 1**
remark #36091: --- begin vector loop cost summary ---
remark #36092: scalar loop cost: 110
remark #36093: vector loop cost: 15.370
remark #36094: estimated potential speedup: **7.120**
remark #36095: lightweight vector operations: 9
remark #36098: vectorized math library calls: 1
remark #36104: --- end vector loop cost summary ---
remark #25018: Estimate of **max trip count of loop=16**
LOOP END

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 128; i++)
     sth[i] = sinf(theta[i]+3.1415927f);
}
```

Actionable Messages: Example (5)

$ icc -c -opt-report=4 -opt-report-phase=loop,vec -opt-report-file=stderr -fargument-noalias -xavx foo.c

Begin optimization report for: foo

Report from: Loop nest & Vector optimizations [loop, vec]

LOOP BEGIN at foo.c(6,3)
remark #15134: vectorization support: **reference theta has aligned access**
remark #15134: vectorization support: **reference sth has aligned access**
remark #15002: LOOP WAS VECTORIZED
remark #36064: **unmasked aligned unit stride loads: 1**
remark #36065: **unmasked aligned unit stride stores: 1**
remark #36091: --- begin vector loop cost summary ---
remark #36092: scalar loop cost: 110
remark #36093: vector loop cost: 13.620
remark #36094: estimated potential speedup: **8.060**
remark #36095: lightweight vector operations: 9
remark #36098: vectorized math library calls: 1
remark #36104: --- end vector loop cost summary ---
remark #25018: Estimate of max trip count of loop=16
LOOP END

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  __assume_aligned(theta,32);
  __assume_aligned(sth,32);
  for (i = 0; i < 128; i++)
    sth[i] = sinf(theta[i]+3.1415927f);
}
```

==========

Actionable Messages: Example (6)

$ icc -c -opt-report=4 -opt-report-phase=loop,vec -opt-report-file=stderr -fargument-noalias -xavx foo.c

Begin optimization report for: foo

Report from: Loop nest & Vector optimizations [loop, vec]

LOOP BEGIN at foo.c(7,3)
remark #15134: vectorization support: reference theta has aligned access
remark #15134: vectorization support: reference sth has aligned access
remark #15002: LOOP WAS VECTORIZED
remark #36064: unmasked aligned unit stride loads: 1
remark #36065: unmasked aligned unit stride stores: 1
remark #36083: **unmasked aligned streaming stores: 1**
remark #36091: --- begin vector loop cost summary ---
remark #36092: scalar loop cost: 110
remark #36093: vector loop cost: 13.620
remark #36094: estimated potential speedup: 8.070
remark #36095: lightweight vector operations: 9
remark #36098: vectorized math library calls: 1
remark #36104: --- end vector loop cost summary ---
remark #25018: Estimate of max trip count of loop=250000
remark #15158: **vectorization support: streaming store was generated for sth**
LOOP END
==========

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  __assume_aligned(theta,32);
  __assume_aligned(sth,32);
for (i = 0; i < 2000000; i++)
    sth[i] = sinf(theta[i]+3.1415927f);
}
```

Example Code for IPO Report

```
1   #include <stdio.h>
2
3   static void __attribute__((noinline)) bar
      (float a[100][100], float b[100][100]) {
4    int i, j;
5    for (i = 0; i < 100; i++) {
6        for (j = 0; j < 100; j++) {
7          a[i][j] = a[i][j] + 2 * i;
8          b[i][j] = b[i][j] + 4 * j;
9      }
10    }
11  }
12
13  static void foo(float a[100][100],
                    float b[100][100])  {
14  int i, j;
15   for (i = 0; i < 100; i++) {
16     for (j = 0; j < 100; j++) {
17       a[i][j] = 2 * i;
18         b[i][j] = 4 * j;
19     }
20   }
21   bar(a, b);
22  }
23
```

```
24  extern int main() {
25  int i, j;
26   float a[100][100];
27    float b[100][100];
28
29  for (i = 0; i < 100; i++) {
30      for (j = 0; j < 100; j++) {
31        a[i][j] = i + j;
32         b[i][j] = i - j;
33      }
34   }
35   foo(a, b);
36   foo(a, b);
37   fprintf(stderr, "%d %d\n",
                  a[99][9], b[1]99]);
38  }
```

Compiled with:

```
icc  -opt-report=3
      –opt-report-phase=ipo   sm.c
```

Features of the IPO Report – Inlining

-opt-report-phase=ipo –opt-report=3

Settings that control the amount of inlining allowed

Report for function main at line 24 of source file sm.c

foo() is inlined at lines 35 & 36

bar() called from foo at line 21 but not inlined into main

External function fprintf

User function bar() at line 3 has no function calls

Static function foo() at line 13 is dead if all calls to it are inlined

```
INLINING OPTION VALUES:
  -inline-factor: 100
...

INLINE REPORT: (main) [1]  sm.c(24,19)
  -> INLINE: [35] foo()
    -> [21] bar()
  -> INLINE: [36] foo()
    -> [21] bar()
   ->EXTERN: [37] fprintf

INLINE REPORT: (bar) [2] sm(3,81)

DEAD STATIC FUNCTION: (foo) sm.c(13,55)
```

Features of the IPO Report – more detail

-opt-report-phase=ipo –opt-report=4

Whole Program Optimization report

```
WHOLE PROGRAM (SAFE)
   [EITHER METHOD]: true
 WHOLE PROGRAM (SEEN)
   [TABLE METHOD]: true
 WHOLE PROGRAM (READ)
   OBJECT READER METHOD]: false
```

% of total routines compiled so far

sz = Size of each inlineable routine in intermediate language units (total = (stmts + exprs))

isz = Increase in size of caller due to inlining

Reasons routines were not inlined

```
INLINE REPORT: (main) [1/3=33.3%] sm.c(24,19)
 -> INLINE: [35] foo (isz = 40) (sz = 47 (25+22))
  -> [21] bar() (isz = 47) (sz = 54 (24+30))
     [[ Called routine is noinline ]]
 -> INLINE: [35] foo (isz = 40) (sz = 47 (25+22))
  -> [21] bar() (isz = 47) (sz = 54 (24+30))
     [[ Called routine is noinline ]]
 -> EXTERN: [37] fprintf
```

Offload Report for Intel® Xeon Phi™ coprocessors

Compile with –opt-report-phase=offload

Separate reports are generated for host and coprocessor

Reports for offloads using Intel® Cilk™ Plus keywords and also for offloads using Intel or OpenMP 4.0 pragmas or directives

Example for OpenMP 4.0 offload pragma:

icc –c -openmp -opt-report-phase=offload offload_test.c

Offload Report – Example with OpenMP

```
01 #pragma omp declare target
02 int compute(int i)  { return i++; }
03 #pragma omp end declare target
04
05 int do_offload() {
06    int i = 0;
07 #pragma omp target map(tofrom:i)
08      { i = compute(i); }
09    return i;
10 }
```

Host Report

```
offload_test.c(6-6):OFFLOAD:do_offload:  Offload to target MIC 1
 Data sent from host to target
       i, scalar size 4 bytes
 Data received by host from target
       i, scalar size 4 bytes
```

Coprocessor Report

```
offload_test.c(6-6):OFFLOAD:do_offload:  Outlined offload region
 Data received by target from host
       i, scalar size 4 bytes
 Data sent from target to host
       i, scalar size 4 bytes
```

Reports On Other Optimization Phases

-opt-report-phase=

par	auto-parallelization report, structured similarly to vectorization report
openmp	report on OpenMP constructs merged into the loop report
pgo	report on Profile Guided Optimization, including which functions had useful profiles
cg	optimizations during code generation, such as intrinsic function lowering
loop	additional loop and memory optimizations, such as cache blocking, prefetching, scalar replacement, etc.
tcollect	data collection for Intel® Trace Analyzer

Mapping old switches to new

-vec-report, -par-report and -openmp-report are deprecated.

They do not give the same output as for the version 14 compiler.

Instead, they are mapped to the closest equivalent phase and level of the new optimization report. Reports are not written to stderr unless you set –opt-report-file=stderr or put this into your configuration file.

Users are encouraged to convert do the new, more powerful switches. You may want to delete *.optrpt files in the ”clean” section of your makefiles.

Reports in Microsoft* Visual Studio*

The new optimization report will be accessible through the IDE in an update to the 15.0 beta compiler. Build with
/Qopt-report-format:vs

There will be an Optimization Diagnostics section under Diagnostics in Project Properties, and an Optimization Reports section under Tools/Options/Intel Composer XE

Optimization notes in the text editor will allow navigation to caller or callee site by clicking

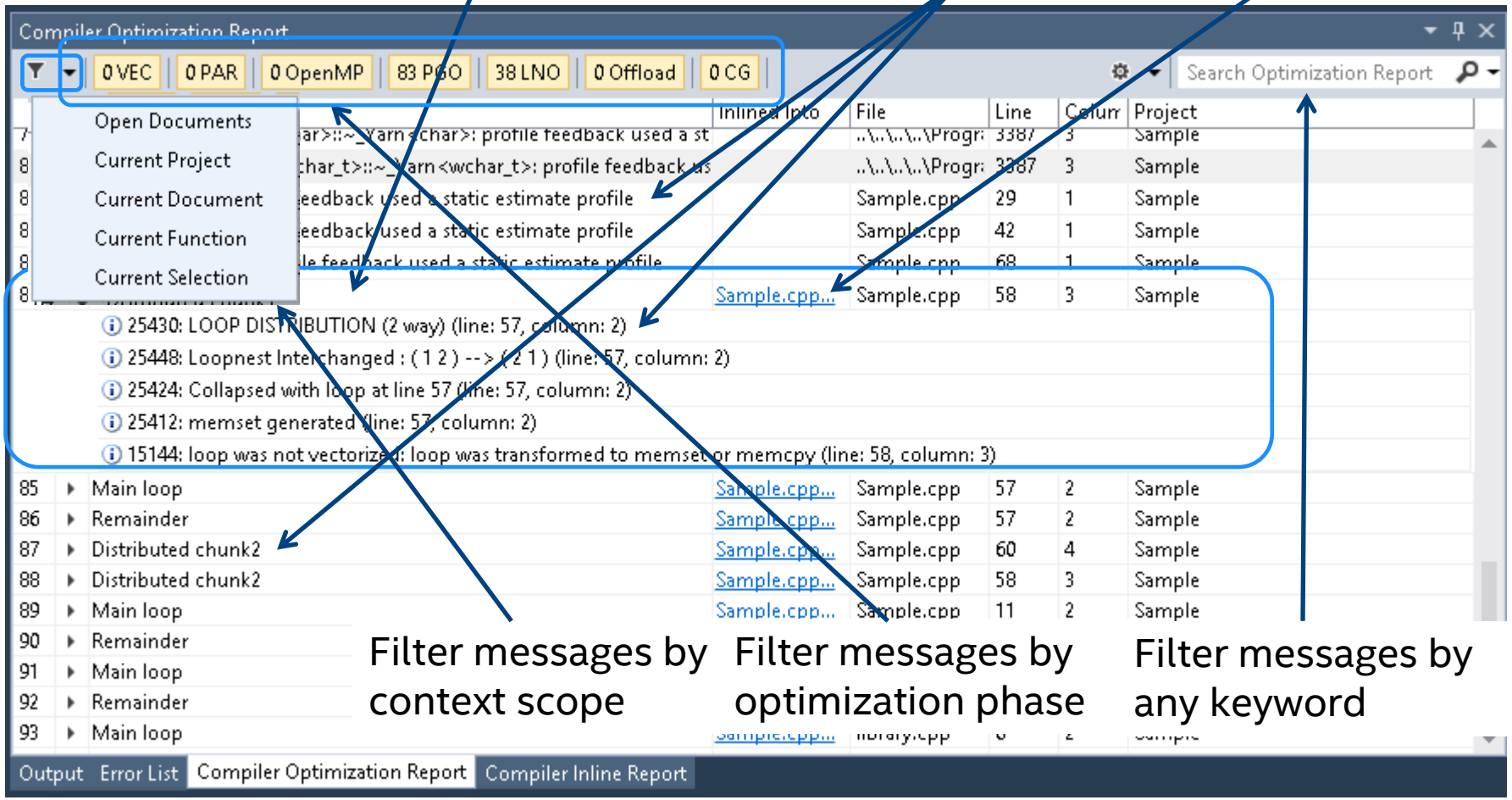
A Report Tool window will display and allow filtering of the optimization report, with direct navigation to referenced source lines by double clicking.

Compiler Optimization Report Tool Window

Hierarchically Presented Report

Jump to source position by double click

Jump to call site by click



Filter messages by context scope

Filter messages by optimization phase

Filter messages by any keyword

Additional Resources

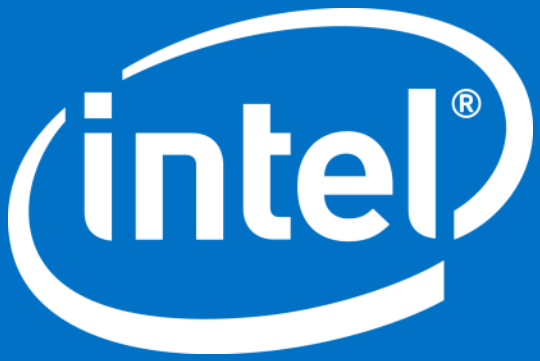
Intel® Composer XE 2015 Compiler User and Reference Guides

Intel Premier Support https://premier.intel.com

You may sign up for the Intel® Software Development Tools 2015 beta test program at
https://softwareproductsurvey.intel.com/survey/150347/2afa/

Additional information about the beta program and about future beta webinars is available at

https://software.intel.com/en-us/articles/intel-software-development-tools-2015-beta


intel®

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804