

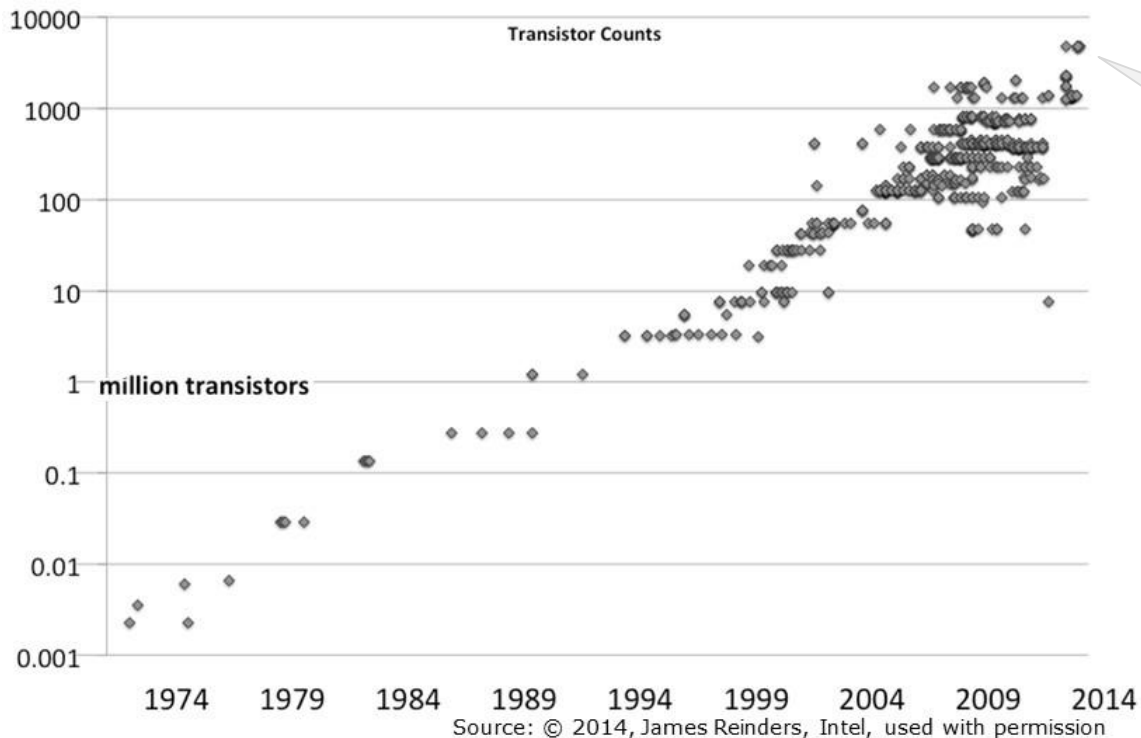


VECTORIZE OR PERFORMANCE DIES: TUNE FOR THE LATEST AVX SIMD

Kevin O'Leary, Intel Technical Consulting Engineer

Moore's Law Is Going Strong

Hardware performance continues to grow exponentially

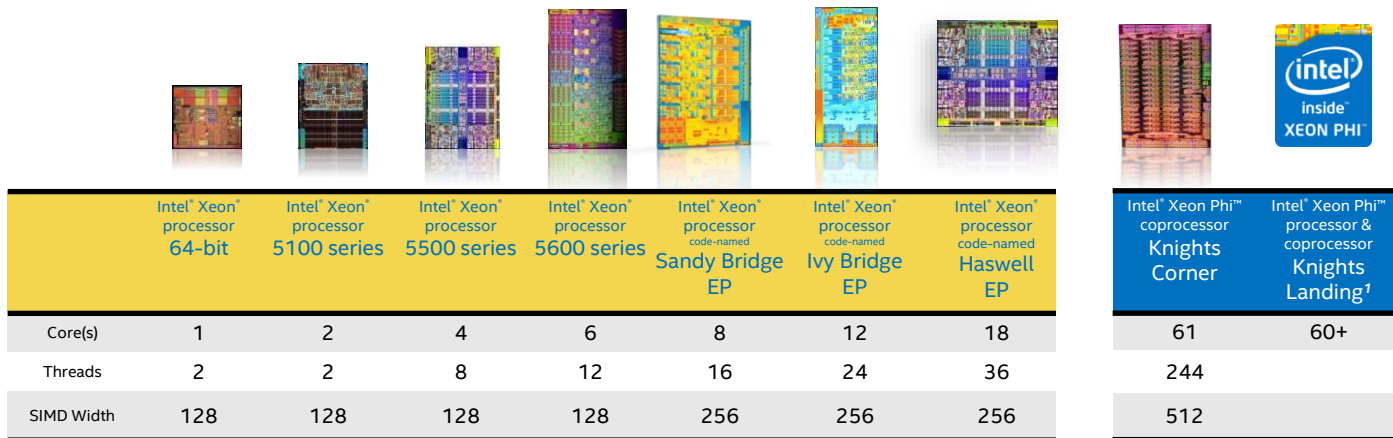


"We think we can continue Moore's Law for at least another 10 years."

Intel Senior Fellow Mark Bohr, 2015

Changing Hardware Impacts Software

More cores → More Threads → Wider vectors



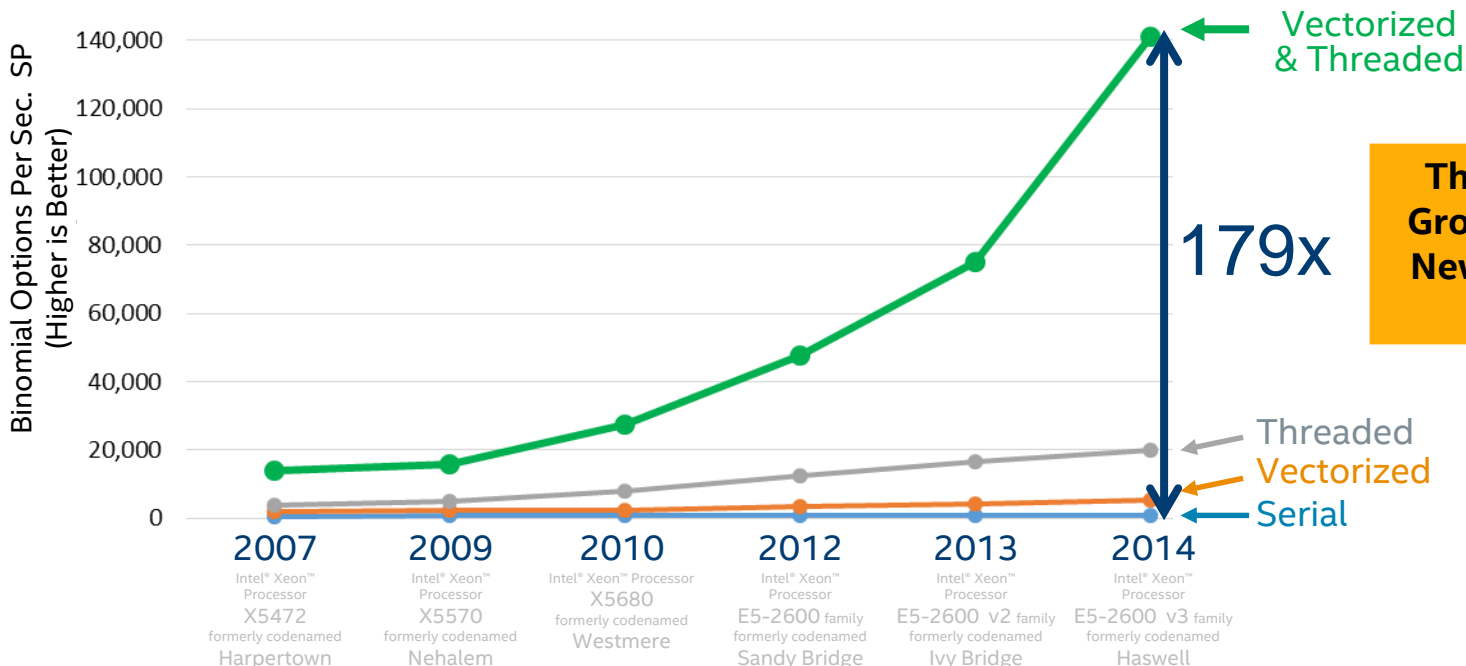
*Product specification for launched and shipped products available on ark.intel.com. 1. Not launched or in planning.

High performance software must be both:

- Parallel (multi-thread, multi-process)
- Vectorized

Vectorize & Thread or Performance Dies

Threaded + Vectorized can be much faster than either one alone



The Difference Is Growing With Each New Generation of Hardware

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. For more information go to <http://www.intel.com/performance>

[Configurations for Binomial Options SP](#)
at the end of this presentation

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.





INTEL[®] ADVISOR VECTORIZATION OPTIMIZATION AND THREAD PROTOTYPING

Software Must Vectorize & Thread or Performance Dies

True today – More true tomorrow – Difference can be substantial!

Vectorization - Have you:

- Recompiled for AVX2 or AVX512 with little gain
- Wondered where to vectorize?
- Recoded intrinsics for new arch.?
- Struggled with compiler reports?

New!

“Intel® Advisor’s Vectorization Advisor fills a gap in code performance analysis. It can guide the informed user to better exploit the vector capabilities of modern processors and coprocessors.”

Dr. Luigi Iapichino
Scientific Computing Expert
Leibniz Supercomputing Centre

Threading - Have you:

- Threaded an app, but seen little benefit?
- Hit a “scalability barrier”?
- Delayed release due to sync. errors?

“Intel® Advisor has allowed us to quickly prototype ideas for parallelism, saving developer time and effort”

Simon Hammond
Senior Technical Staff
Sandia National Laboratories

Here is What Will Be Covered

Overview

5 steps to Efficient Vectorization

Vectorization Efficiency

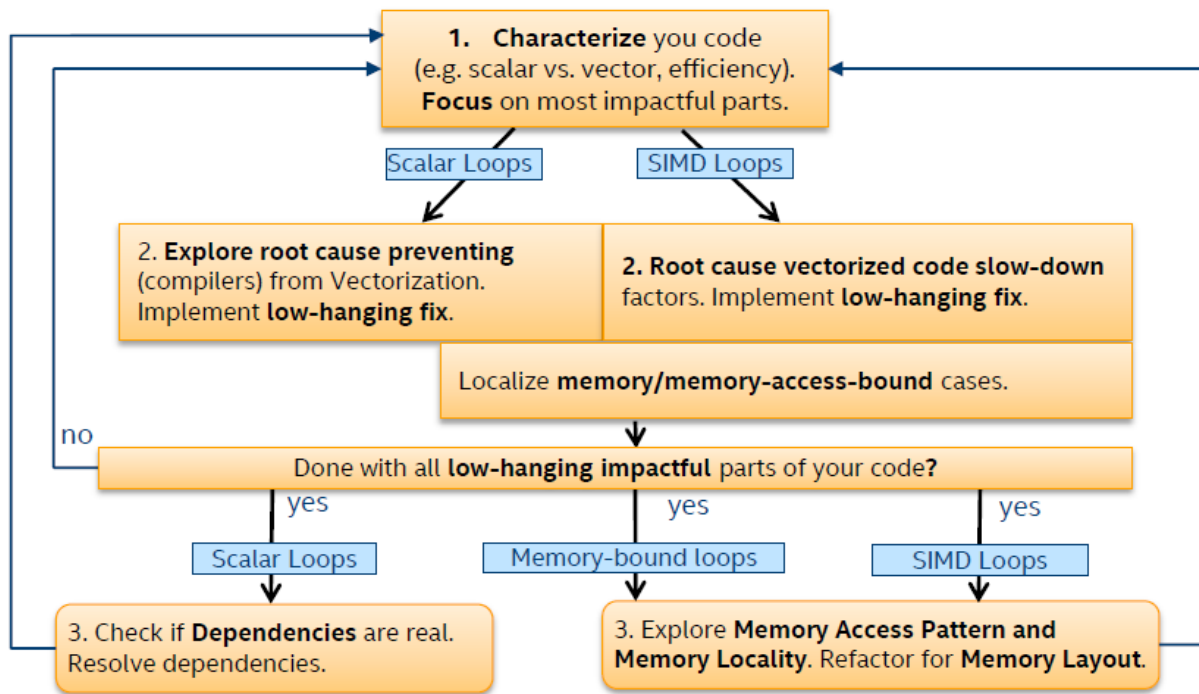
Background on Vectorization

New Features

Call to Action

Intel® Advisor helps you increase performance!

Recommended methodology



Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



The Right Data At Your Fingertips

Get all the data you need for high impact vectorization

Filter by which loops are vectorized!

What prevents vectorization?

Where should I add vectorization and/or threading parallelism? Intel Advisor XE 2016

Elapsed time: 9.49s Vectorized Not Vectorized FILTER: All Modules All Sources Loops All Threads

Summary Survey Report Refinement Reports Annotation Report

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops			
						Vect...	Efficiency	Gain...	VL (...)
[loop in matvec at Multiply.c:72]	1 Inefficient ...	5.281s	5.281s	Vectorized (Bo...		AVX	~55%	2.19x	4
[loop in matvec at Multiply.c:66]	1 Ineffective ...	2.828s	2.828s	Vectorized (Bo...		AVX	~91%	3.65x	4
[loop in matvec at Multiply.c:49]		0.531s	5.812s	Scalar					
[loop in matvec at Multiply.c:49]		0.516s	3.344s	Scalar					
[loop in matvec at Multiply.c:85]	1 Assumed d...	0.063s	0.063s	Scalar	dependence...				
[loop in main at driver.c:1511]		0.016s	0.207s	Scalar	loop with function				
[loop in ...]		0s							

Focus on hot loops

What vectorization issues do I have?

Which Vector instructions are being used?

How efficient is the code?

Get Fast Code Fast!

5 Steps to Efficient Vectorization - Vector Advisor

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops*	Self Time	Total Time	Compiler Vectorization
			Loop Type Why No Vectorization?
@ [loop in runCforallLambdaLoops]	0.094s	0.094s	Scalar vector dependence prevents vector...
@ [loop in runCforallLambdaLoops]	0.140s	3.744s	Scalar inner loop was already vectorized
✓ [loop in std::complex_base<double,struct_C_double_complex>::i...]	0.031s	0.031s	Vectorized (Body)

Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations
Peeled loop; loop stats were reordered

2. Guidance: detect problem and recommend how to fix it

Issue: Peeled/Remainder loop(s) present

All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at [Vector Essentials, Utilizing Full Vectors...](#)

Recommendation: Align memory access
Projected maximum performance gain: High
Projection confidence: Medium

The compiler created a peeled loop because one of the memory accesses in the source loop does not start at a data boundary. Align the memory access and tell the compiler your memory access is aligned.

3. "Accurate" Trip Counts: understand parallelism granularity and overheads

Total Time	Trip Counts	Iteration Duration	Call Count
	Median	Min	Max
3.151s	1	1	1
0.440s	1	1	1
0.010s	1	1	2
0.010s	1	2	1
0.010s	1	3	5

4. Loop-Carried Depen

Problems and Messages

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2	Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3	Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4	Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5	Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P6	Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7	Write after read dependency	site2	dqtest2.cpp, idle.h	dqtest2	New

Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.coc1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.coc622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.coc925	No information available	100% / 0% / 0%	All unit strides

ID	Stride	Type	Source	Modules	Alignment
@ P22	0; 0; 1	Unit stride	runCRawLoops.coc637	lcal.exe	
@ P23	0; 0	Unit stride	runCRawLoops.coc638	lcal.exe	
@ P30	-1575; -63; -26; -25; -1; 0; 1; 25; 63; 2164801	Variable stride	runCRawLoops.coc628	lcal.exe	

```

635 j2 = ( j2 + 64-1 ) ;
636 p[ip][0] += y[i2+32];
637 p[ip][1] += z[i2+32];
638 i2 += e[i2+32];
639 j2 += f[j2+32];
    
```

Part of Intel® Advisor, Intel® Parallel Studio XE 2016

Efficiently Vectorize your Code

Intel Advisor – Vectorization Advisor

Intel Advisor XE 2016

Elapsed time: 9.49s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources | Loops | All Threads

Summary | Survey Report | Refinement Reports | Annotation Report

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops				Trip Cou
						Vect...	Efficiency	Gain...	VL (...)	
[loop in matvec at Multiply.c:72]	1 Inefficient ...	5.281s	5.281s	Vectorized (Bo...		AVX	~59%	2.19x	4	25; 2
[loop in matvec at Multiply.c:66]	1 Ineffective ...	2.828s	2.828s	Vectorized (Bo...		AVX	~91%	3.65x	4	25; 2
[loop in matvec at Multiply.c:49]		0.531s	5.812s	Scalar						102
[loop in matvec at Multiply.c:49]		0.516s	3.344s	Scalar						102
[loop in matvec at Multiply.c:85]	1 Assumed d...	0.063s	0.063s	Scalar	vector dependence...					102
[loop in main at Driver.c:151]		0.016s	9.297s	Scalar	loop with function ...					1000000
[loop in matvec at Multiply.c:49]	1 Ineffective ...	0.000s	0.000s	Scalar						

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: Multiply.c:72 matvec

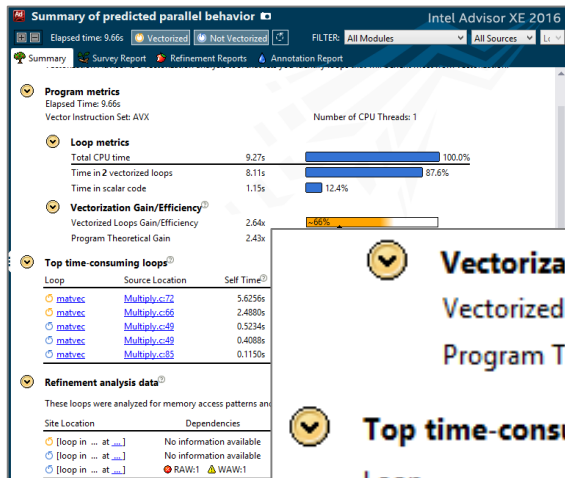
Line	Source	Total Time	%	Loop Time	%	Traits
65	#pragma nounroll					
66	for (j = 0; j < size2; j++) { [loop in matvec at Multiply.c:66] Vectorized AVX loop processes Float64 data type(s) No loop transformations applied [loop in matvec at Multiply.c:66] Scalar remainder loop No loop transformations applied [loop in matvec at Multiply.c:66] Scalar peeled loop [not executed] No loop transformations applied	2.625s		2.828s	0	
67	b[i] += a[i][j] * x[j];	0.688s				Extracts; In...
68	}					
69	/* In the following loop we are indexing the row of "a" by "1", this causes a cons					
	Selected (Total Time):	2.625s				

Optimization Notice

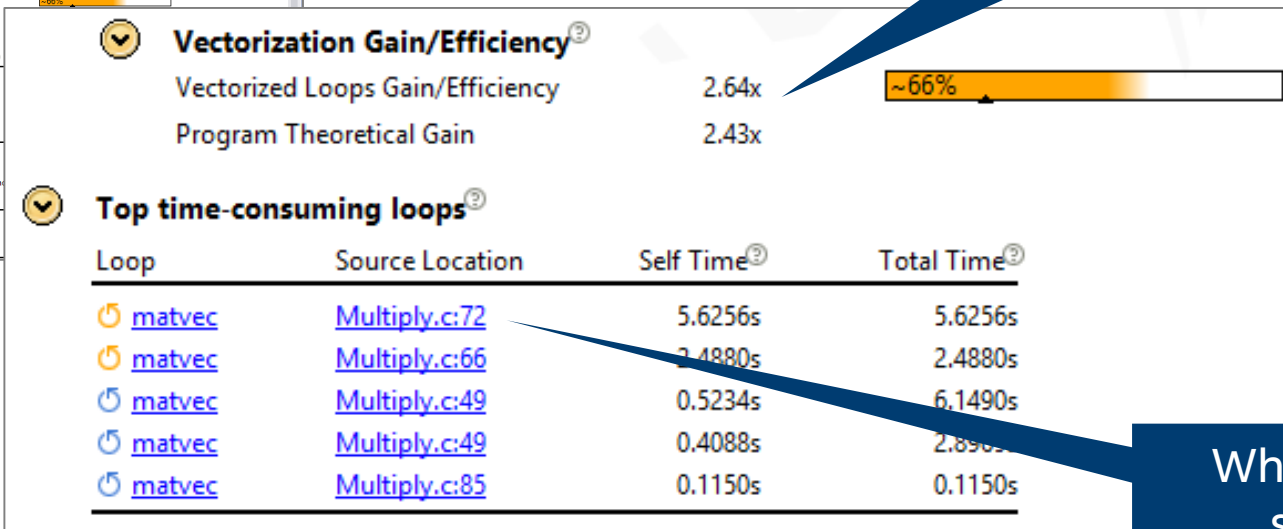
Copyright © 2016, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



Summary View: Plan Your Next Steps



What can I expect to gain?



Where do I start?

Optimization Notice

Vector Efficiency: All The Data In One Place

My “performance thermometer”

Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops				Trip Counts	Instruction Set Analysis		
					Vect...	Efficiency	Gain...	VL (...)		Traits	Data T...	Num.
1 Inefficient ...	5.281s	5.281s	Vectorized (Bo...		AVX	~55%	2.19x	4	25; 2	Inserts	Float64	3; 7
1 Ineffective ...	2.828s	2.828s	Vectorized (Bo...		AVX	~91%	3.65x	4	25; 2		Float64	3

~91%

~91%: Achieved Vectorization Efficiency

Achieved Vectorization Efficiency = (Estimated Gain/Vector Length) * 100%

Estimated Gain = 3.65x

Vector Length = 4

Orange color = Achieved vectorization efficiency is higher than reference efficiency for original scalar loop

Efficiency is approximately 91%, which means actual efficiency may be lower

(25%): Reference Efficiency for original scalar loop

Reference Efficiency = (1x/Vector Length) * 100%

(100%): Theoretical Maximum Vectorization Efficiency

Maximum Vectorization Efficiency = (Theoretical Maximum Gain/Vector Length) * 100%

Theoretical Maximum Gain = Currently selected Vector Length = 4

Look at Vector Issues and Traits to find out why

- All kinds of “memory manipulations”
- Usually an indication of “bad” access pattern

Spend your time in the most efficient place!

A typical vectorized loop consists of...

Main vector body

- Fastest among the three!

Fastest!

Optional peel part

- Used for the unaligned references in your loop. Uses Scalar or slower vector

Less
Fast

Remainder part

- Due to the number of iterations (trip count) not being divisible by vector length. Uses Scalar or slower vector.

Larger vector register means more iterations in peel/remainder

- Make sure you Align your data! (and you tell the compiler it is aligned!)
- Make the number of iterations divisible by the vector length!

6 Factors That Impact Vectorization Efficiency

Loop-carried dependencies

```
DO I = 1, N
  A(I+1) = A(I) + B(I)
ENDDO
```

Function calls

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;
  sumx = sumx + func(x, y, xp);
}
```

Pointer aliasing

```
void scale(int *a, int *b)
{
  for (int i = 0; i < 1000; i++)
    b[i] = z * a[i];
}
```

Unknown loop iteration count

```
struct _x { int d; int bound; };

void doit(int *a, struct _x *x)
{
  for(int i = 0; i < x->bound; i++)
    a[i] = 0;
}
```

Indirect memory access

```
for (i=0; i<N; i++)
  A[B[i]] = C[i]*D[i]
```

Outer loops

```
for(i = 0; i <= MAX; i++) {
  for(j = 0; j <= MAX; j++) {
    D[i][j] += 1;
  }
}
```

many

Data Dependencies – Tough Problem #1

Is it safe to force the compiler to vectorize?

Data dependencies

```
for (i=0;i<N;i++)          // Loop carried dependencies!  
  
    A[i] = A[i-1]*C[i]; // Need the ability to check if it  
  
                          // it is safe to force the compiler
```

Issue: Assumed dependency present

The compiler assumed there is an anti-dependency (Write after read – WAR) or true dependency (Read after write – RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Enable vectorization

Potential performance gain: Information not available until Beta Update release

Confidence this recommendation applies to your code: Information not available until Beta Update release

The Correctness analysis shows there is no real dependency in the loop for the given workload. Tell the compiler it is safe to vectorize using the `restrict` keyword or a [directive](#).

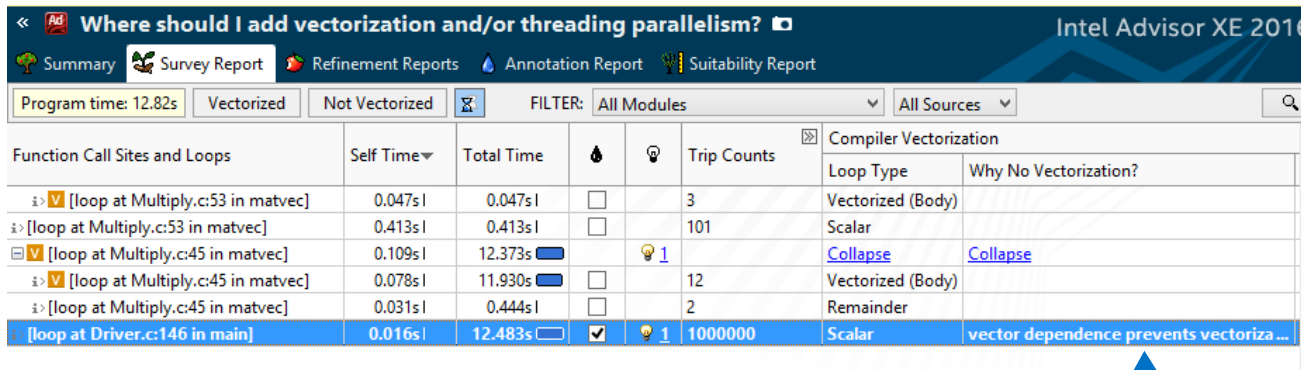
ICL/ICC/ICPC Directive	IFORT Directive	Outcome
<code>#pragma simd</code> or <code>#pragma omp simd</code>	<code>!DIR\$ SIMD</code> or <code>!\$OMP SIMD</code>	Ignores all dependencies in the loop
<code>#pragma ivdep</code>	<code>!DIR\$ IVDEP</code>	Ignores only vector dependencies (which is safest)

Read More:

- [User and Reference Guide for the Intel C++ Compiler 15.0](#) > **Compiler Reference** > **Pragmas** > **Intel-specific Pragma Reference** >
 - `ivdep`
 - `omp simd`

Check if It Is Safe to Vectorize

Loop-Carried Dependencies Analysis Verifies Correctness



Where should I add vectorization and/or threading parallelism? Intel Advisor XE 2016

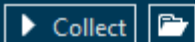
Summary Survey Report Refinement Reports Annotation Report Suitability Report

Program time: 12.82s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops	Self Time	Total Time	🔥	💡	Trip Counts	Compiler Vectorization	
						Loop Type	Why No Vectorization?
↳ [loop at Multiply.c:53 in matvec]	0.047s	0.047s	<input type="checkbox"/>		3	Vectorized (Body)	
↳ [loop at Multiply.c:53 in matvec]	0.413s	0.413s	<input type="checkbox"/>		101	Scalar	
↳ [loop at Multiply.c:45 in matvec]	0.109s	12.373s	<input type="checkbox"/>	💡 1		Collapse	Collapse
↳ [loop at Multiply.c:45 in matvec]	0.078s	11.930s	<input type="checkbox"/>		12	Vectorized (Body)	
↳ [loop at Multiply.c:45 in matvec]	0.031s	0.444s	<input type="checkbox"/>		2	Remainder	
↳ [loop at Driver.c:146 in main]	0.016s	12.483s	<input checked="" type="checkbox"/>	💡 1	1000000	Scalar	vector dependence prevents vectoriza...

2.1 Check Dependencies

Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.



Command Line

Select loop for
Dependency
Analysis and
press play!

Vector Dependence
prevents
Vectorization!

Improve Vectorization

Memory Access Pattern Analysis

Where should I add vectorization and/or threading parallelism?

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Elapsed time: 8,52s Vectorized Not Vectorized FILTER: All Modules All Sources

Function Call Sites and Loops				Loop Type	Why No Vectorization?
[loop at fractal.cpp:179 in <lambda1>::op ...	<input type="checkbox"/>	High vector ...	0,013sI 12,020sI	Collapse	Collapse
[loop at fractal.cpp:179 in <lambda1>::o ...	<input checked="" type="checkbox"/>	Serialized use ...	0,013sI 11,281sI	Vectorized (Body)	
[loop at fractal.cpp:179 in <lambda1>::o ...	<input checked="" type="checkbox"/>	Data type co ...	0,000sI 0,163sI	Peeled	
[loop at fractal.cpp:179 in <lambda1>::o ...	<input checked="" type="checkbox"/>	Data type co ...	0,000sI 0,576sI	Remainder	
[loop at fractal.cpp:177 in <lambda1>::oper ...	<input type="checkbox"/>	Data type co ...	0,010sI 12,030sI	Scalar	

2.2 Check Memory Access Patterns

Identify and explore complex memory accesses for marked loops. Fix the reported problems.



Command Line

Run Memory Access Patterns analysis, just to check how memory is used in the loop and the called function

Get specific advice for Improving Vectorization

Source

Top Down

Loop Assembly

Recommendations

Compiler Diagnostic Details

Issue: Ineffective peeled/remainder loop(s) present

All or some [source loop](#) iterations are not executing in the [loop body](#). Improve performance by moving source loop iterations from [peeled/remainder](#) loops to the loop body.

Recommendation: Collect trip counts data Confidence: Need More Data

Recommendation: Specify the expected loop trip count Confidence: Low

The compiler cannot statically detect the [trip count](#). To fix: Identify the expected number of iterations using a [directive](#):
`#pragma loop_count.`

Example: Iterate through a loop a minimum of three, maximum of ten, and average of five times:

```
#include <stdio.h>
int mysum(int start, int end, int a) {
    int iret=0;
    #pragma loop_count min(3), max(10), avg(5)
    for (int i=start;i<=end;i++)
```

Read More:

- [loop_count](#)
- [Getting Started with Intel Compiler Pragmas and Directives and ...resources for Intel Advisor users](#)

Advisor XE shows hints how to decrease vectorization overhead

Recommendation: Enforce vectorized remainder Confidence: Low

Recommendation: Use a smaller vector length Confidence: Low

Recommendation: Align data Confidence: Low

Recommendation: Add data padding Confidence: Low

New Features in Intel® Advisor

Next generation Intel® Xeon Phi™ processor (code named Knights Landing)

Explore non-executed code paths

- Generate multiple code paths for instructions that your machine does not even support (for example AVX-512)

Batch mode workflow

Filter by thread

Loop Analytics

Gather instruction profiling

Vectorization Advisor runs on and optimizes for Intel® Xeon Phi™ architecture

AVX-512 ERI – specific to Intel® Xeon Phi

Loops	Vector Issues	Self Time	Loop Type	Vectorized Loops	Instruction Set Analysis							
				Vector ISA	Efficiency	Gain Esti...	VL (V...	Traits	Data Types	Vector ...	Instruction Sets	
[Loop]	3 Possible in...	35.226s	5.4%	Vectorized+Threaded (Body; Peeled; Re...	AVX512	-26%	2.21x	8	Divisions; FMA; Gathers	Float32; ...	256/512	AVX; AVX2; AVX512; ...
[loc]	2 Possible in...	26.025s	4.0%	Vectorized (Body)+Threaded (OpenMP)	AVX512			8	Divisions; Gathers; FMA	Float32; ...	256/512	AVX; AVX512ER; AVX512F...
[loc]	1 High vecto...	5.876s		Vectorized (Peeled)+Threaded (OpenMP)	AVX512			8	Divisions; Gathers; FMA	Float32; ...	256/512	AVX2; AVX512ER; AVX512...
[loc]	1 High vecto...	3.324s		Vectorized (Remainder)+Threaded (Open...	AVX512			8	Divisions; Gathers; FMA	Float32; ...	256/512	AVX2; AVX512ER; AVX512...
[loop]		34.599s	5.3%	Vectorized (Body; Remainder)	AVX512	-70%	5.64x	8	Divisions; FMA; Square Roots	Float32; ...	256/512	AVX2; AVX512ER; AVX512...
[loop]	1 Possible in...	33.849s	5.2%	Vectorized (Body; Peeled; Remainder)	AVX512	-28%	2.24x	8	Divisions; FMA; Gathers	Float32; ...	256/512	AVX; AVX2; AVX512ER; AV...
[loop]		19.839s	3.1%	Vectorized (Body; Remainder)	AVX512	-72%	11.46x	16; 8				

Efficiency (72%), Speed-up (11.5x), Vector Length (16)

Issue: Possible inefficient memory access patterns present
Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization by the compiler. Improve performance by investigating.

Recommendation: Confirm inefficient memory access patterns
There is no confirmation inefficient memory access patterns are present. To confirm: Run a [Memory Access Patterns analysis](#).

Confidence: Need More Data

Issue: Ineffective peeled/remainder loop(s) present
All or some [source loop](#) iterations are not executing in the [loop body](#). Improve performance by moving source loop iterations from [peeled/remainder](#) loops to the loop body.

Recommendation: Collect trip counts data
The Survey Report lacks [trip counts](#) data that might generate more precise recommendations. To fix: Run a [Trip Counts analysis](#).

Recommendation: Align data

Recommendation: Add data padding
The [trip count](#) is not a multiple of [vector length](#). To fix: Do one of the following:

- Increase the size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.

Windows® OS	Linux® OS
/Ox:auto-assume-safe-padding	-xauto-assume-safe-padding

Performance optimization problem and advice how to fix it

Program metrics
Elapsed Time: 142.79s
Vector Instruction Set: AVX, AVX2, AVX512, SSE, SSE2
Number of CPU Threads: 4

Loop metrics

Metric	Value	Percentage
Total CPU time	454.08s	100.0%
Time in 88 vectorized loops	41.86s	9.2%

Optimization Notice

Start Tuning for AVX-512 without AVX-512 hardware

Intel® Advisor - Vectorization Advisor

Use `-axCOMMON-AVX512 -xAVX` compiler flags to generate both code-paths

- AVX(2) code path (executed on Haswell and earlier processors)
- AVX-512 code path for newer hardware

Compare AVX and AVX-512 code with Intel Advisor

Loops	Self Time	Loop Type	Vectorized Loops					Instruction Set Analysis				Advanced
			Vect...	Efficiency	Gain...	VL (...)	Compiler Es...	Traits	Data T...	Vector W...	Instruction Sets	Vectorization D...
[loop in s352_at loopstl.cpp:5939]	0,641s	Vectorized (Body)	AVX2	~54%	2,15x	4	2,15x	FMA; Inserts	Float32	128	AVX; FMA	
[loop in s352_at loopstl.cpp:5939]	n/a	Remainder [Not Executed]				4		FMA				
[loop in s352_at loopstl.cpp:5939]	0,641s	Vectorized (Body)	AVX2			4	2,15x	Inserts; FMA				
[loop in s352_at loopstl.cpp:5939]	n/a	Vectorized (Body) [Not Executed]	AVX512			16	3,20x	Gathers; FMA				
[loop in s352_at loopstl.cpp:5939]	n/a	Vectorized (Remainder) [Not Executed]	AVX512			16	2,70x	Gathers; FMA				
[loop in s125_ ASomp\$parallel_for@...]	0,496s	Vectorized Versions	AVX2	~100%	13,54x	8	<13,54x	FMA; NT-stores				
[loop in s125_ ASomp\$parallel_for@...]	n/a	Peeled [Not Executed]				8		FMA				
[loop in s125_ ASomp\$parallel_for@...]	n/a	Remainder [Not Executed]				8		FMA				
[loop in s125_ ASomp\$parallel_for@...]	0,465s	Vectorized (Body)	AVX2			8	13,54x					
[loop in s125_ ZSomp\$parallel_for@...]	n/a	Vectorized (Peeled) [Not Executed]	AVX512			16	6,77x	FMA				
[loop in s125_ ZSomp\$parallel_for@...]	n/a	Vectorized (Body) [Not Executed]	AVX512			32	30,61x	NT-stores				
[loop in s125_ ZSomp\$parallel_for@...]	n/a	Vectorized (Remainder) [Not Executed]	AVX512			16	9,78x	FMA				

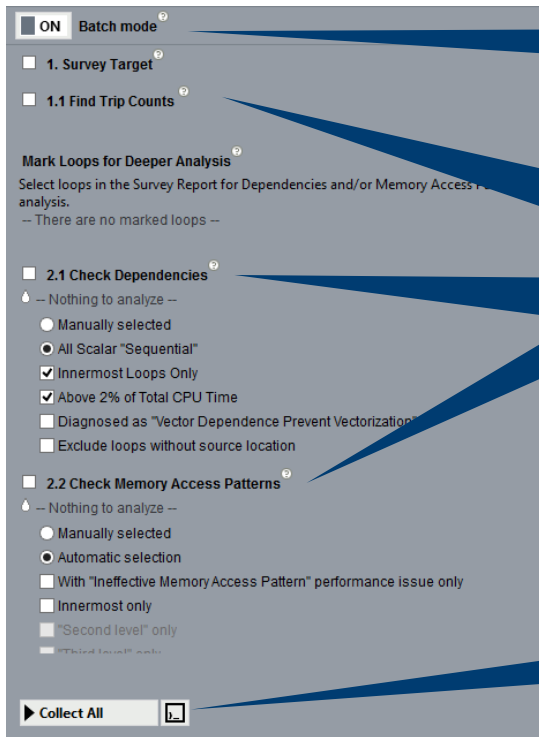
Inserts (AVX2) vs. Gathers (AVX-512)

Speed-up estimate: 13.5x (AVX2) vs. 30.6x (AVX-512)

Optimization Notice

Batch Mode Workflow Saves Time

Intel® Advisor - Vectorization Advisor



Turn On
Batch Mode

Select
analyses to
run

Click
Collect all

Run several analyses in batch
as a single run

Contains pre-selected criteria
for advanced analyses

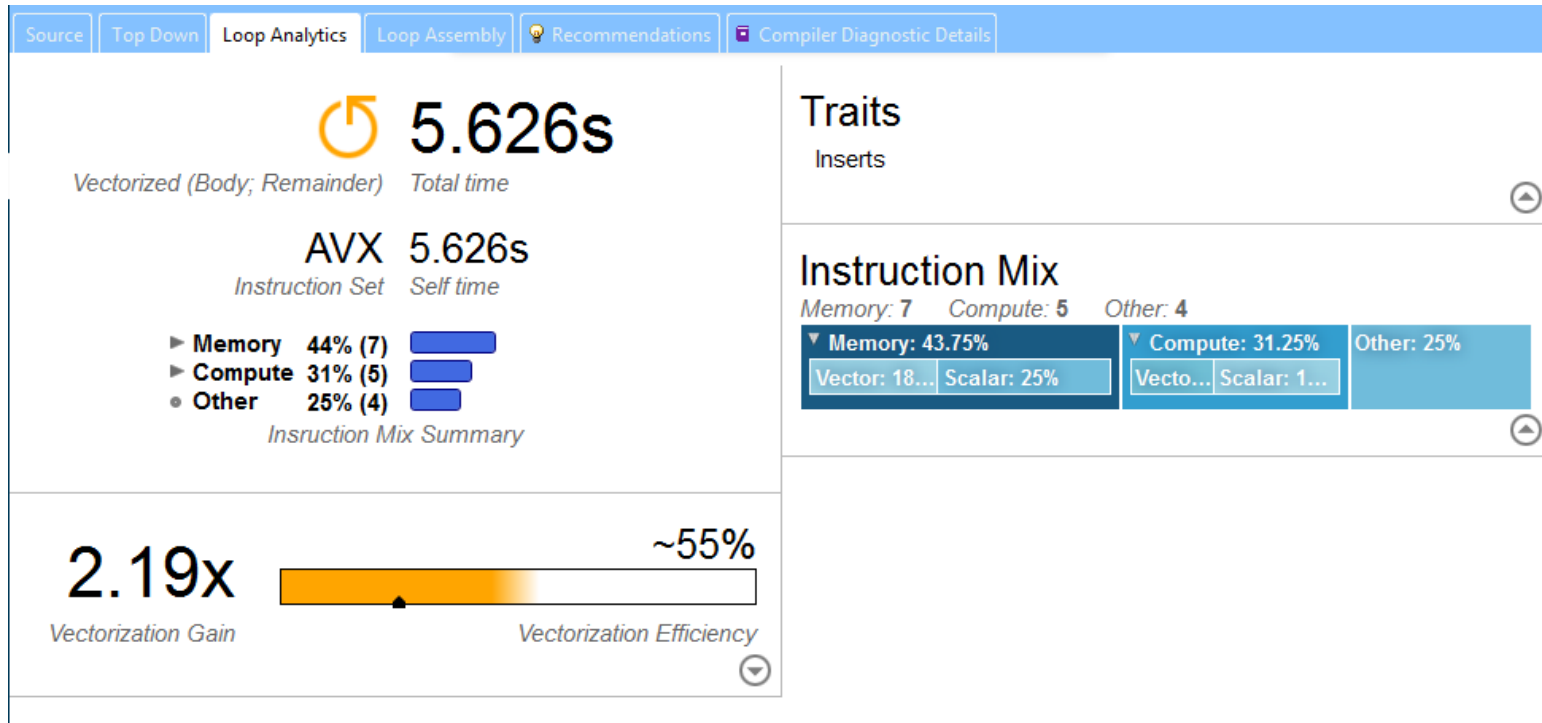
Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Loop Analytics

Get detailed information about your loops

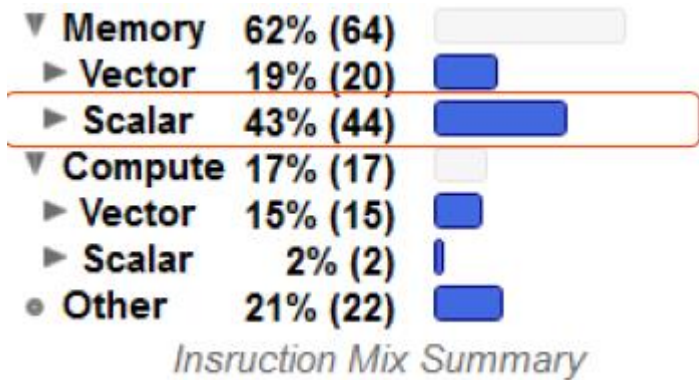


Optimization Notice

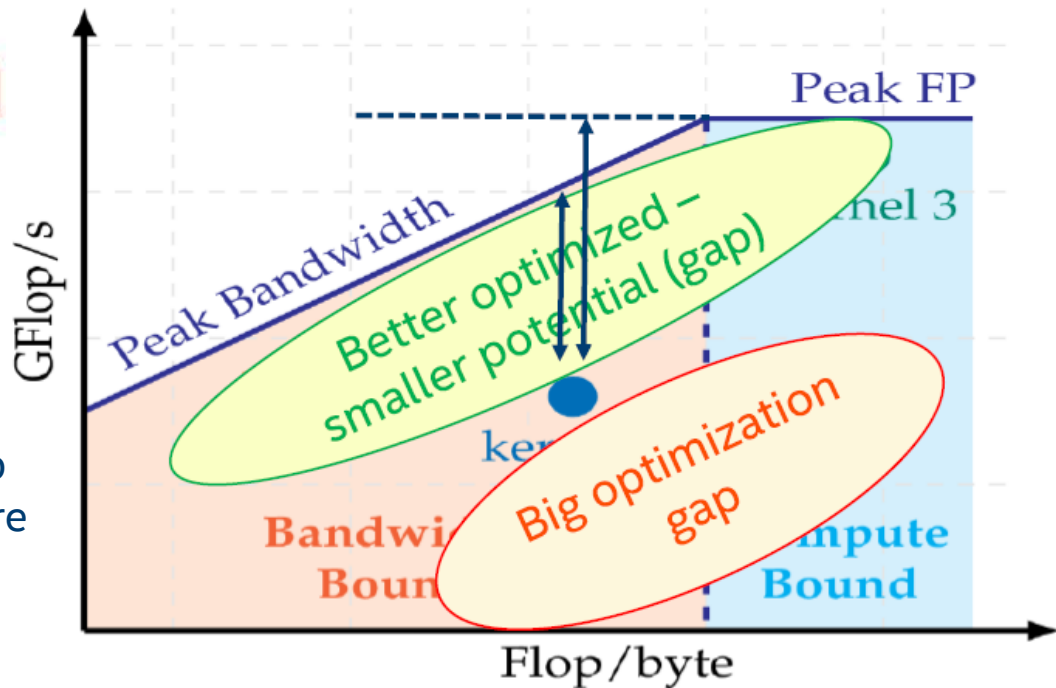
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Am I memory bound or VPU/CPU bound?



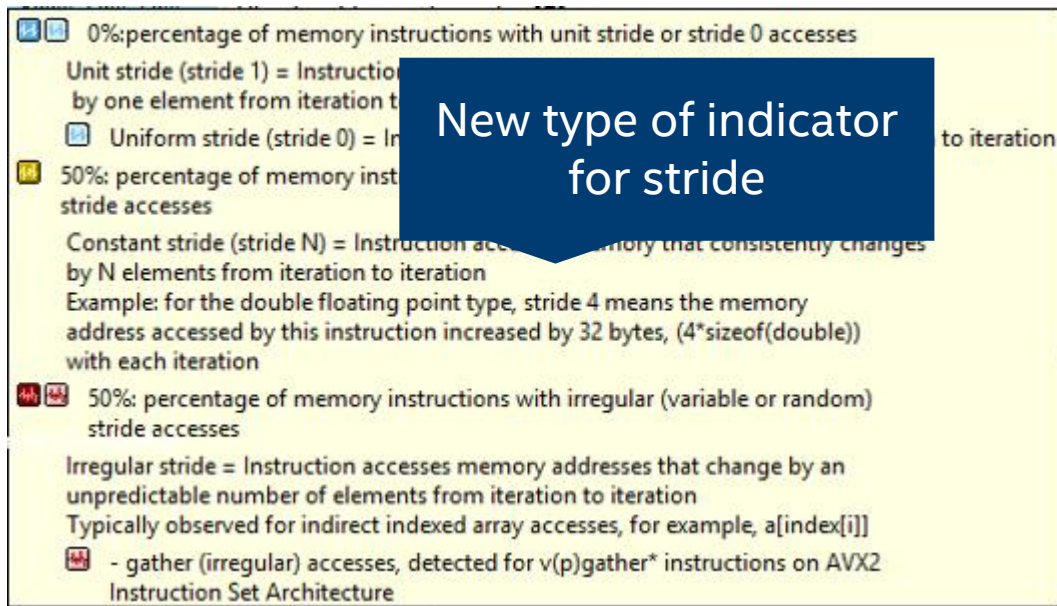
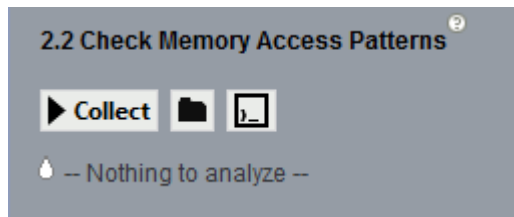
The types of instructions in your loop will be an indicator of whether you are memory bound or compute bound.



Irregular access patterns decreases performance!

Gather profiling

Run Memory Access Pattern Analysis



0%: percentage of memory instructions with unit stride or stride 0 accesses

Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration

Uniform stride (stride 0) = Instruction accesses memory that consistently changes by 0 elements from iteration to iteration

50%: percentage of memory instructions with uniform stride accesses

Constant stride (stride N) = Instruction accesses memory that consistently changes by N elements from iteration to iteration

Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4*sizeof(double)) with each iteration

50%: percentage of memory instructions with irregular (variable or random) stride accesses

Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration

Typically observed for indirect indexed array accesses, for example, a[index[i]]

- gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture

New type of indicator for stride

Gather/Scatter analysis is very important for AVX512

AVX512 Gather/Scatter in wider use than on previous instruction sets

- Many more applications can now be vectorized
- Gives good average performance but far from optimal
- Much greater need for Gather/Scatter profiling
- With Intel® Advisor you get both dynamic and static gather/scatter information

Memory Analysis Is Critical

Determine Possible Bandwidth or Latency Issues

Footprint	Small enough	Big enough	Source	Stride	Operand Type	Operand Size ...	Aggregated footprint
Access Pattern			<pre>m=1; m<=half; m++) { - fCpMod(i + lbv[3*m], Ymax); = fCpMod(j + lbv[3*m+1], Ymax); = fCpMod(k + lbv[3*m+2], Zmax); = (nextx * Ymax + nexty) * Zmax + nextz;</pre>	<ul style="list-style-type: none"> [0] [3] [0] [3] [0] [3] 	int	32	48
Unit Stride	Effective SIMD No Latency and BW bottlenecks	Effective SIMD Bandwidth bottleneck	<pre>lboisiteLength + 1*lbay.nq + m + half], lbf[i1next*lboisitele</pre>	[0] [1] [-4]	float64:int	32:64	9MB
Const stride	Medium SIMD Latency bottleneck possible	Medium SIMD Latency and Bandwidth bottleneck possible	<pre>lboisiteLength + 1*lbay.nq + m + 1], lbf[i1*lboisiteLength + 1</pre>				
Irregular Access, Gather/Scatter	Bad SIMD Latency bottleneck possible	Bad SIMD Latency bottleneck					
			Assembly	Physical Stride	Operand Info	Address range	Memory access
			f298 1254 add r14d, r12d				
			f2cb 1256 mov r12, qword ptr [r9+rsi*8]	-43775, 118377...	int*1, int*1, i...	0x27561058 - 0x27e6cf20	9MB
			f2cf 1256 vmovsd xmm0, qword ptr [r8+rbx*8]	1	float64*1	0x27561098 - 0x275610d0	64B
			f2d5 1256 mov qword ptr [r8+rbx*8], r12	1	int*1	0x27561098 - 0x275610d0	64B
			f2d9 1256 mov r13d, dword ptr [r1p+0x1565bc]	0	int*1	0x18589c - 0x18589c	4B

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.




Detect possible latency or bandwidth issues

Enhanced Memory Analysis

Is your
memory
range small
enough to fit
in cache?

Address	Line	Physical Stride	Operand Info	Vector Length	Operand Size (bits)	Address range	Memory access fo...
0x14002eabd							
0x14002eac1							
0x14002eac7							
0x14002eacb							
0x14002ead0							
0x14002ead0	2		float32*8	8	256	0x23a2d960 - 0x23a ...	288B
0x14002ead6	2		float32*8	8	256	0x23a2d980 - 0x23a ...	288B
0x14002eadd							
0x14002eae1							
0x14002eae5			float32*8;int ...	8	32	0x87af10 - 0x87b038	300B
0x14002eaec							
0x14002eaf0							
0x14002eaf4			float32*8;int ...	8	32	0x87af28 - 0x87b038	300B
0x14002eafb	2		float32*8	8	256	0x879f70 - 0x87b038	288B
0x14002eb01	2		float32*8	8	256	0x879f90 - 0x87b038	288B

 Gather (irregular) access

Operand Size (bits): 32
Operand Type: float32,int32
Vector Length: 8
Memory access footprint: 300B

▼ **Gather details**

Mask is constant
Mask: [11111111]
Active elements in the mask: 100.0%

Optimization Notice

Mask Utilization and FLOPS profiler

- Long-waiting in HPC: accurate HW independent FLOPs measurement tool
- Not just count FLOPs. Has following additions:
 - (AVX-512 only) Mask-aware. Masked-Memory/Unmasked-Compute pattern aware
 - Unique capability to correlate FLOPs with performance data (obtained without instrumentation). Gives FLOPs/s.
- Lightweight instrumentation, PIN-based, benefits from “threadchecker tools” and more generally Advisor framework integration.

Why is Mask Utilization important?

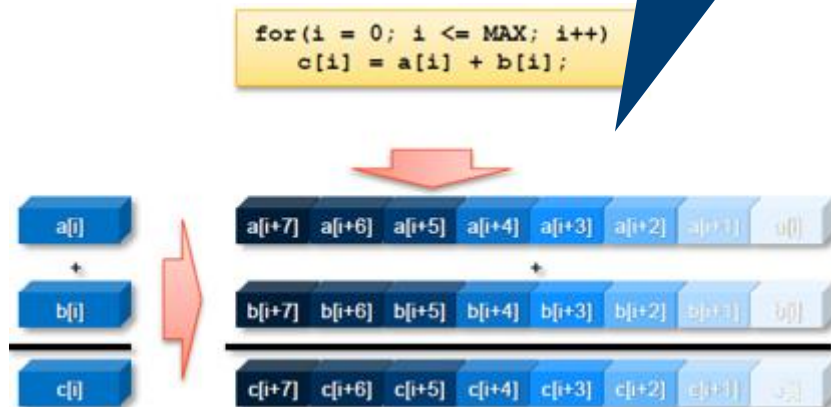
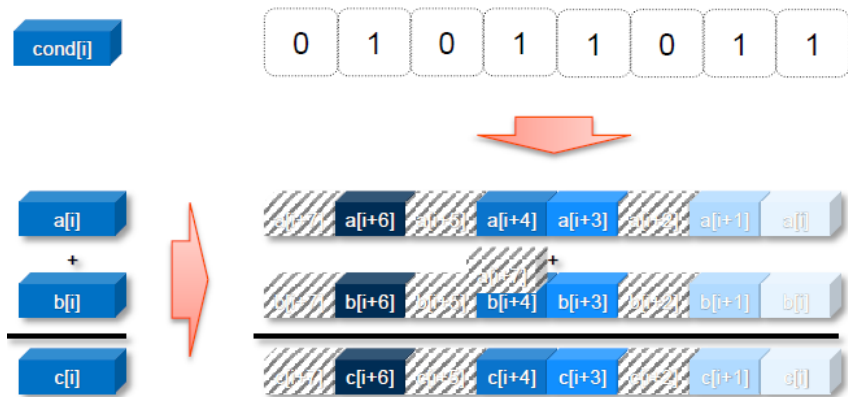
3 elements suppressed

SIMD Utilization = 5/8
(62.5%)

Not utilizing full vectors!!

Fully utilized!

```
for(i = 0; i <= MAX; i++)  
  if (cond(i))  
    c[i] = a[i] + b[i];
```

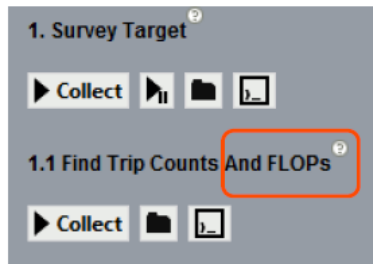


FLOPs and Mask Utilization Profiler

Tech Preview

set **ADVIXE_EXPERIMENTAL=FLOPS**

advixe-cl.exe --collect survey
advixe-cl.exe --collect trip counts



FLOPs, Masks, Trip Counts					
Median	GFLOPs/s	Arithmetic Intensity	Mask Utiliz...	GBytes/s	GFLOP
19	2,456	0.125		19.6498	3.94488
4; 3	2,351	0.125	63,29%	18.8111	0.36693
19	2,136	0.0795455		26.8513	2.50206
19	1,910	0.0681818		28.011	1.07231
3	1,774	0.0833333		21.2898	0.11287
4	1,192	0.0666667		17.8726	0.1505
19	0,911	0.0681818		13.3635	0.0285

Call to Action

Modernize your Code

- To get the most out of your hardware, you need to modernize your code with vectorization and threading.
- Taking a methodical approach such as the one outlined in this presentation, and taking advantage of the powerful tools in Intel® Parallel Studio XE, can make the modernization task dramatically easier.
- Send e-mail to vector_advisor@intel.com to get the latest information on some exciting new capabilities that are currently under development.

Resources

Intel® Advisor Links

- Vectorization Guide
 - <http://bit.ly/autovectorize-guide>
- Explicit Vector Programming in Fortran
 - <http://bit.ly/explicitvector-fortran>
- Optimization Reports
 - <http://bit.ly/optimizereports>
- Beta Registration & Download
 - <http://bit.ly/PSXE2017-Beta>

Code Modernization Links

- Modern Code Developer Community
 - software.intel.com/modern-code
- Intel Code Modernization Enablement Program
 - software.intel.com/code-modernization-enablement
- Intel Parallel Computing Centers
 - software.intel.com/ipcc
- Technical Webinar Series Registration
 - <http://bit.ly/spring16-tech-webinars>
- Intel Parallel Universe Magazine
 - software.intel.com/intel-parallel-universe-magazine

Additional Resources

For Intel® Xeon Phi™ coprocessors, but also applicable:

- <https://software.intel.com/en-us/articles/vectorization-essential>
- <https://software.intel.com/en-us/articles/fortran-array-data-and-arguments-and-vectorization>

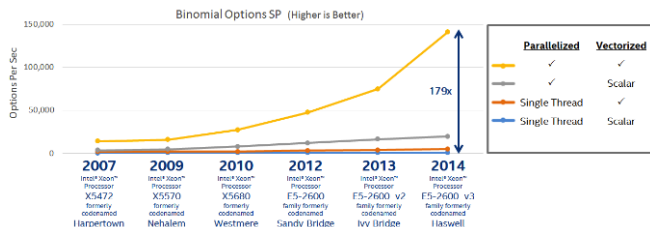
Intel® Parallel Studio XE Composer Edition User and Reference Guides:

- <https://software.intel.com/en-us/intel-cplusplus-compiler-16.0-user-and-reference-guide-pdf>
- <https://software.intel.com/en-us/intel-fortran-compiler-16.0-user-and-reference-guide-pdf>

Compiler User Forums

- <http://software.intel.com/forums>

Configurations for Binomial Options SP



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

Performance measured in Intel Labs by Intel employees

Platform Hardware and Software Configuration

Platform	Unscaled Core Frequency	Cores/Socket	Num Sockets	L1 Data Cache	L1 I Cache	L2 Cache	L3 Cache	Memory	Memory Frequency	Memory Access	H/W Prefetchers Enabled	HT Enabled	Turbo Enabled	C States	O/S Name	Operating System	Compiler Version
Intel® Xeon™ 5472 Processor	3.0 GHZ	4	2	32K	32K	12 MB	None	32 GB	800 MHZ	UMA	Y	N	N	Disable	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ X5570 Processor	2.93 GHZ	4	2	32K	32K	256K	8 MB	48 GB	1333 MHZ	NUMA	Y	Y	Y	Disable	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ X5680 Processor	3.33 GHZ	6	2	32K	32K	256K	12 MB	48 MB	1333 MHZ	NUMA	Y	Y	Y	Disable	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 2690 Processor	2.9 GHZ	8	2	32K	32K	256K	20 MB	64 GB	1600 MHZ	NUMA	Y	Y	Y	Disable	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 2697v2 Processor	2.7 GHZ	12	2	32K	32K	256K	30 MB	64 GB	1867 MHZ	NUMA	Y	Y	Y	Disable	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 2697v2 Processor	2.2 GHz	14	2	32K	32K	256K	35 MB	64 GB	2133 MHZ	NUMA	Y	Y	Y	Disable	Fedora 20	3.13.5-202.fc20	icc version 14.0.1

Optimization Notice

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

