



HPC codes modernization using vector and threading parallelism – part 2 (tools)

Zakhar A. Matveev, PhD,

Intel Russia, Intel Software and Services Group

May' 2015





Code modernization: Intel® Parallel Studio XE 2016 Beta



Intel® Parallel Studio XE

Faster code faster!

Vectorizing **Compiler**

Squeeze all the performance out of the latest instruction set

Threaded Performance **Libraries**

Pre-vectorized, pre-threaded, pre-optimized

Vectorization Optimization and Thread Prototyping

Data driven design tools help you vectorize & thread effectively

High Level **Parallel Models**

Productive solutions for thread, process & vector parallelism

Parallel Performance **Profilers**

Quickly discover bottlenecks and tune for high performance

Threading **Inspector**

Find and debug non-deterministic threading errors



[Download Today](#)

Google:

“Intel Parallel Studio 2016”

Or go directly to:

<https://software.intel.com/en-us/articles/intel-parallel-studio-xe-2016-beta>

Intel® Parallel Studio XE 2016 Suites

Vectorization – Boost Performance By Utilizing Vector Instructions / Units

- Intel® Advisor XE - **Vectorization Advisor** identifies new vectorization opportunities as well as improvements to existing vectorization and highlights them in your code. It makes actionable coding recommendations to boost performance and estimates the speedup.

Scalable MPI Analysis – Fast & Lightweight Analysis for 32K+ Ranks

- Intel® Trace Analyzer and Collector add **MPI Performance Snapshot** feature for easy to use, scalable MPI statistics collection and analysis of large MPI jobs to identify areas for improvement

Big Data Analytics – Easily Build IA Optimized Data Analytics Application

- Intel® Data Analytics Acceleration Library (Intel® DAAL) will help data scientists speed through big data challenges with optimized IA functions

Standards – Scaling Development Efforts Forward


- Supporting the evolution of industry standards of **OpenMP***, **MPI**, **Fortran** and **C++** Intel® Compilers & performance libraries

Free Intel® Software Development Tools:

<https://software.intel.com/en-us/qualify-for-free-software>

Free Software Tools

Supporting qualified students, educators, academic researchers and open source contributors



Free Intel® Software Development Tools for:



Academic Researcher ›

For unfunded research (research not funded by grants).



Student ›

For current students at degree-granting institutions.



Educator ›

For use in teaching curriculum.



Open Source Contributor ›

For developers actively contributing to open source projects.

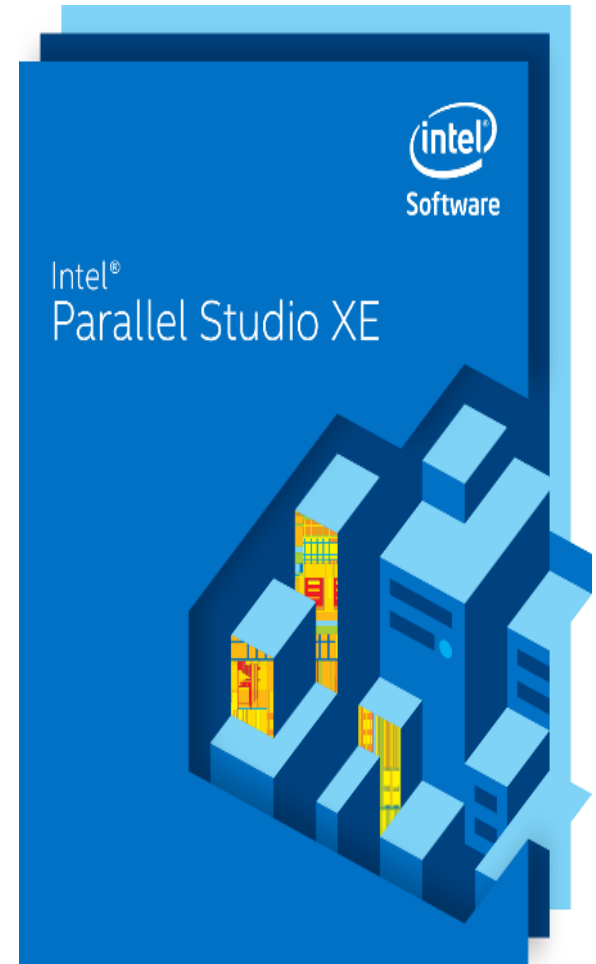
Beta 2016 program : Call to Action

Participate in the Beta Program today!

- Register at **bit.ly/psxe2016beta**
- Or simply send e-mail to **vector_advisor@intel.com**

Submit Feedback via Intel® Premier Support

Tell us about your experiences using the Intel® Parallel Studio XE 2016 Beta



Intel® Advisor XE

Vectorization Optimization and Thread Prototyping



In Beta
Sign-up!

Data Driven Code Modernization and Optimization For Vector and Threading Parallelism Intel® Advisor XE – Vectorization Advisor

Have you:

- Recompiled with AVX2, but seen little benefit?
- Wondered where to start adding vectorization?
- Recoded intrinsics for each new architecture?
- Struggled with cryptic compiler vectorization messages?

Breakthrough for vectorization design

- What is blocking vectorization and why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?

Vectorization challenges

LLNL (Hornung, Keasler, 2013):

"Typical codes get less than 5% of their FP instructions SIMD-ized... multi-physics codes - have thousands of small loops, which are all important"

Efficient vectorization is sometimes challenging:

- “Poor” baselines : solid gap
 - “Ninja gap” as well
- Industry code complexities
 - Vector productivity problem: “thousands of loops”
 - Need to know both static and dynamic code characteristics
- Some codes: demand for expensive data layout reorganization

“Vectorization Advisor”

1. *“All the data you need in one place”.*

Leverages Intel Compiler diagnostics, performance data and ISA statistics.

2. Detects “hot” un-vectorized or “under vectorized” loops. Identifies what is blocking efficient vectorization and why

3. Identify performance penalties and recommends fixes for them (including OpenMP4.x)

4. Memory layout analysis: explore alternative data reorganizations

5. Increase the confidence that vectorization is safe

**Beta program : started April'15.
Release: end of August '15.**

Vectorization Advisor.

Assist code modernization for x86 SIMD

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time
[loop in runCForallLambdaLoops]	0.094s	
[loop in runCForallLambdaLoops]	0.140s	
[loop in std::complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s
Vectorized SSE; SSE2 loop processing Float32; Float64 data type		
Peeled loop: loop stats were reordered		
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	5.000s
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	5.000s
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	

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

Issue: Peeled/Remainder loop(s) present

...nel loop. Improve performance by moving ...nel loop. Read more at [Vector Essentials](#).

...memory accesses in the source loop does not ...l the compiler your memory access is aligned.

```
(at), 32);
```

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

Trip Counts			
Median	Min	Max	Call Count
101	101	101	12000000
3	3	3	1000000
101	101	101	2000000
1000000	1000000	1000000	1

4. Loop-Carried Dependency Analysis

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

5. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRowLoops	runCRowLoops.cxx:1063	RAW:1	No information available	No information available
loop_site_139	runCRowLoops	runCRowLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRowLoops	runCRowLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns		Correctness Report			
ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCRowLoops.cxx:637	lcalcs.exe	
<pre> 635 j2 = (j2 * 64 - 1) ; 636 p[ip][0] += y[i2+32]; 637 p[ip][1] += z[j2+32]; 638 i2 += e[i2+32]; 639 j2 += f[j2+32]; </pre>					
P23	0; 0	Unit stride	runCRowLoops.cxx:638	lcalcs.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRowLoops.cxx:628	lcalcs.exe	
<pre> 626 i1 = 64-1; 627 j1 = 64-1; 628 p[ip][2] += b[j1][i1]; </pre>					

Data-Driven Threading Design

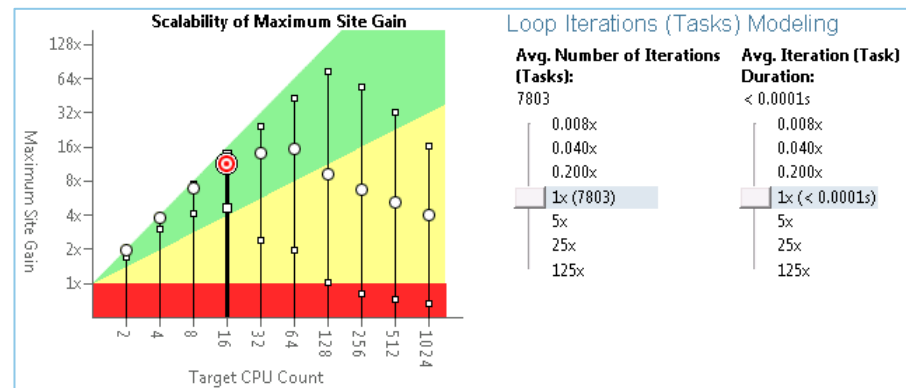
Intel® Advisor XE – Thread Prototyping

Have you:

- Tried threading an app, but seen little performance benefit?
- Hit a “scalability barrier”? Performance gains level off as you add cores?
- Delayed a release that adds threading because of synchronization errors?

Breakthrough for threading design:

- Quickly prototype multiple options
- Project scaling on larger systems
- Find synchronization errors before implementing threading
- Separate design and implementation - Design without disrupting development



**Add Parallelism with Less Effort,
Less Risk and More Impact**

<http://intel.ly/advisor-xe>

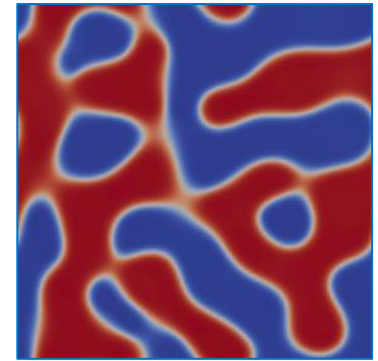
Vectorization Advisor in action

Live Case Study with “DL-MESO”

DL-MESO



Hartree Centre
Science & Technology Facilities Council



Computational fluid dynamics engine

- New mesoscopic simulation engine
- Applicable for problems such as inkjet printing and steel production
- Lattice Boltzmann Equation

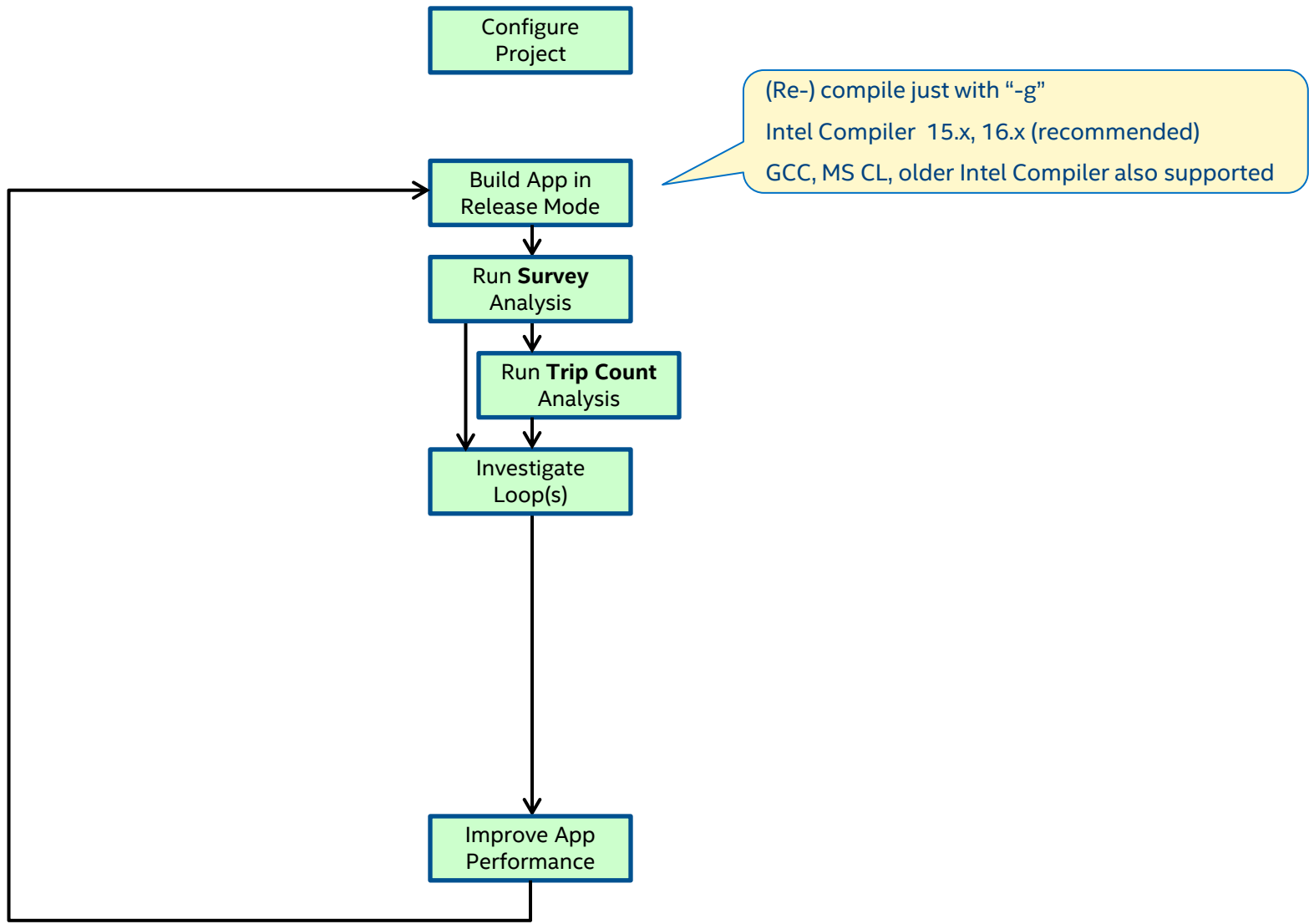
Developed by EPSRC CPP5

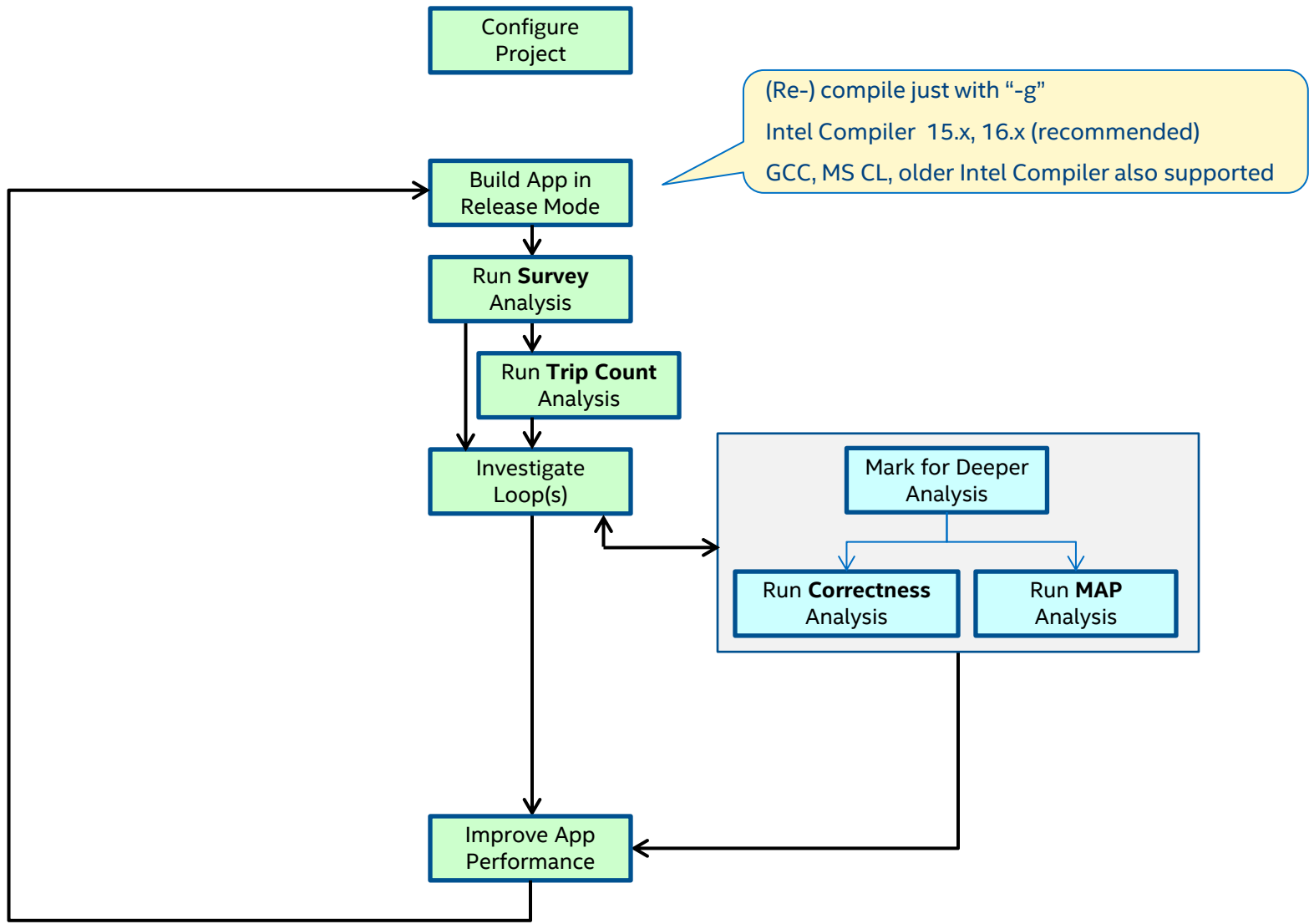
- including Hartree, Oxford, Imperial College
- Michael Seaton at Hartree as major contributor

Workload characteristics:

- “Flat profile”, many small kernels
- Profiles are very diverse depending on input datasets

0. Workflow





1. The Right Data At Your Fingertips

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	<input type="checkbox"/>		Scalar	vector dependence prevents vector...
⊞ [loop in runCforallLambdaLoops]	0.140s	3.744s	<input type="checkbox"/>		Scalar	inner loop was already vectorized
⊞ [loop in std::Complex_base<double,struct _C_double_complex>::f...	0.031s	0.031s	<input checked="" type="checkbox"/>		Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations						
Peeled loop: loop stmts were reordered						
⊞ [loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	544.0...	<input type="checkbox"/>		Scalar	nonstandard loop is not a vectoriza ...
⊞ [loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	544.0...	<input type="checkbox"/>		Scalar	nonstandard loop is not a vectoriza ...
⊞ [loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	0.234s	<input type="checkbox"/>		Scalar	nonstandard loop is not a vectoriza ...

Hot Loops: optimize one by one

Loops	🔥	Vector Issues	Self Time	Total Time	Loop Type
i> [loop at lbpSUB.cpp:1280 in fPropagationSwap]	<input type="checkbox"/>	💡 1 Assumed depend ...	1,691s	1,691s	Scalar
i> [loop at lbpSUB.cpp:761 in fGetEquilibriumF]	<input type="checkbox"/>	💡 2 Data type conver ...	1,384s	1,384s	Scalar
i> [loop at lbpGET.cpp:281 in fGetSpeedSite]	<input type="checkbox"/>	💡 2 Data type conversi ...	1,027s	1,027s	Scalar
i> [loop at lbpBGK.cpp:30 in fSiteFluidCollisionBGK]	<input type="checkbox"/>		0,599s	0,599s	Scalar
i> [loop in I10_OUTPUT]	<input type="checkbox"/>		0,150s	0,150s	Scalar
i> [loop at lbpBGK.cpp:21 in fSiteFluidCollisionBGK]	<input type="checkbox"/>		0,119s	2,479s	Scalar
i> [loop at lbpSUB.cpp:1264 in fPropagationSwap]	<input type="checkbox"/>	💡 1 Assumed depend ...	0,090s	0,622s	Scalar
⊕ [loop at lbpGET.cpp:42 in fGetSpeedSite]	<input type="checkbox"/>	💡 3 Ineffective peel ...	0,080s	0,080s	Scalar Optimized: Expand
i> [loop at lbpSUB.cpp:1276 in fPropagationSwap]	<input type="checkbox"/>		0,060s	1,751s	Scalar
⊕ [loop at lbpGET.cpp:152 in fGetSpeedSite]	<input type="checkbox"/>	💡 2 Ineffective peel ...	0,059s	0,059s	Scalar Optimized: Expand

Will only work through first 2 loops with max impact

Both loops were not auto-vectorized

Relatively "flat" profile

Quickly focus on loops which really matter

The Right Data At Your Fingertips

Get all the data you need for high impact vectorization

Am I vector?

Trip Counts

What prevents vectorization?

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
							Vector...	Efficiency	Vector L..
[loop at stl_algo.h:4740 in std::tr...		0.170s	0.170s		Scalar	non-vectorizable loop ins ...			
[loop at loopstl.cpp:2449 in s234_]	2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop at loopstl.cpp:2449 in s...		0.150s	0.150s	12	Vectorized (Body)		AVX		4
[loop at loopstl.cpp:2449 in s...		0.020s	0.020s	4	Remainder				
[loop at loopstl.cpp:7900 in vas_]		0.170s	0.170s	500	Scalar	vectorization possible but ...			4
[loop at loopstl.cpp:3509 in s2 ...]	1 High vector register ...	0.160s	0.160s	12	Expand	Expand	AVX	~69%	8
[loop at loopstl.cpp:3891 in s279_]	2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%	8
[loop at loopstl.cpp:6249 in s414_]		0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
[loop at numeric.h:247 in std ...]	1 Assumed dependency...	0.150s	0.150s	49	Scalar	vector dependence preve ...			

Focus on hot loops

What vectorization issues do I have?

Which Vector instructions are being used?

How efficient is the code?

Assist user at different LoD and perspectives

END-USER GUIDANCE

Item	Time	Time	Vectorization	Notes	Architecture	Operations	Count	Precision
[loop at ru...]	0.310s	0.310s	<input type="checkbox"/>	Loop was already vectorized	AVX	Inserts; Extracts	128/256	Float64
[loop at ru...]	0.309s	2.679s	<input type="checkbox"/>	Loop was vectorized	AVX	Inserts; Extracts	128/256	Float64
[loop at ru...]	0.258s	0.258s	<input type="checkbox"/>	volatile assignment was not vectorized. Try using no ...	AVX	Inserts; Extracts	128/256	Float64
[loop at ru...]	0.258s	0.258s	<input type="checkbox"/>	<Expand to see more ...>	AVX	Extracts	128/256	Float64
[loop at ru...]	0.240s	0.240s	<input type="checkbox"/>	<Expand to see more ...>	AVX	Inserts	128/256; 128	Float64

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

Issues: 1
Recommendations: 2

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. Read more at [Glossary and Vector Essentials](#), [Utilizing Full Vectors...](#)

Use a smaller vector length

ANALYSIS on WORKLOAD (high-level),...

Item	Time	Time	Vectorization	Notes	Architecture	Operations
[loop at nbody.cc:57 in main]	1,810s	1,810s	<input type="checkbox"/>	Vectorized (Body)	AVX	Square Roots; Inserts; Extracts; Masked Stores
[loop at nbody.cc:57 in main]	0,010s	0,010s	<input type="checkbox"/>	Peeled		
[loop at nbody.cc:54 in main]	0,000s	1,820s	<input type="checkbox"/>	Scalar	AVX	Shuffles; Inserts; Extracts
[loop at nbody.cc:54 in main]	0,000s	1,820s	<input type="checkbox"/>	Scalar		

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

File: nbody.cc:57 main

```

52 SOURCE, ..
53
54 for ( size_t i = 0; i < n; ++i ) {
55     real dvx = 0, dvy = 0, dvz = 0;
56     ##pragma vector always
57     for ( size_t j = 0; j < n; ++j ) {
58         [loop at nbody.cc:57 in main]
59         Scalar loop. Not vectorized
60         No loop transformations were applied
61         [loop at nbody.cc:57 in main]
62         Vectorized AVX loop processing Float32; Float64; Int32; UInt32 data t
63         No loop transformations were applied
64
65         if ( j != i ) {
66             real dx = x[j] - x[i], dy = y[j] - y[i], dz = z[j] - z[i];
67             real dist2 = dx*dx + dy*dy + dz*dz;
68             real mOverDist3 = m[j] / ( dist2 * Sqrt( dist2 ) );
69             dvx += mOverDist3 * dx;
70             dvy += mOverDist3 * dy;
71             dvz += mOverDist3 * dz;
72         }
73     }
74 }
    
```

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_8	main	nbody.cc:85	RAW:3	67% / 1% / 32%	Mixed strides
loop_site_7	main	nbody.cc:14	No dependencies found	25% / 75% / 0%	Mixed strides
loop_site_5	main	nbody.cc:20	RAW:3	50% / 50% / 0%	Mixed strides

and ASSEMBLY (low-level).

```

20 for ( int j = 0; j < nparticles; ++j ) {
21     // Avoid singularity and interaction with self
22     const float softening = 1e-20;
23
24     // Newton's law of universal gravity
25     const float dx = particle[j].x - particle[i].x;
26     const float dy = particle[j].y - particle[i].y;
27     const float dz = particle[j].z - particle[i].z;
28 }
    
```

Address	Line	Assembly	Operand Size (bytes)	Stride
0:14037c493	27	vbroadcastss xmm7, dword ptr [rbp+rcx*8+0x1c]	4	12
0:14037c49a	26	vbroadcastss xmm5, dword ptr [rbp+rcx*8+0x18]	4	12
0:14037c4a1	28	vbroadcastss xmm4, dword ptr [rbp+rcx*8+0x8]	4	12
0:14037c4a8	27	vbroadcastss xmm9, dword ptr [rbp+rcx*8+0x4]	4	12
0:14037c4af	26	vbroadcastss xmm13, dword ptr [rbp+rcx*8]	4	12
0:14037c4b6	20	vmovups ymm10, ymmword ptr [rip+0x1542c2]	32	0

2.

Is it safe to vectorize:
Tough problem #1 for not yet
vectorized codes.

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>::ci...]	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop starts were reordered				
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...]	0.000s	0.234s	Scalar	nonstandard loop is not a vectoriza...

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
 Projection 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. This example aligns memory using a 32-byte boundary:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

// Somewhere else
__assume_aligned(array, 32);
// Use array in loop
```

4. Loop-Carried Dependency Analysis

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	

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`

Data Dependencies – Tough Problem #1

Dynamic check will **know** if indices overlap.


```
1) fSwapPairM ( lbf[il*lbsitelength + l*lbsy.nq + m + half],  
               lbf[ilnext*lbsitelength + l*lbsy.nq + m]);
```

Compiler Assumption:

```
i>  [loop at lbpSUB.cpp:1280 in fPropagationSwap]  vector dependence prevents vectorization
```

```
2) fSwapPairM ( lbf[il*lbsitelength + l*lbsy.nq + m + half],  
               lbf[ilnext*lbsitelength + l*lbsy.nq + m]);
```

Compiler Assumption:

```
i>  [loop at lbpSUB.cpp:1280 in fPropagationSwap]  vector dependence prevents vectorization
```

**Both loops “equally bad” :
from static analysis/compilation “best knowledge”**

Data Dependencies – Tough Problem #1

Dynamic check **knows** if memory accesses really overlap.

```
1) fSwapPairM ( lbf[il*lbsitelength + l*lbsy.nq + m + half],  
               lbf[ilnext*lbsitelength + l*lbsy.nq + m]);
```

 [loop at lbpSUB.cpp:1280 in fPropagationSw ...  No dependencies found

```
2) fSwapPairM ( lbf[il*lbsitelength + l*lbsy.nq + m + half],  
               lbf[ilnext*lbsitelength + l*lbsy.nq + m]);
```

 [loop at lbpSUB.cpp:1280 in fPropagationSw ...  RAW:1

 Read after write dependency

Correctness Analysis: confirm dependencies are **REAL**

2.

Any speed-up out of there?
Use SIMD to make your code
faster, instead of slower.

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time
[loop in runCforallLambdaLoops]	0.094s	
[loop in runCforallLambdaLoops]	0.140s	
[loop in std::Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s
Vectorized SSE; SSE2 loop processing Float32; Float64 data type		
Peeled loop: loop stats were reordered		
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	5.000s
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	5.000s
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	0.000s

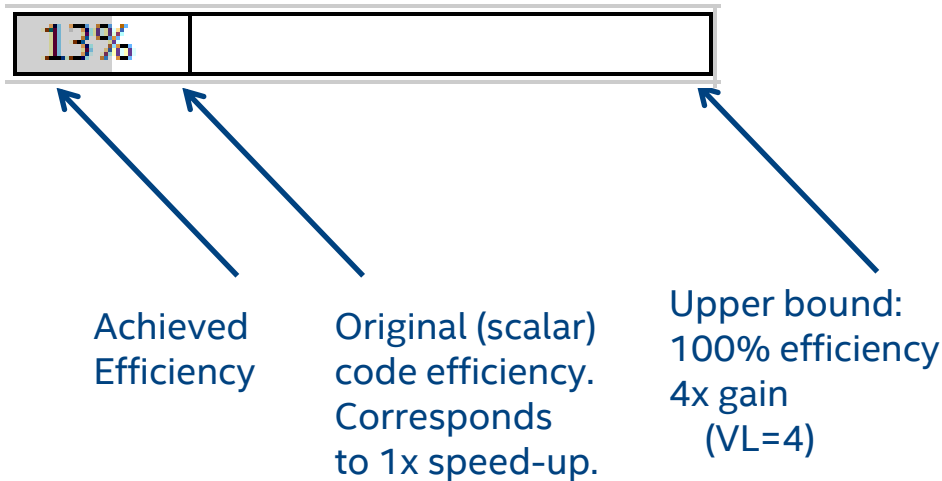
3. "Accurate" Trip Counts: understand parallelism granularity and overheads

Median	Min	Max	Call Count
101	101	101	12000000
3	3	3	1000000
101	101	101	2000000
1000000	1000000	1000000	1

Vector Efficiency: my performance thermometer all the data in one place

Elapsed time: 8,01s

Loops	Vecto...	Efficiency ▲	Estimated Gain	Vect... Co	Traits	Vector Widths	Self Time	
[loop at lbpSUB.cpp:1280 in fPropagationS...	AVX	13%	0,53	4	0,53	Blends; Extracts; Inserts; Shuffles	128/256	2,312s
[loop at lbpGET.cpp:152 in fGetFracSite]	AVX	30%	2,38	8	2,34	Blends; Inserts; Masked Stores	128/256	0,030s
[loop at lbpGET.cpp:42 in fGetOneMassSite]	AVX	36%	2,86	8	2,79		256	0,100s
[loop at lbpGET.cpp:78 in fGetTotMassSite]	AVX	36%	2,86	8	2,79		256	0,010s
[loop at lbpGET.cpp:334 in fGetOneDirecSp ...]	AVX	38%	3,05	8	2,97	Type Conversions	128/256	0,011s
[loop at lbpBGK.cpp:840 in fCollisionBGK]	AVX	100%	2,05	2	2,05		128	0,080s



- **Auto-vectorization:** affected <3% of code
 - With moderate speed-ups
- First attempt to **simply put #pragma simd:**
 - Introduced slow-down
- Look at Vector Issues and Traits to find out **why**
 - All kinds of “memory manipulations”
 - Usually an indication of “bad” access pattern

Survey: find out if your code is “undervectorized” and why

3.

Tough problem #1 for already
vectorized codes

Non-Contiguous Memory – Tough Problem #2

Potential to vectorize but may be inefficient

- Unit-Stride access to arrays

```
for (i=0;i<N;i++)  
    A[i] = C[i]*D[i]; //Accessing array elements 1 by 1
```

- Non-unit-stride (constant stride) access to arrays

```
for (i=0;i<N;i+=2)  
    A[i] = C[i]*D[i]; //Incrementing "i" by 2: not unit stride  
                      //Often indication of demand for AoS ->  
                      // SoA conversion
```

- Indirect reference in a loop

```
for (i=0;i<N;i++)  
    A[B[i]] = C[i]*D[i]; //We have to decode B[i] to find out  
                        //which element of A to reference
```


Object-oriented programming

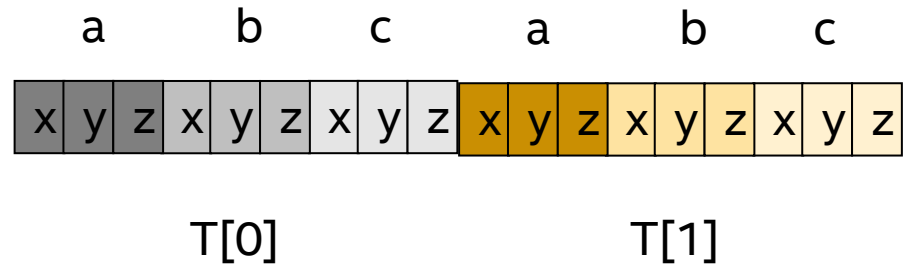
```
Class Point {float  
x,y,z;}
```

```
Class Triangle {Point  
a,b,c;}
```

```
Triangle T[100];
```

```
Point Cross( const Point& a, const Point& b ) {  
    return Point( a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z,  
a.x*a.y-a.y-b.x );  
}
```

```
void ComputeNormals( Point normal[__restrict], const  
Triangle p[], size_t n )  
    for( size_t i=0; i<n; ++i )  
        normal[i] = Cross( p[i].b-p[i].a, p[i].c-p[i].a );  
}
```



Object oriented programming may inhibit SIMD code generation

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"/>	4 Serialized use ...	0,013s 12,020s	Collapse	Collapse
[loop at fractal.cpp:179 in <lambda1>::o ...	<input checked="" type="checkbox"/>	2 Data type co ...	0,013s 11,281s	Vectorized (Body)	
[loop at fractal.cpp:179 in <lambda1>::o ...	<input checked="" type="checkbox"/>	2 Data type co ...	0,000s 0,163s	Peeled	
[loop at fractal.cpp:179 in <lambda1>::o ...	<input checked="" type="checkbox"/>	2 Data type co ...	0,000s 0,576s	Remainder	
[loop at fractal.cpp:177 in <lambda1>::oper ...	<input type="checkbox"/>	2 Data type co ...	0,010s 12,030s	Scalar	

Select loops of interest

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

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>::ci...]	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peel loop; loop starts were reordered				
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...]	0.000s	0.234s	Scalar	nonstandard loop is not a vectoriza...

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. This example aligns memory using a 32-byte boundary:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

// Somewhere else
__assume_aligned(array, 32);
// Use array in loop
```

3. Loop-Carried Dependency Analysis

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

4. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cxx:1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns | **Correctness Report**

ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	
<pre> 635 j2 = (j2 & 64-1) ; 636 p[ip][0] += y[i2+32]; 637 p[ip][1] += z[j2+32]; 638 i2 += e[i2+32]; 639 j2 += f[j2+32]; </pre>					
P23	0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628	lcals.exe	
<pre> 626 i1 &= 64-1; 627 j1 &= 64-1; 628 p[ip][2] += b[j1][i1]; </pre>					

4.

It's time for explicit parallelism
choices to make your code
faster, not slower.

Example of Outer Loop Vectorization

```
#pragma omp declare simd
int lednam(float c)
{ // Compute n >= 0 such that c^n > LIMIT
  float z = 1.0f; int iters = 0;
  while (z < LIMIT) {
    z = z * c; iters++;
  }
  return iters;
}
```

```
float in_vals[];
#pragma omp simd
for(int x = 0; x < Width; ++x) {
  count[x] = lednam(in_vals[x]);
}
```

x = 0

z = z * c

z = z * c

iters = 2

x = 1

z = z * c

z = z * c

.....

iters = 23

x = 2

z = z * c

z = z * c

.....

iters = 255

x = 3

z = z * c

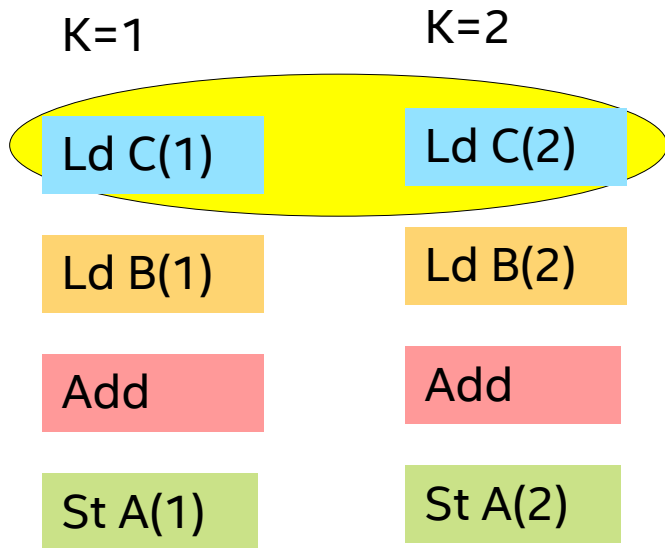
z = z * c

.....

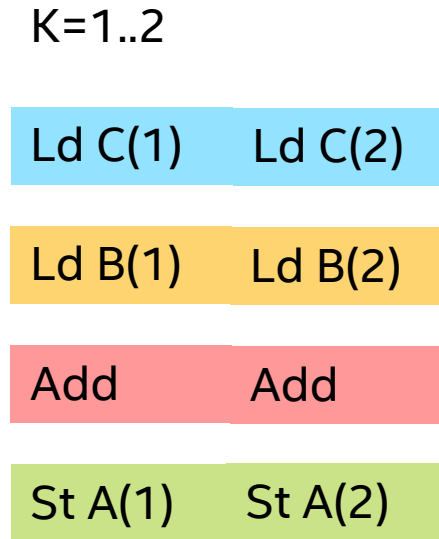
iters = 37

Vectorization yesterday

```
DO 1 k = 1,n  
1  A(k) = B(k) + C(k)
```



Scalar code



Vector code

Vector code generation was straightforward
Emphasis on analysis and disambiguation

Vectorization today

```
#pragma omp simd reduction(+:....)
```

```
for(p=0; p<N; p++) {
```

```
  // Blue work
```

```
  if(...) {
```

```
    // Green work
```

```
  } else {
```

```
    // Red work
```

```
  }
```

```
  while(...) {
```

```
    // Gold work
```

```
    // Purple work
```

```
  }
```

```
  y = foo(x);
```

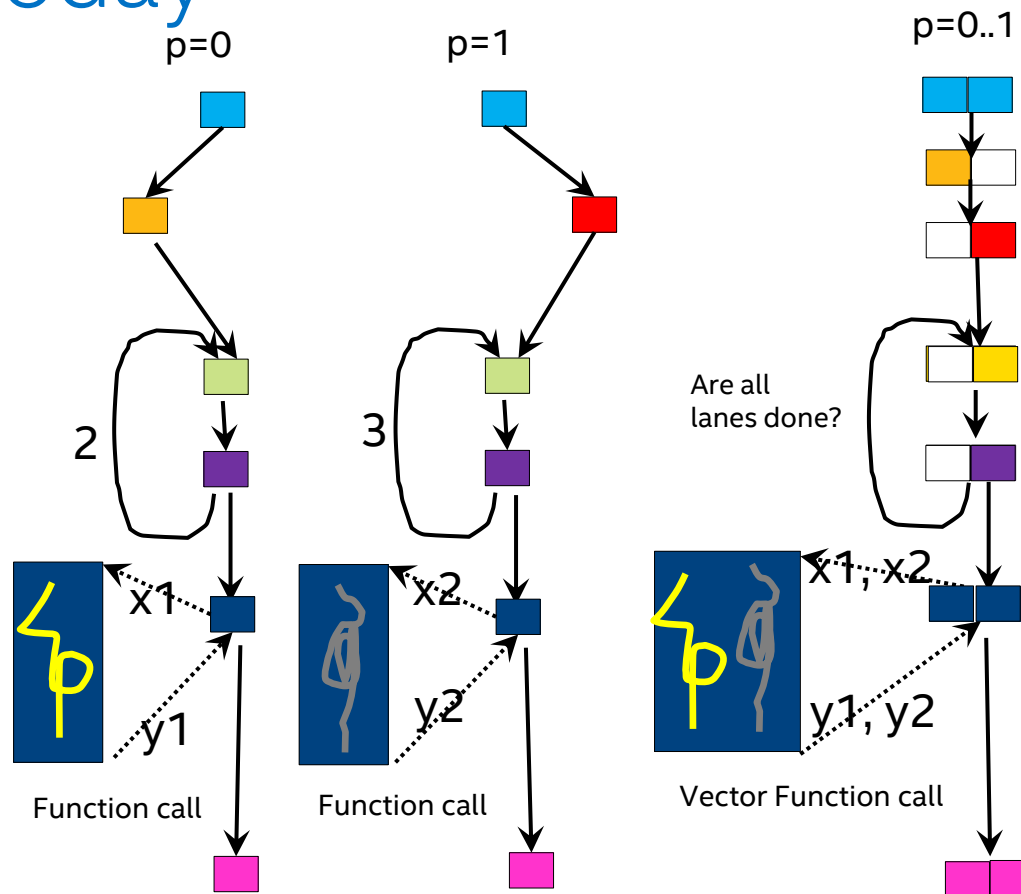
```
  // Pink work
```

```
}
```

Two fundamental problems

Data divergence

Control divergence



Vector code generation has become a more difficult problem
Increasing need for user guided explicit vectorization
Explicit vectorization maps threaded execution to simd hardware

Time for parallelism choices: Where to introduce parallelism and how?

```
for(int i=0; i<Xmax; i++)           ← Here?
  for(int j=0; j<Ymax; j++)
    for(int k=0; k<Zmax; k++) {     ← Here????
      //do some work
      for (int l=0; l<qdim; l++) {   ← Here???
        for (int m=1; m<=half; m++) { ← Here??
          //...
          fSwapPairM (...);
        }
      }
    }
  }
}
```

No performance without “explicit parallelism” choices
(no performance “by default”)
No good choices without knowing “the DATA”

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>::ci...]	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peel loop; loop starts were reordered				
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...]	0.000s	0.234s	Scalar	nonstandard loop is not a vectoriza...

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. This example aligns memory using a 32-byte boundary:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

// Somewhere else
__assume_aligned(array, 32);
// Use array in loop
```

3. Loop-Carried Dependency Analysis

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

4. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cxx:1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	
P23	0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628	lcals.exe	

```

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

```

626         i1 &= 64-1;
627         j1 &= 64-1;
628         p[ip][2] += b[j1][i1];
    
```

Time for parallelism choices: Advisor MAP to make informed optimal decision!

```
for(int i=0; i<Xmax; i++)  
  for(int j=0; j<Ymax; j++)  
    for(int k=0; k<Zmax; k++) {  
      //do some work  
      for (int l=0; l<qdim; l++) {  
        for (int m=1; m<=half; m++) {  
          //...  
          fSwapPairM (...);  
        }  
      }  
    }  
  }  
}
```

Strides Distribution

81% / 12% / 6%

Strides Distribution

26% / 6% / 68%

Memory Access Patterns analysis (+ also Trip Counts)
to drive decision
wrt most appropriate parallelism level

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>::...]	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				
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...]	0.000s	544.0...	Scalar	nonstandard loop is not a vectoriza...
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...]	0.000s	0.234s	Scalar	nonstandard loop is not a vectoriza...

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. This example aligns memory using a 32-byte boundary:

```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

// Somewhere else
__assume_aligned(array, 32);
// Use array in loop
```

Background on loop vectorization

A typical vectorized loop consists of

Main vector body

- Fastest among the three!

Optional peel part

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

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!
- Make the number of iterations divisible by the vector length!

This is where we want our loops to be executing!

Get Specific Advice For Improving Vectorization

Intel® Advisor XE – Vectorization Advisor

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

Summary | Survey Report | Refinement Reports | Annotation Report | Suitability Report

Elapsed time: 8,81s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops		
						Vecto...	Estim...	Vector Len
[loop at market...]			11,460s	Scalar				
[loop at arena.cpp:88 in tbb::tbb::...]		0,000s	11,460s	Scalar				
[loop at fractal.cpp:179 in <lambda1>::op ...]	5 Ineffective ...	0,000s	2,022s	Collapse	Collapse			
[loop at fractal.cpp:179 in <lambda1>::o ...]	2 Data type co ...	0,000s	2,022s	Remainder				

Click to see recommendation

Top Down | Source | Loop Assembly | Assistance | 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.

Disable unrolling
The [trip count](#) after loop unrolling is too small compared to [factor](#) using a [directive](#).

ICL/ICC/ICPC Directive	IFORT Directive
#pragma nounroll	!DIR\$ NOUNROLL
#pragma unroll	IDIR\$ UNROLL

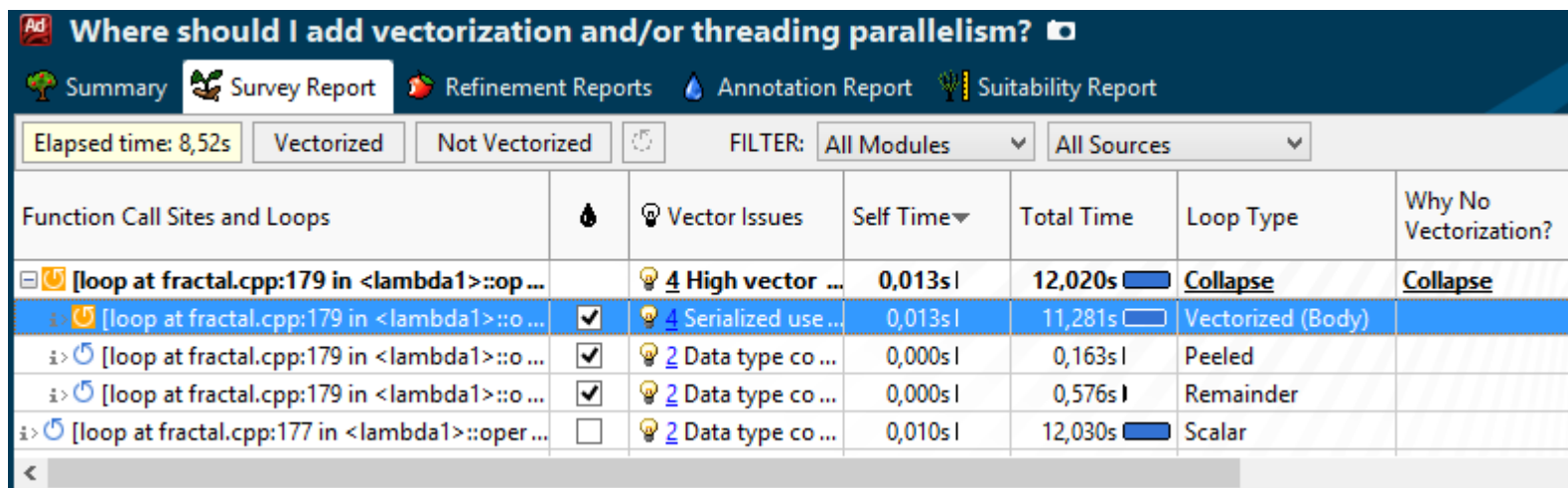
Read More:

- [User and Reference Guide for the Intel C++ Compiler 15.0 > Compiler Reference > Pragmas > Intel-specific Pragma Reference > unroll/nounroll.](#)

Advisor XE shows hints to move iterations to vector body.

Don't Just Vectorize, Vectorize Efficiently

See detailed times for each part of your loops. Is it worth more effort?



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

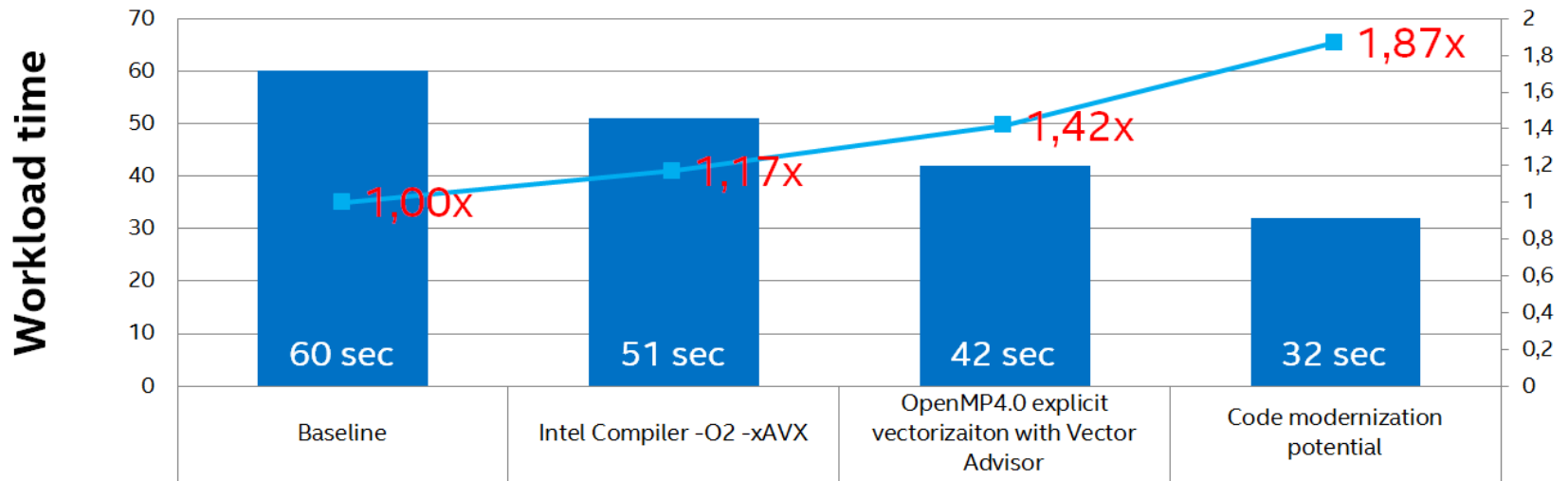
Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?
[loop at fractal.cpp:179 in <lambda1>::op ...]	4 High vector ...	0,013s	12,020s	Collapse	Collapse
[loop at fractal.cpp:179 in <lambda1>::o ...]	4 Serialized use ...	0,013s	11,281s	Vectorized (Body)	
[loop at fractal.cpp:179 in <lambda1>::o ...]	2 Data type co ...	0,000s	0,163s	Peeled	
[loop at fractal.cpp:179 in <lambda1>::o ...]	2 Data type co ...	0,000s	0,576s	Remainder	
[loop at fractal.cpp:177 in <lambda1>::oper ...]	2 Data type co ...	0,010s	12,030s	Scalar	

Vectorization Advisor: use cases



DL-MESO : major CFD package for cross-UK industrial consortium : new chemical products development (led by STFC Hartree/Daresbury)

DL-MESO optimization use case



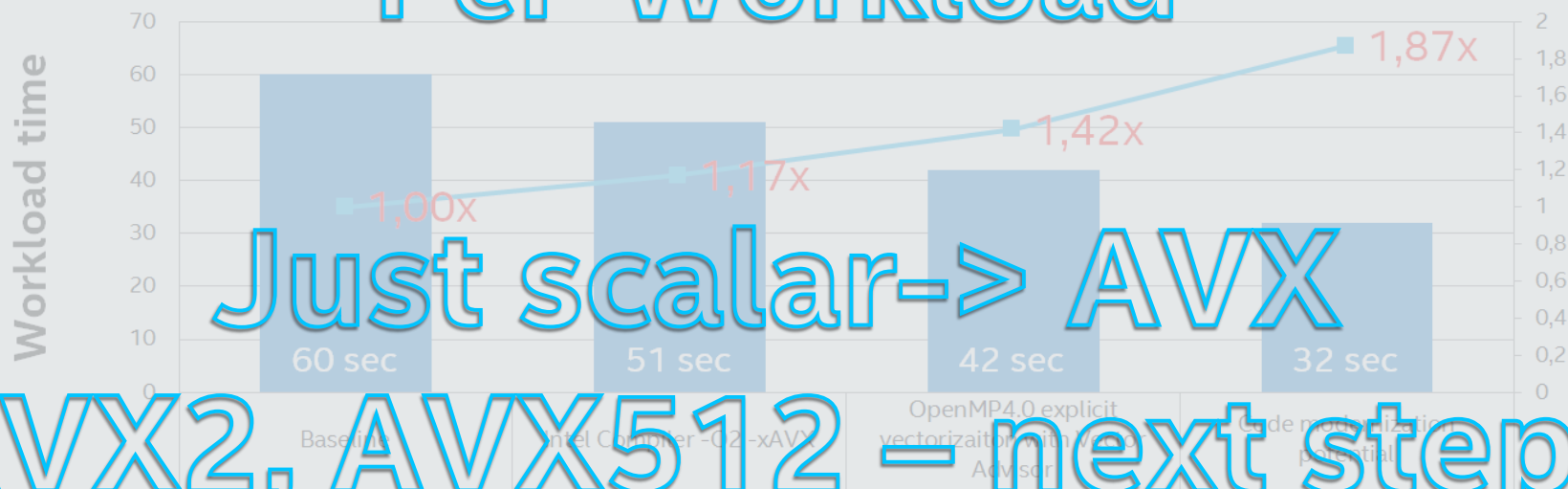
Vectorization Advisor: use cases



DL-MESO : major CFD package for cross-UK industrial consortium : new chemical products development (led by STFC Hartree/Daresbury)

DL-MESO Optimization Use Cases

Per-Workload



Just scalar -> AVX

AVX2, AVX512 - next steps

Intel® Parallel Studio XE 2016 Beta

What's New

Intel® Parallel Studio XE 2016 Suites

Vectorization – Boost Performance By Utilizing Vector Instructions / Units

- Intel® Advisor XE - **Vectorization Advisor** identifies new vectorization opportunities as well as improvements to existing vectorization and highlights them in your code. It makes actionable coding recommendations to boost performance and estimates the speedup.

Scalable MPI Analysis – Fast & Lightweight Analysis for 32K+ Ranks

- Intel® Trace Analyzer and Collector add **MPI Performance Snapshot** feature for easy to use, scalable MPI statistics collection and analysis of large MPI jobs to identify areas for improvement

Big Data Analytics – Easily Build IA Optimized Data Analytics Application

- Intel® Data Analytics Acceleration Library (Intel® DAAL) will help data scientists speed through big data challenges with optimized IA functions

Standards – Scaling Development Efforts Forward

- Supporting the evolution of industry standards of **OpenMP***, **MPI**, **Fortran** and **C++** Intel® Compilers & performance libraries

Intel® Advisor XE – New! Vectorization Advisor

Data Driven Vectorization Design TO REWORK

Have you:

- Recompiled with AVX2, but seen little benefit?
- Wondered where to start adding vectorization?
- Recoded intrinsics for each new architecture?
- Struggled with cryptic compiler vectorization messages?

Breakthrough for vectorization design

- What vectorization will pay off the most?
- What is blocking vectorization and why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?

The screenshot shows the Intel Advisor XE 2016 Vectorization Advisor interface. The top window displays a table of function call sites and loops. The table has columns for Function Call Sites and Loops, Vector Issues, Self Time, Total Time, Trip Counts, Loop Type, Why No Vectorization?, and Vectorized Loops (Vector..., Efficiency, Vector L...). The table lists several loops, including those at `loopstl.cpp:4740`, `loopstl.cpp:2449`, `loopstl.cpp:2900`, `loopstl.cpp:3509`, `loopstl.cpp:3891`, `loopstl.cpp:6249`, and `stl_numeric.h:247`. The `loopstl.cpp:3509` loop is highlighted in blue, indicating it is vectorized. The `loopstl.cpp:3509` loop is highlighted in blue, indicating it is vectorized. The `loopstl.cpp:3509` loop is highlighted in blue, indicating it is vectorized.

The bottom window shows the source code for `loopstl.cpp:3509`. The code is as follows:

```
3504 fortime_(stl);
3505 i_1 = *ntimes;
3506 for (nl = 1; nl <= i_1; ++nl)
    [loop at loopstl.cpp:3506 in s273_]
    Scalar loop. Not vectorized: inner loop was already vectorized
    No loop transformations were applied
3507 {
3508     i_2 = *n;
3509     for (i_1 = 1; i_1 <= i_2; ++i_1)
        [loop at loopstl.cpp:3509 in s273_]
        Vectorized AVX Loop processing Float32; Float64; Int32 data type(s) having Inserts; Extracts; Mapped Str
        Selected (Total Time): 0.010s
```

More Performance
Fewer Machine Dependencies

Intel® Advisor XE – Vectorization Advisor

Provides the data you need for high impact vectorization

Compiler diagnostics + Performance Data = All the data you need in one place

- Find “hot” un-vectorized or “under vectorized” loops.
- Trip counts

Recommendations – How do I fix it?

Correctness via dependency analysis

- Is it safe to vectorize?

Memory Access Patterns analysis

- Unit stride vs Non-unit stride access, Unaligned memory access, etc.

Intel® C/C++ and Fortran Compilers 16.0

Get best performance with latest standards

Standards:

- More of C++14, generic lambdas, member initializers and aggregates
- More of C11, `_Static_assert`, `_Generic`, `_Noreturn`, and more
- OpenMP 4.0 C++ User Defined Reductions, Fortran Array Reductions

Vectorization:

- OpenMP 4.1 asynchronous offloading, `simdlen`, `simd ordered`
- Significant improvement in alignment analysis, vectorization robustness
- Much improved Neighboring Gather optimization

Fortran:

- F2008 Submodules, Impure Elemental Functions
- F2015 `TYPE(*)`, `DIMENSION(..)`, `RANK` intrinsic, attributes for args with `BIND`

Intel® Math Kernel Library (Intel® MKL) 11.3

Better performance with new two-stage API for Sparse BLAS routines

Additional Sparse Matrix Vector Multiplication API

- new two-stage API for Sparse BLAS level 2 and 3 routines

MKL MPI wrappers

- all MPI implementations are API-compatible but MPI implementations are **not ABI-compatible**
- MKL MPI wrapper solves this problem by providing an MPI-independent ABI to MKL

Support For Batched Small Matrix multiplication

- a single call executes multiple independent GEMM operation simultaneously

Support for Philox4x35 and ARS5 RNG

- two new pseudorandom number generators with a period of 2^{128} are highly optimized for multithreaded environment

Sparse Solver SMP improvements

- significantly improved overall scalability for Intel Xeon Phi coprocessors and Intel Xeon processors

Intel® Data Analytics Acceleration Library 2016

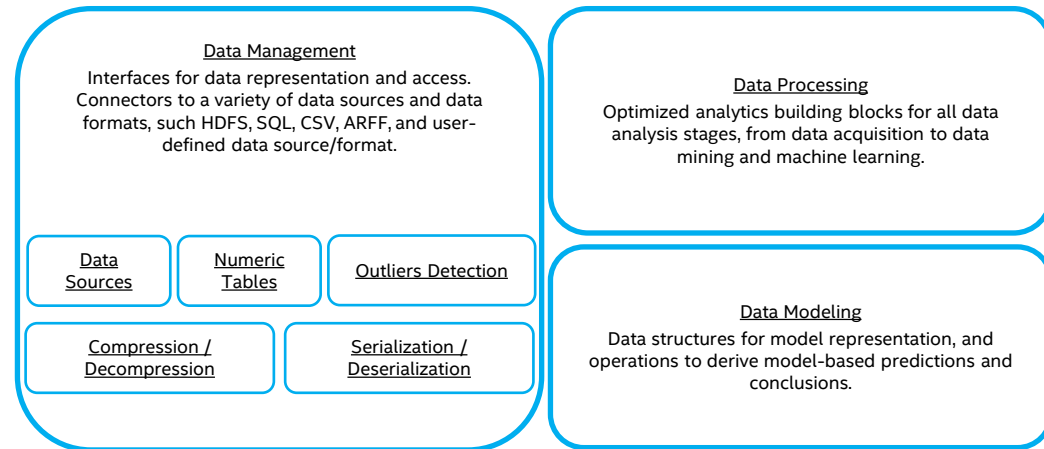
A C++ and Java API library of optimized analytics building blocks for all data analysis stages, from data acquisition to data mining and machine learning. Essential for engineering high performance Big Data applications.

New library targeting data analytics market

- **Customers:** analytics solution providers, system integrators, and application developers (FSI, Telco, Retail, Grid, etc.)
- **Key benefits:** improved time-to-value, forward-scaling performance and parallelism on IA, advanced analytics building blocks

Key features

- Building blocks highly optimized for IA to support all data analysis stages
- Support batch, streaming, and distributed processing with easy connectors to popular platforms (Hadoop, Spark) and tools (R, Python, Matlab)
- Flexible interfaces for handling different data sources (CSV, MySQL, HDFS, RDD (Spark))
- Rich set of operations to handle sparse and noisy data
- C++ and Java APIs



Important features offered in the initial Beta

Analysis

- PCA
- Variance-Covariance Matrix
- Distances
- Matrix decompositions (SVD, QR, Cholesky)
- EM for GMM
- Uni-/multi-variate outlier detection
- Statistical moments

Machine learning

- Linear regression
- Apriori
- K-Means clustering
- Naïve Bayes
- LogitBoost, BrownBoost, AdaBoost
- SVM

- Data layouts: AOS, SOA, homogeneous, CSR
- Data sources: csv, MySQL, HDFS/RDD
- Compression/decompression: ZLIB, LZO, RLE, BZIP2
- Serialization/deserialization

Intel® VTune™ Amplifier XE 2016 Beta

Enhanced GPU and Microarchitecture Profiling

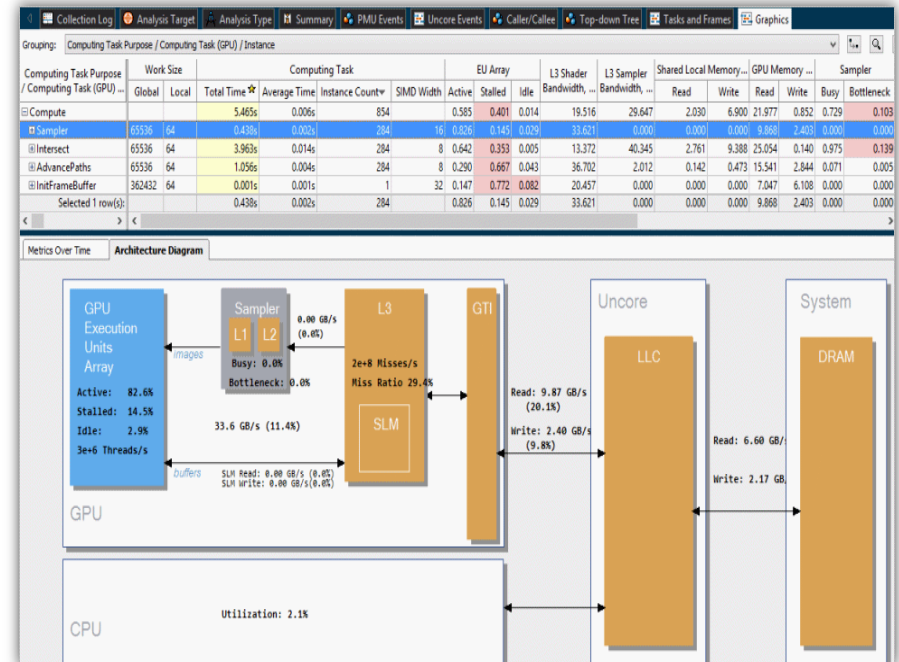
New OS and IDE support: Visual Studio* 2015 & Windows* 10 Threshold

Intel® HD Graphics (GPU) profiling

- GPU Architecture Annotation Diagram
- GPU profiling on Linux (OpenCL, Media SDK)

Microarchitecture tuning

- General Exploration analysis with confidence indication
- Driverless 'perf' EBS with stacks



Intel® VTune™ Amplifier XE 2016 Beta Improved OpenMP* and Hybrid Support

Intel OpenMP analysis enhancements

- Precise trace-based imbalance calculation that is especially useful for profiling of small region instances
- Classification and issue highlighting of potential gains, e.g., imbalance, lock contention, creation overhead, etc.
- Detailed analysis of barrier-to-barrier region segments

OpenMP Region / Function / Call Stack	OpenMP Potential Gain						OpenMP Potential Gain (% of Collection Time)						Elapsed Time	Number of OpenMP threads	Inst. Count
	Imbalance	Lock Con...	Creation	Scheduling	Reduc...	Other	Imbalance (%)	Lock Con... (%)	Creation (%)	Scheduling (%)	Red... (%)	Other (%)			
@@@_grad_Somp\$parallel24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f514695	0.20%	0s	0.00	3.12%	0s	0.00%	0.6%	0.0%	25.9%	0.0%	0.0%	11.75s	24	75	
@@@_Somp\$parallel24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f185231	0.07%	0.000%	0s	0s	0s	0.00%	0.6%	0.0%	0.0%	0.0%	0.0%	0.28%	24	1	
@@@Serial - outside any region												0.0%	0.01%		
@@@_Somp\$parallel24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f339345	0.00%	0s	0s	0s	0s	0.00%	0.0%	0.0%	0.0%	0.0%	0.0%	0.00%	24	75	
@@@_Somp\$parallel24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f361365	0.00%	0s	0s	0s	0s	0.00%	0.0%	0.0%	0.0%	0.0%	0.0%	0.00%	24	75	

Dynamic scheduling overhead

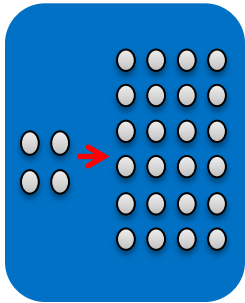
MPI+OpenMP: multi-rank analysis on a compute node

- Per-rank OpenMP potential gain and serial time metrics
- Per-rank Intel MPI communication busy wait time detection

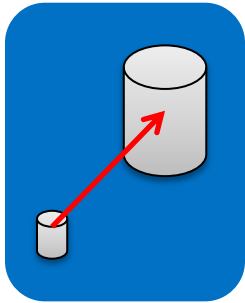
OpenMP Region / Function / Call Stack	OpenMP Potential Gain						OpenMP Potential Gain (% of Collection Time)						Elapsed Time	Number of OpenMP threads	Inst. Count
	Imbalance	Lock Con...	Creation	Sch...	Red...	Other	Imbalance (%)	Lock Con... (%)	Creation (%)	Sch... (%)	Red... (%)	Other (%)			
@@@_grad_Somp\$parallel24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f514695	3.04%	0s	0.00%	0.00%	0.00%	0.00%	34.6%	0.0%	0.0%	0.0%	0.1%	11.00%	24	76	
@@@_Somp\$parallel24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f185231	0.08%	0s	0s	0s	0s	0.00%	0.8%	0.0%	0.0%	0.0%	0.0%	0.28%	24	1	
@@@Serial - outside any region												0.0%	0.02%		
@@@_Somp\$parallel24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f339345	0.00%	0s	0s	0s	0s	0.00%	0.0%	0.0%	0.0%	0.0%	0.0%	0.00%	24	75	
@@@_Somp\$parallel24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f361365	0.00%	0s	0s	0s	0s	0.00%	0.0%	0.0%	0.0%	0.0%	0.0%	0.00%	24	75	
@@@_Somp\$parallel24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f263269	0.00%	0s	0s	0s	0s	0.00%	0.0%	0.0%	0.0%	0.0%	0.0%	0.00%	24	1	

MPI Performance Snapshot

Scalable profiling for MPI and Hybrid



Lightweight – Low overhead profiling for 32K+ Ranks



Scalability- Performance variation at scale can be detected sooner



Identifying Key Metrics – Shows PAPI counters and MPI/OpenMP imbalances

MPI Performance Snapshot Summary

Application: ./hybrid
Ranks: 4
Used statistics: pcs_r4_f020.txt

Overview

MPI Time: 14.70 sec	48.99%
MPI Imbalance: 14.69 sec	48.98%
Computation Time: 15.30 sec	50.98%
OpenMP Time: 14.90 sec	49.66%
OpenMP Imbalance: 7.37 sec	24.57%
Serial Time: 0.40 sec	1.32%



Memory Usage

Peak memory consumption (rank 1):	0.79 MB
Mean memory consumption:	0.75 MB

Per process memory usage affects the application scalability.

Performance by Metric

WallClock time: 30.00 sec
Total application lifetime. The time is elapsed time for the slowest process. This metric is the sum of the MPI Time and the Computation time below.

MPI Time: 14.70 sec **48.99%**
Time spent inside the MPI library. High values are usually bad. This value is **HIGH**. The application is **Communication-bound**. [More details...](#)

MPI Imbalance: 14.69 sec **48.98%**
Mean unproductive wait time per-process spent in the MPI library calls when a process is waiting for data. This time is part of the MPI time above. High values are usually bad. This value is **HIGH**. The application workload is **NOT well balanced** between MPI ranks. [More details...](#)

Computation Time: 15.30 sec **50.98%**
Mean time per-process spent in the application code. This is the sum of the OpenMP Time and the Serial time. High values are usually good. This value is **AVERAGE**. The application is **Computation-bound**. [More details...](#)

OpenMP Time: 14.90 sec **49.66%**
Mean time per process spent in the OpenMP parallel regions. High values are usually good and indicate that the application is well-threaded. This value is **AVERAGE**.

OpenMP Imbalance: 7.37 sec **24.57%**
Mean unproductive wait time per-process spent in OpenMP parallel regions (normally at synchronization barriers). High values are usually bad. This value is **HIGH**. The application's OpenMP work sharing is **NOT well load-balanced**. [More details...](#)

Serial Time: 0.40 sec **1.32%**
Mean application time per-process spent outside OpenMP parallel regions. High values may be good or bad depending on the application algorithm. This value is **NEGLECTIBLE**. This application is **well parallelized** via OpenMP directives.

Summary/Call to Action

Participate in the Beta Program today!

- Register at **bit.ly/psxe2016beta**
- Or simply send e-mail to **vector_advisor@intel.com**

Submit Feedback via Intel® Premier Support

Tell us about your experiences using the Intel® Parallel Studio XE 2016 Beta



Additional Resources

All links start with: <https://software.intel.com/>

Learn more about Vectorization Advisor:

<https://software.intel.com/en-us/articles/vectorization-advisor-faq>

<https://software.intel.com/en-us/intel-advisor-xe>

Vectorization Guide:

<https://software.intel.com/articles/a-guide-to-auto-vectorization-with-intel-c-compilers/>

Explicit Vector Programming in Fortran:

<https://software.intel.com/articles/explicit-vector-programming-in-fortran>

Optimization Reports:

<https://software.intel.com/videos/getting-the-most-out-of-the-intel-compiler-with-new-optimization-reports>

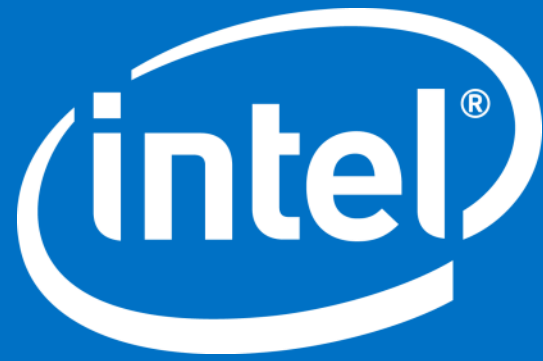
Beta Registration & Download:

<https://software.intel.com/en-us/articles/intel-parallel-studio-xe-2016-beta>

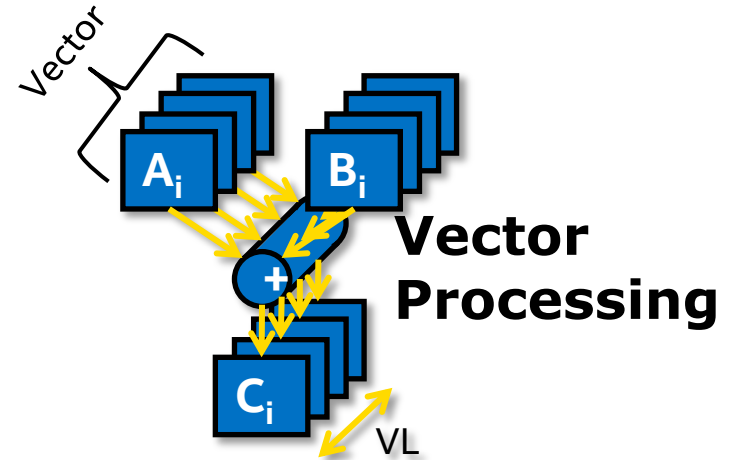
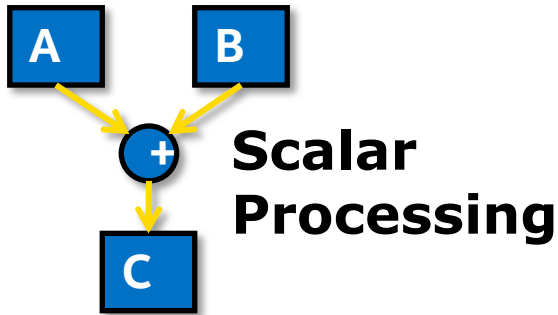
For Intel® Xeon Phi™ coprocessors, but also applicable:

<https://software.intel.com/en-us/articles/vectorization-essential>

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



Recap



AVX: Adding
2 vectors (SP)

	4.4	1.1	3.1	-8.5	-1.3	1.7	7.5	5.6
+	-0.3	-0.5	0.5	0	0.1	0.8	0.9	0.7
=	4.1	0.6	3.6	-8.5	-1.2	2.5	8.4	6.3

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 © 2015v, 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