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

Zakhar A. Matveev, PhD,

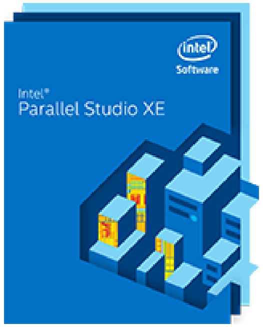
Intel Russia, Intel Software and Services Group

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

Launching August 2015

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

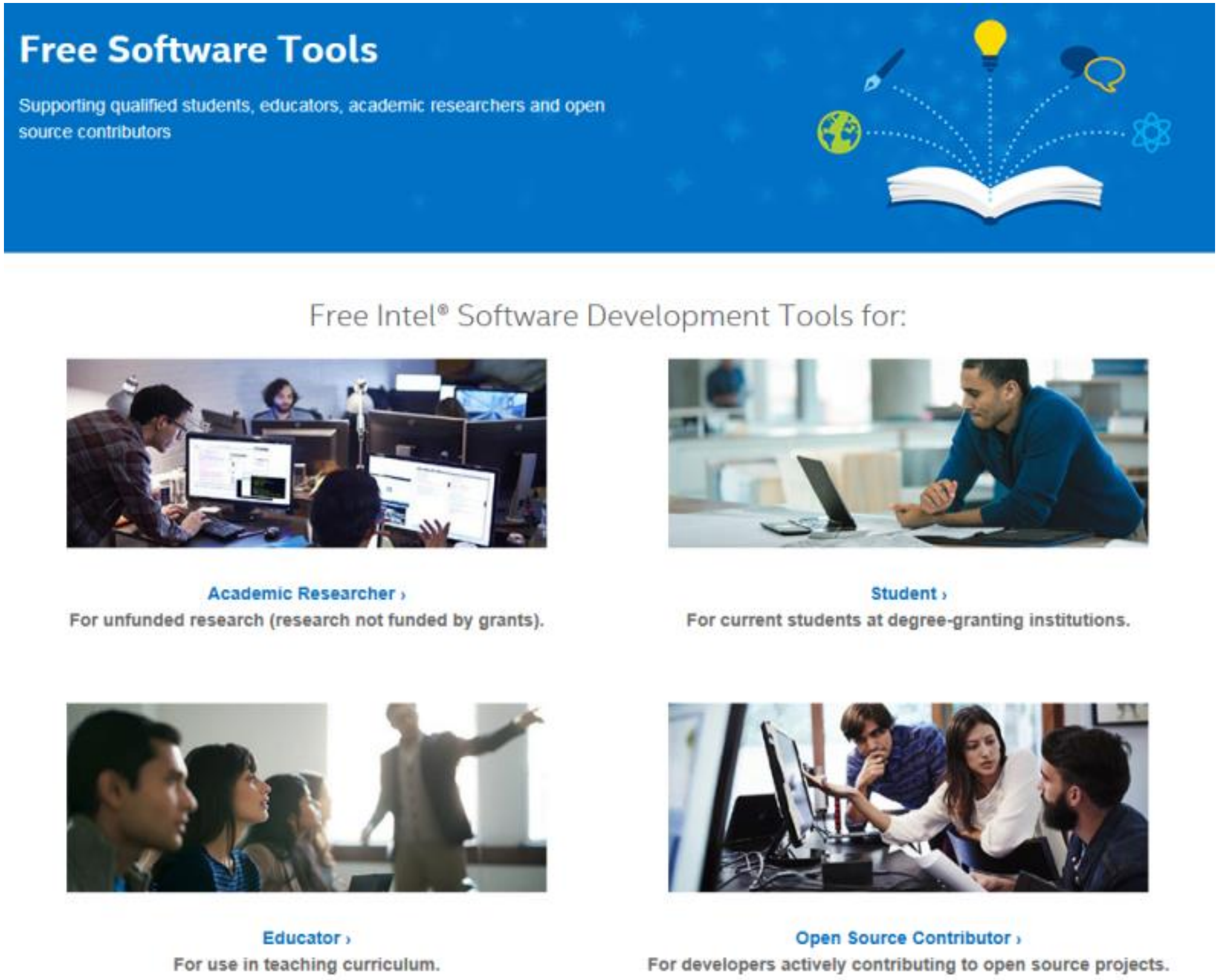
- Intel® Data Analytics Acceleration Library (Intel® DAAL) will help data scientists speed through big data challenges with optimized IA functions

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.

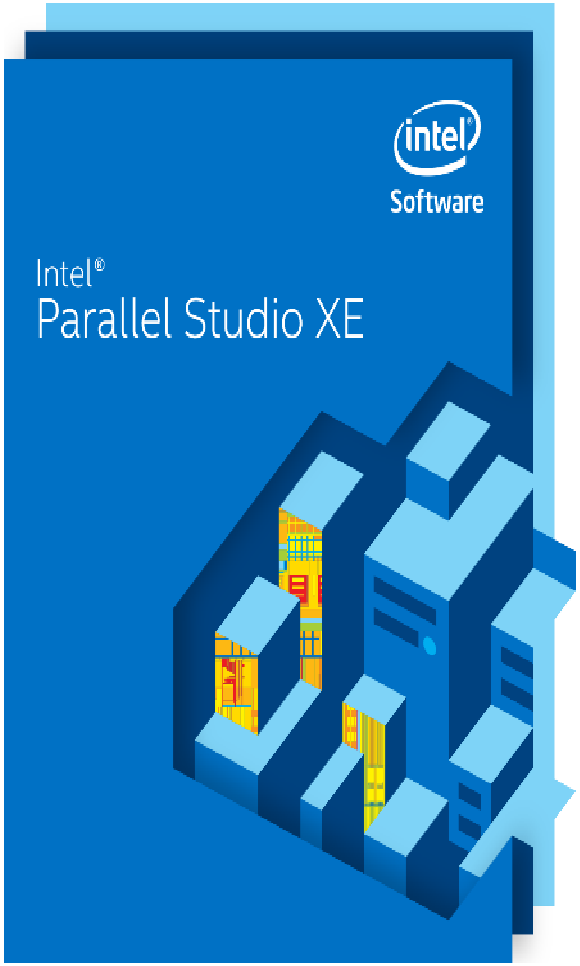
Beta 2016 program : Call to Action

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



Coming in 16.x

Intel® Advisor XE

Vectorization Optimization and Thread Prototyping

SIMD Programming 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”

- Vectorization productivity problem: “thousands of loops”
- Too much raw info (static and dynamic) to drive informed code modernization **decisions**
  - Where to vectorize?
  - How to get more benefit from vectorization
- Demand for extensive data layout re-organizations

Developers need an assistant tool to get applications vectorized faster, with higher efficiency and confidence

“Vectorization Advisor” – Advisor XE

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

*Leverages **Intel Compiler** opt-report+ and **dynamic profile**. Support for other compilers, C, C++, Fortran, for MPI env.*

2. Detects “hot” un-vectorized or “under vectorized” loops.

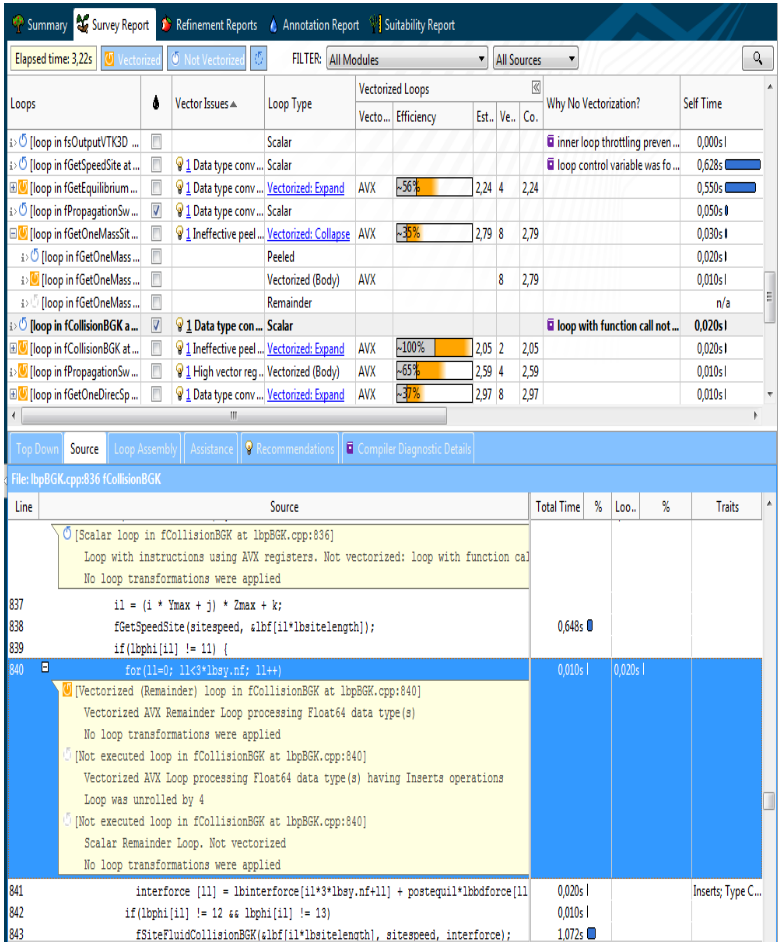
Identifies what is blocking efficient vectorization, where to add it

3. Identify performance penalties and recommend fixes

*Explicit **advices** with “true intelligence”, covering OpenMP4.x.*

4. Memory layout analysis

5. Increase the confidence that vectorization is safe



Loops		Vector Issues ▲	Loop Type	Vectorized Loops					Why No Vectorization?	Self Time
				Vecto...	Efficiency	Est..	Ve..	Co.		
[loop in fsOutputVTK3D ...	☐		Scalar						inner loop throttling preven...	0,000s
[loop in fGetSpeedSite at ...	☐	1 Data type conv...	Scalar						loop control variable was fo...	0,628s
[loop in fGetEquilibrium ...	☐	1 Data type conv...	Vectorized: Expand	AVX	~56%	2,24	4	2,24		0,550s
[loop in fPropagationSw ...	☑	1 Data type conv...	Scalar							0,050s
[loop in fGetOneMassSit ...	☐	1 Ineffective peel ...	Vectorized: Collapse	AVX	~35%	2,79	8	2,79		0,030s
[loop in fGetOneMass ...	☐		Peeled							0,020s
[loop in fGetOneMass ...	☐		Vectorized (Body)	AVX			8	2,79		0,010s
[loop in fGetOneMass ...	☐		Remainder							n/a
[loop in fCollisionBGK a ...	☑	**1 Data type con ...**	**Scalar**						**loop with function call not ...**	**0,020s**
[loop in fCollisionBGK at ...	☐	1 Ineffective peel ...	Vectorized: Expand	AVX	~100%	2,05	2	2,05		0,020s
[loop in fPropagationSw ...	☐	1 High vector reg..	Vectorized (Body)	AVX	~65%	2,59	4	2,59		0,010s
[loop in fGetOneDirecSp ...	☐	1 Data type conv ...	Vectorized: Expand	AVX	~37%	2,97	8	2,97		0,010s

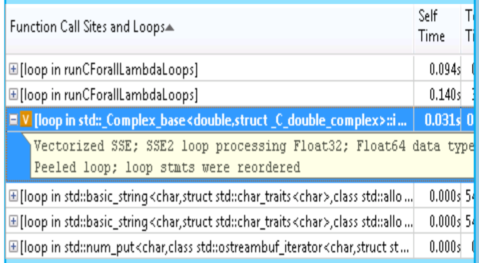
File: lbpBGK.cpp:836 fCollisionBGK

```
Line  Source                                                                      Total Time  %  Loo..   %  Traits
        [Scalar loop in fCollisionBGK at lbpBGK.cpp:836]
          Loop with instructions using AVX registers. Not vectorized: loop with function cal
          No loop transformations were applied
837           il = (i * Ymax + j) * Zmax + k;
838           fGetSpeedSite(sitespeed, &lbf[il*lbsitelength]);                       0,648s
839           if(lbphi[il] != 11) {
840             for(ll=0; ll<3*lbsy.nf; ll++)                                         0,010s     0,020s
        [Vectorized (Remainder) loop in fCollisionBGK at lbpBGK.cpp:840]
          Vectorized AVX Remainder Loop processing Float64 data type(s)
          No loop transformations were applied
        [Not executed loop in fCollisionBGK at lbpBGK.cpp:840]
          Vectorized AVX Loop processing Float64 data type(s) having Inserts operations
          Loop was unrolled by 4
        [Not executed loop in fCollisionBGK at lbpBGK.cpp:840]
          Scalar Remainder Loop. Not vectorized
          No loop transformations were applied
841               interforce [ll] = lbinterforce[il*3*lbsy.nf+ll] + postequil*lbbdforce[ll  0,020s         Inserts; Type C...
842             if(lbphi[il] != 12 && lbphi[il] != 13)                                0,010s
843               fSiteFluidCollisionBGK(&lbf[il*lbsitelength], sitespeed, interforce);  1,072s
```

Vectorization Advisor.

Assist code modernization for x86 SIMD

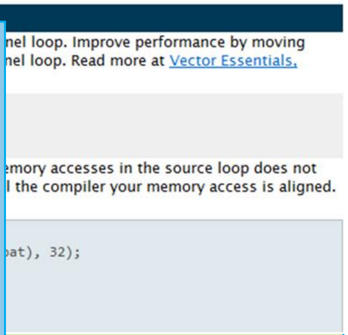
1. Compiler diagnostics + Performance Data + SIMD efficiency information



Function Call Sites and Loops▲	Self Time	T Ti
⊞[loop in runCForallLambdaLoops]	0.094s	(
⊞[loop in runCForallLambdaLoops]	0.140s	3
⊟ V [loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0
⊞[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo ...	0.000s	54
⊞[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo ...	0.000s	54
⊞[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	(

```
Vectorized SSE; SSE2 loop processing Float32; Float64 data type
Peeled loop; loop stmts were reordered
```

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



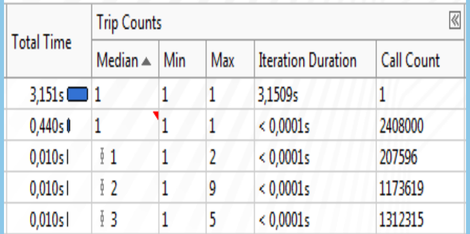
⚠ 2 Issue: Peeled/Remainder loop(s) present

nel loop. Improve performance by moving
nel loop. Read more at Vector Essentials.

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

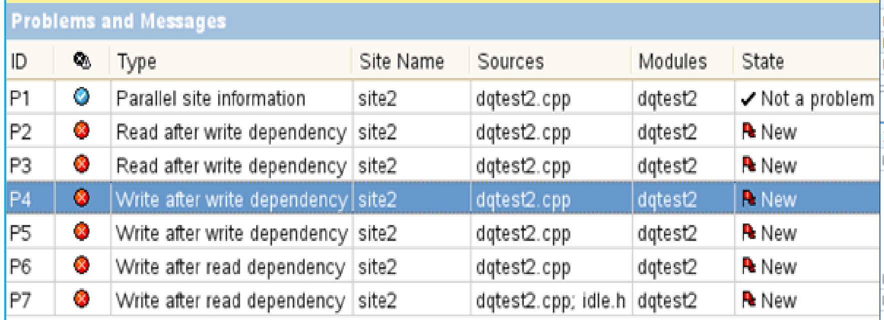
```
pat), 32);
```

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



Total Time	Trip Counts				
	Median ▲	Min	Max	Iteration Duration	Call Count
3,151s	1	1	1	3,1509s	1
0,440s	1	1	1	< 0,0001s	2408000
0,010s	1	1	2	< 0,0001s	207596
0,010s	2	1	9	< 0,0001s	1173619
0,010s	3	1	5	< 0,0001s	1312315

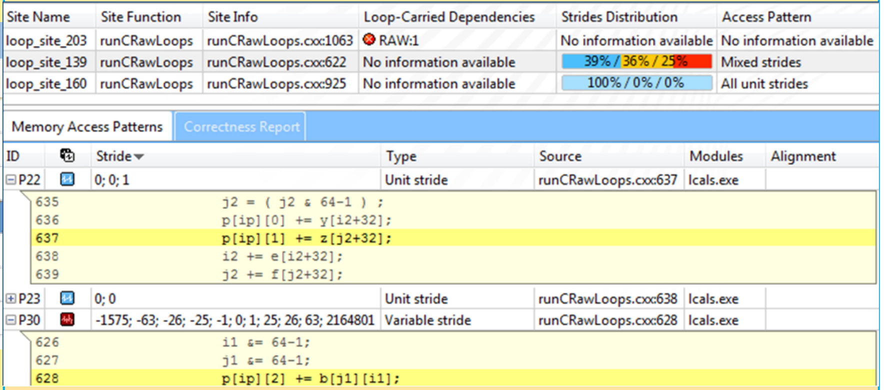
4. Loop-Carried Dependency Analysis



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	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	

P22:

```
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];
```

P30:

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

"**Vectorization Advisor** permitted me to focus my work where it really mattered. When you have only a limited amount of time to spend on optimization, it is invaluable."

Gilles Civario,
Sr. Software Architect,
Irish Centre for High-End Computing

"**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

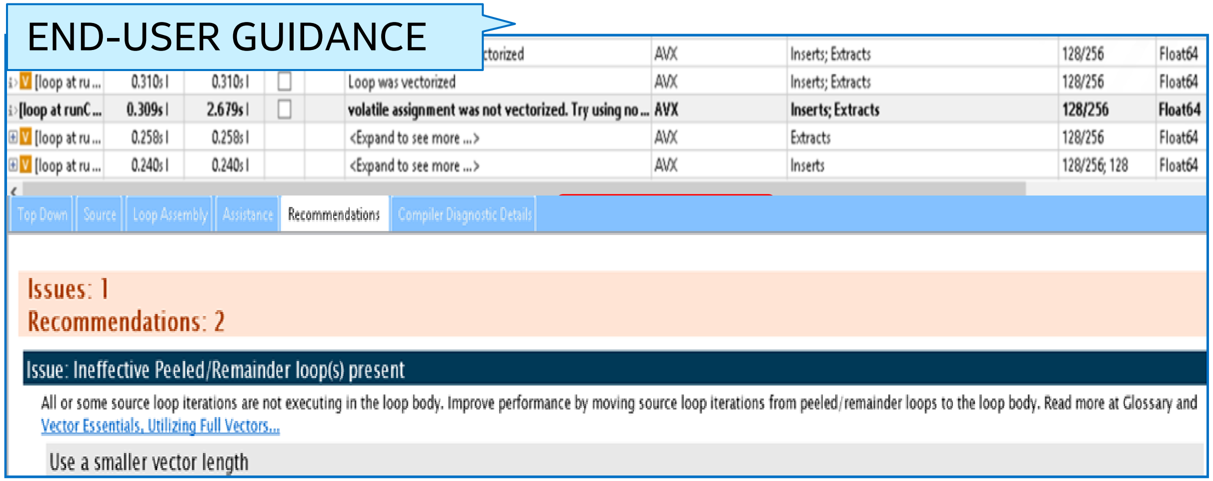
"**Intel® Advisor XE** has allowed us to quickly prototype ideas for parallelism, saving developer time and effort, and has already been used to highlight subtle parallel correctness issues in complex multi-file, multi-function algorithms."

Simon Hammond
Senior Technical Staff
Sandia National Laboratories

"**Intel® Advisor XE** has been extremely helpful in identifying the best pieces of code for parallelization. We can save several days of manual work by targeting the right loops. At the same time, we can use Advisor to find potential thread safety issues to help avoid problems later on."

Carlos Boneti
HPC software engineer, ***Schlumberger***

Assist user at different LoD and perspectives



END-USER GUIDANCE

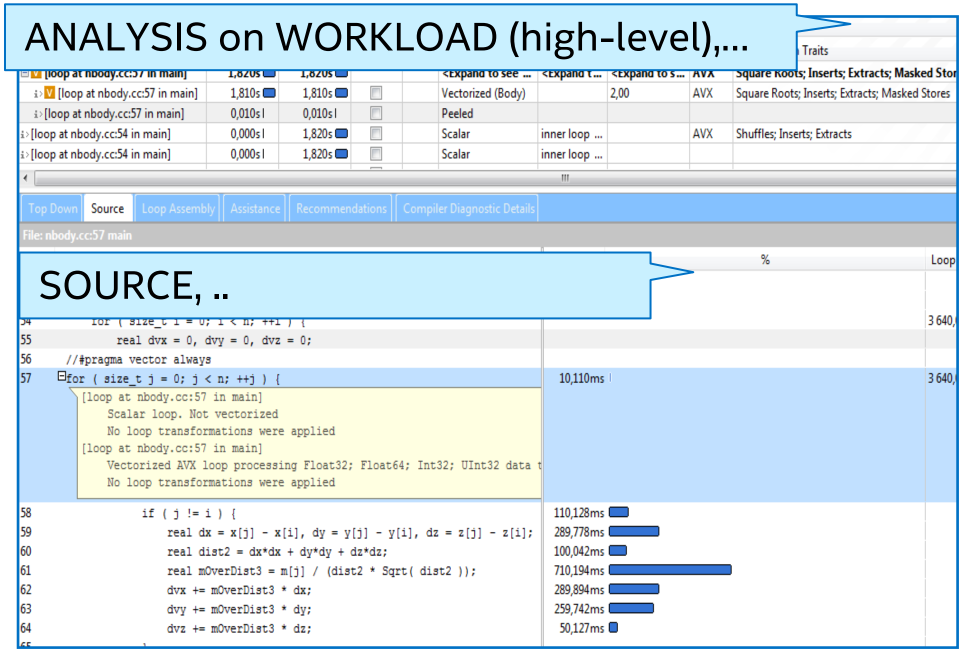
				ctorized	AVX	Inserts; Extracts	128/256	Float64
[loop at ru...	0.310s	0.310s	□	Loop was vectorized	AVX	Inserts; Extracts	128/256	Float64
[loop at runC...	**0.309s**	**2.679s**	□	**volatile assignment was not vectorized. Try using no...**	**AVX**	**Inserts; Extracts**	**128/256**	**Float64**
[loop at ru...	0.258s	0.258s		<Expand to see more ...>	AVX	Extracts	128/256	Float64
[loop at ru...	0.240s	0.240s		<Expand to see more ...>	AVX	Inserts	128/256; 128	Float64

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



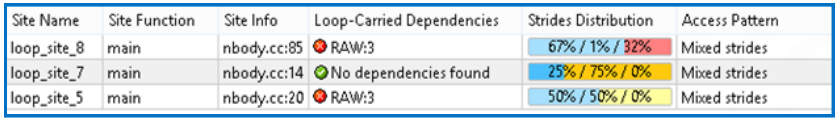
ANALYSIS on WORKLOAD (high-level),...

								Traits
[loop at nbody.cc:57 in main]	1,820s	1,820s		<Expand to see ...	<Expand t...	<Expand to s...	AVX	Square Roots; Inserts; Extracts; Masked Stor
[loop at nbody.cc:57 in main]	1,810s	1,810s	□	Vectorized (Body)		2,00	AVX	Square Roots; Inserts; Extracts; Masked Stores
[loop at nbody.cc:57 in main]	0,010s	0,010s	□	Peeled				
[loop at nbody.cc:54 in main]	0,000s	1,820s	□	Scalar	inner loop ...		AVX	Shuffles; Inserts; Extracts
[loop at nbody.cc:54 in main]	0,000s	1,820s	□	Scalar	inner loop ...			

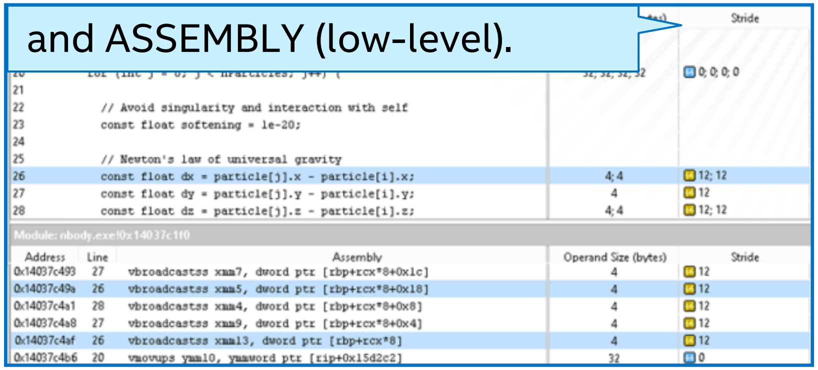
File: nbody.cc:57 main

SOURCE, ..

```
                                                                         %        Loop
54        for ( size_t i = 0; i < n; ++i ) {                                      3 640,
55            real dvx = 0, dvy = 0, dvz = 0;
56      //#pragma vector always
57      ⊟for ( size_t j = 0; j < n; ++j ) {                    10,110ms           3 640,
          [loop at nbody.cc:57 in main]
              Scalar loop. Not vectorized
              No loop transformations were applied
          [loop at nbody.cc:57 in main]
              Vectorized AVX loop processing Float32; Float64; Int32; UInt32 data t
              No loop transformations were applied
58                  if ( j != i ) {                           110,128ms
59                      real dx = x[j] - x[i], dy = y[j] - y[i], dz = z[j] - z[i];   289,778ms
60                      real dist2 = dx*dx + dy*dy + dz*dz;   100,042ms
61                      real mOverDist3 = m[j] / (dist2 * Sqrt( dist2 ));   710,194ms
62                      dvx += mOverDist3 * dx;               289,894ms
63                      dvy += mOverDist3 * dy;               259,742ms
64                      dvz += mOverDist3 * dz;                50,127ms
```



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



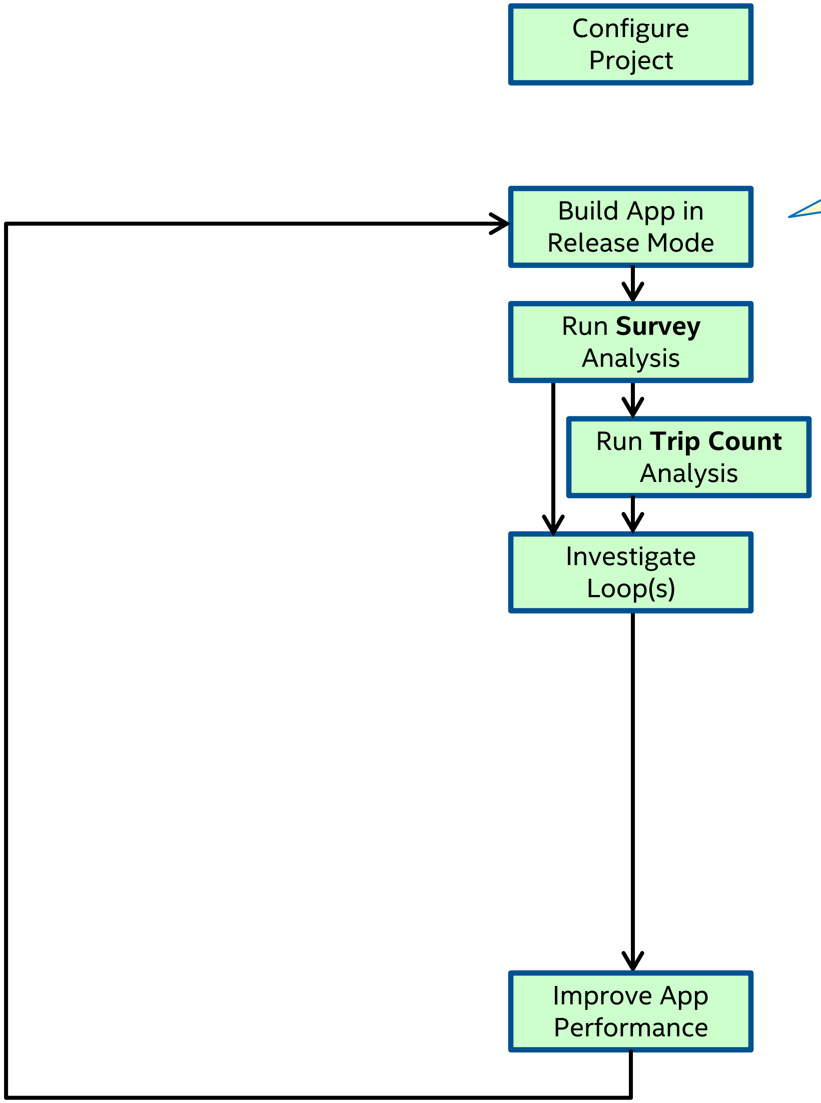
and ASSEMBLY (low-level).

```
20        for (int j = 0; j < nParticles; j++) {                   32; 32; 32; 32   0; 0; 0; 0
21
22          // Avoid singularity and interaction with self
23          const float softening = 1e-20;
24
25          // Newton's law of universal gravity
26          const float dx = particle[j].x - particle[i].x;       4; 4   12; 12
27          const float dy = particle[j].y - particle[i].y;       4      12
28          const float dz = particle[j].z - particle[i].z;       4; 4   12; 12
```

Module: nbody.exe!0x14037c1f0

Address	Line	Assembly	Operand Size (bytes)	Stride
0x14037c493	27	vbroadcastss xmm7, dword ptr [rbp+rcx*8+0x1c]	4	12
0x14037c49a	26	vbroadcastss xmm5, dword ptr [rbp+rcx*8+0x18]	4	12
0x14037c4a1	28	vbroadcastss xmm4, dword ptr [rbp+rcx*8+0x8]	4	12
0x14037c4a8	27	vbroadcastss xmm9, dword ptr [rbp+rcx*8+0x4]	4	12
0x14037c4af	26	vbroadcastss xmm13, dword ptr [rbp+rcx*8]	4	12
0x14037c4b6	20	vmovups ymm10, ymmword ptr [rip+0x15d2c2]	32	0

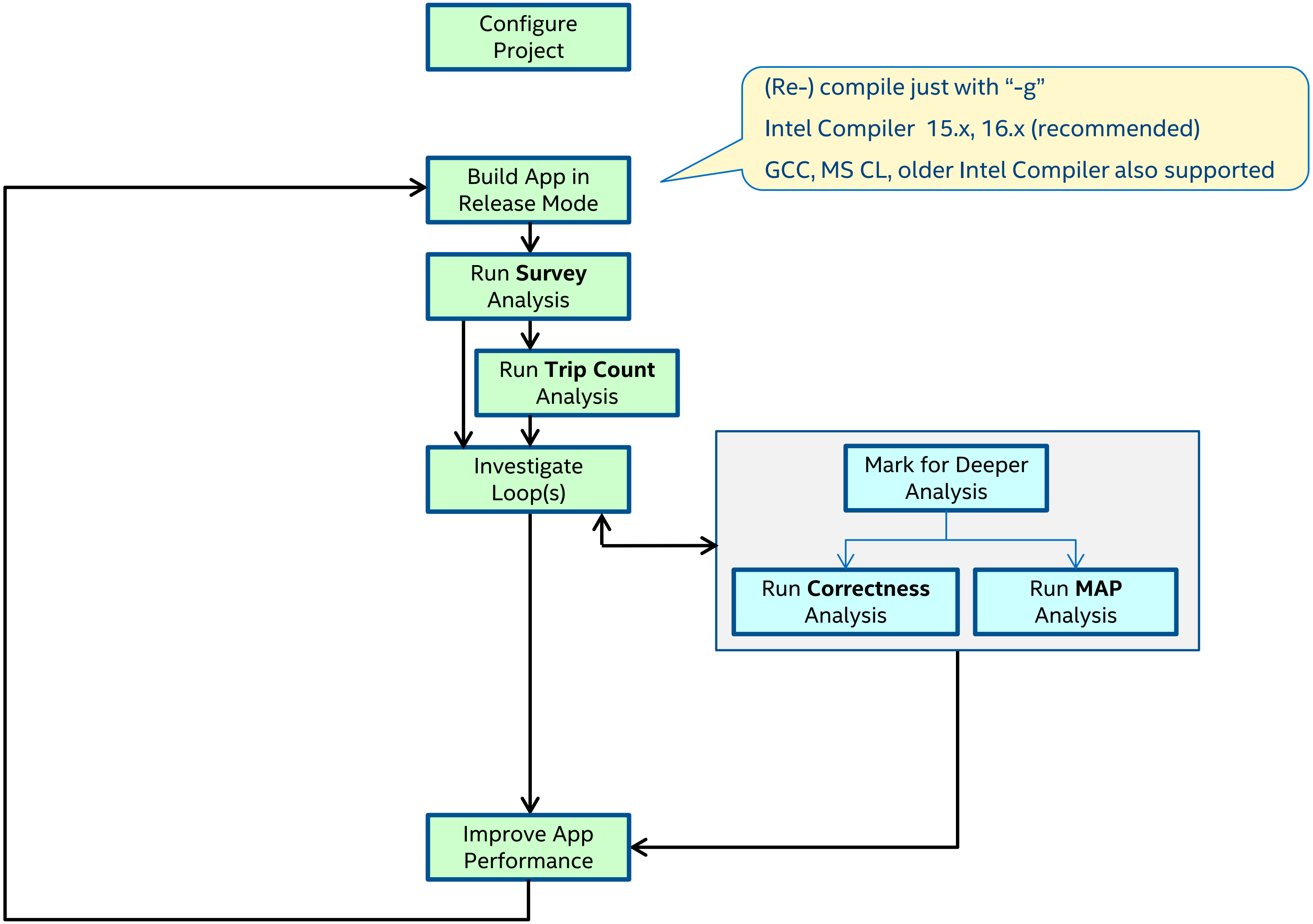
0.
Workflow



(Re-) compile just with “-g”

Intel Compiler 15.x, 16.x (recommended)

GCC, MS CL, older Intel Compiler also supported



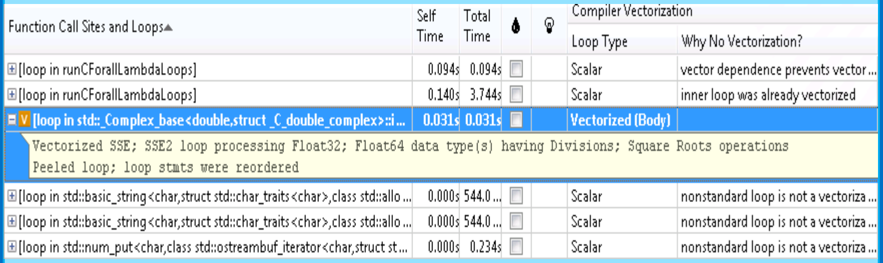
(Re-) compile just with “-g”

Intel Compiler 15.x, 16.x (recommended)

GCC, MS CL, older Intel Compiler also supported

1.
The Right Data At Your Fingertips

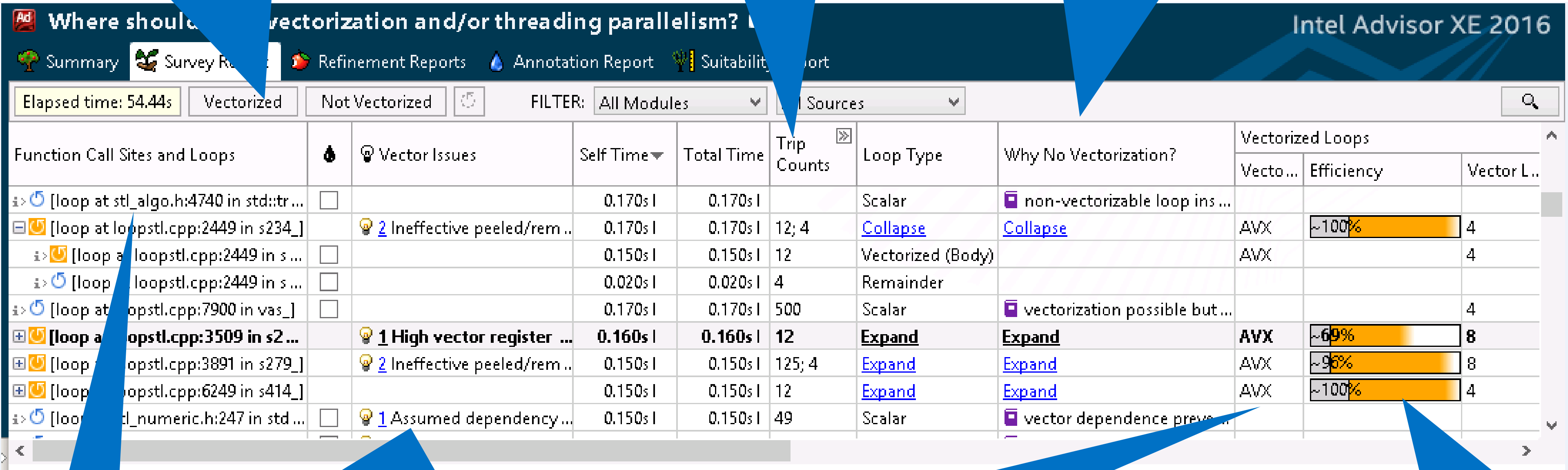
1. Compiler diagnostics + Performance Data + SIMD efficiency information



Function Call Sites and Loops▲	Self Time	Total Time	🔥	💡	Compiler Vectorization	
					Loop Type	Why No Vectorization?
⊞[loop in runCForallLambdaLoops]	0.094s	0.094s	□		Scalar	vector dependence prevents vector...
⊞[loop in runCForallLambdaLoops]	0.140s	3.744s	□		Scalar	inner loop was already vectorized
⊟ V [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 stmts 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...

The Right Data At Your Fingertips

Get all the data you need for high impact vectorization



Elapsed time: 54.44s | Vectorized | Not Vectorized | FILTER: All Modules | l Sources

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

Filter by which loops are vectorized!

Trip Counts

What prevents vectorization?

Focus on hot loops

What vectorization issues do I have?

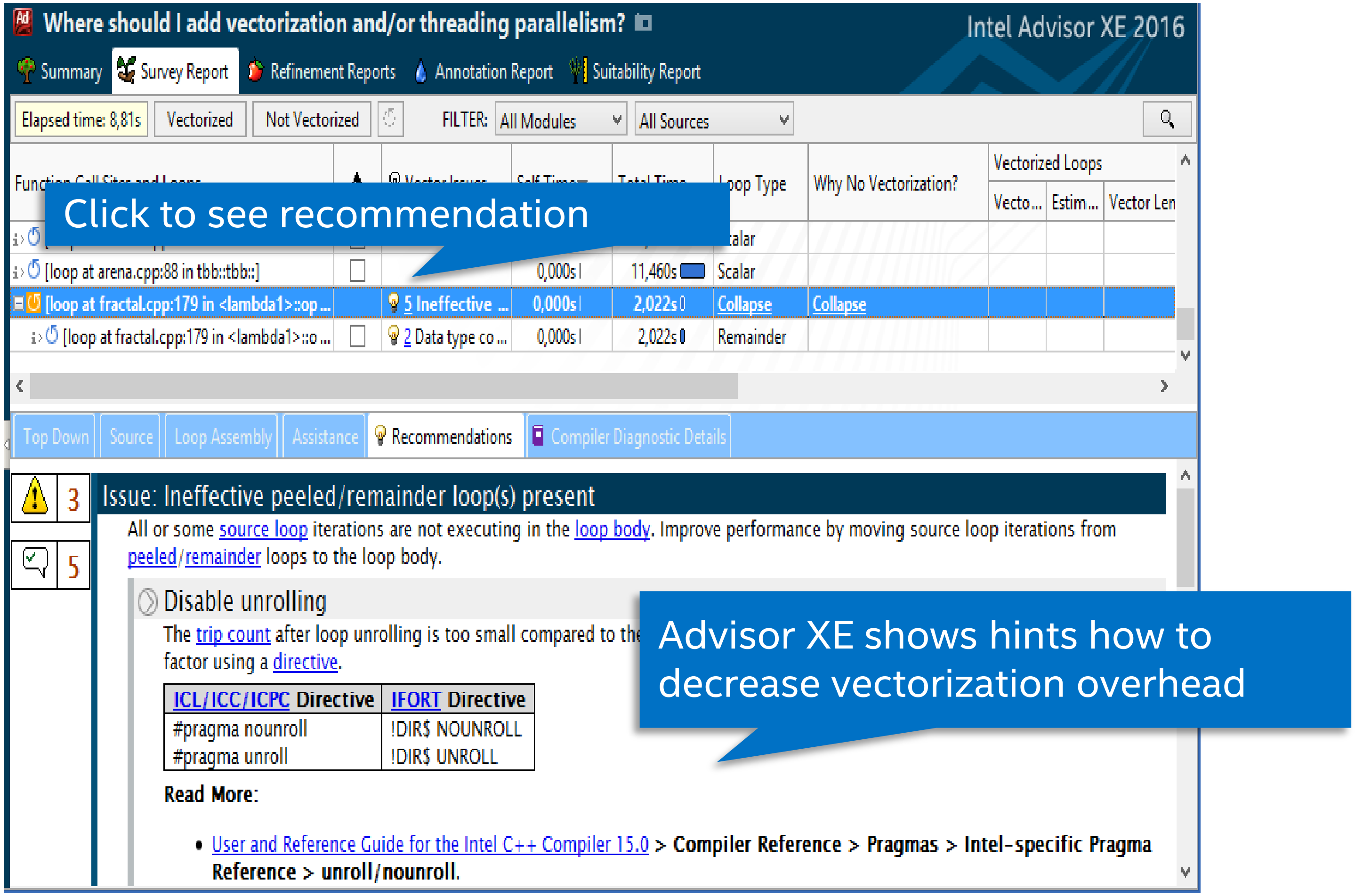
Which Vector instructions are being used?

How efficient is the code?

Get Faster Code Faster! Intel® Advisor XE
Vectorization Optimization and Thread Prototyping

Get Specific Advice For Improving Vectorization

Intel® Advisor XE – Vectorization Advisor



Where should I add vectorization and/or threading parallelism?

Intel Advisor XE 2016

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops		
							Vecto...	Estim...	Vector Len
	☐				alar				
[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

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 the ... factor using a directive.

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

Read More:

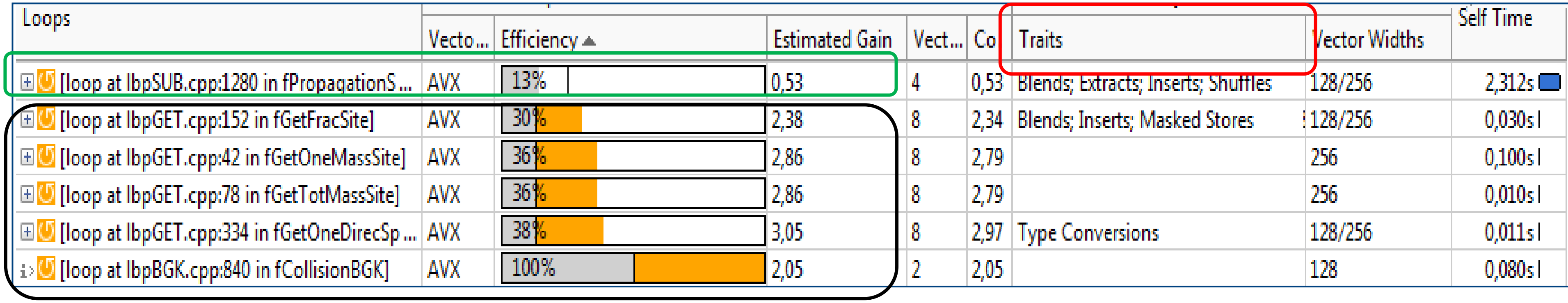
- User and Reference Guide for the Intel C++ Compiler 15.0 > **Compiler Reference > Pragmas > Intel-specific Pragma Reference > unroll/nounroll.**

Advisor XE shows hints how to decrease vectorization overhead

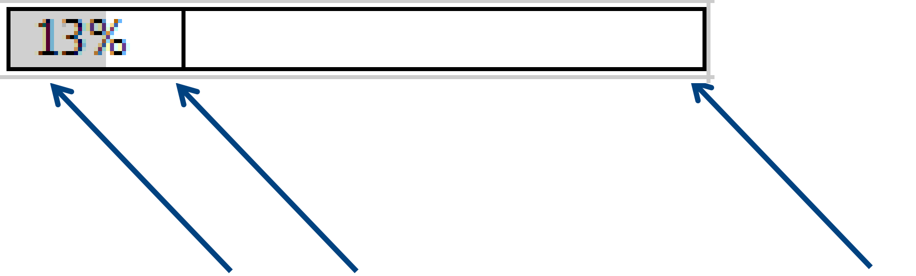
Vector Efficiency: my performance thermometer

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



Achieved Efficiency

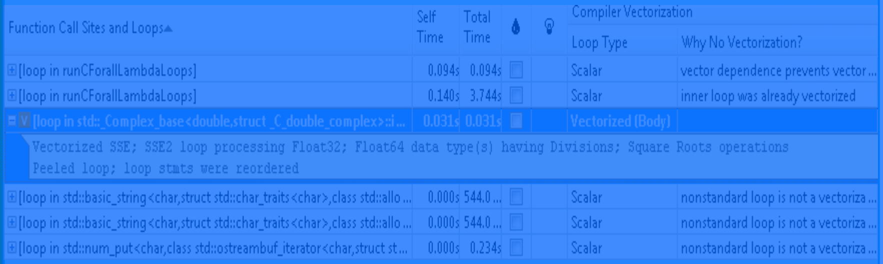
Original (scalar) code efficiency. Corresponds to 1x speed-up.

Upper bound: 100% efficiency 4x gain (VL=4)

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

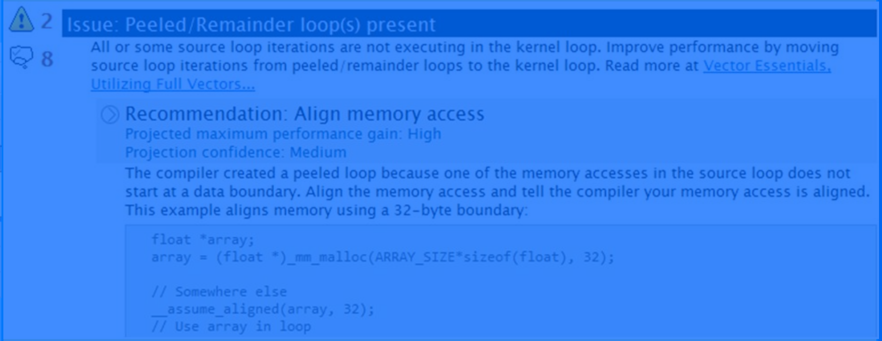
2.
Is it safe to vectorize:
Tough problem #1 for not yet vectorized codes.

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
⊟ V [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 stmts 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

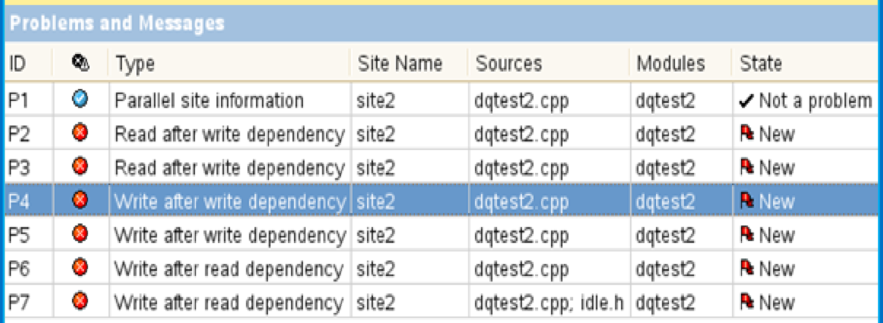
Projected maximum performance gain: High
Projection confidence: Medium

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
```

4. Loop-Carried Dependency Analysis

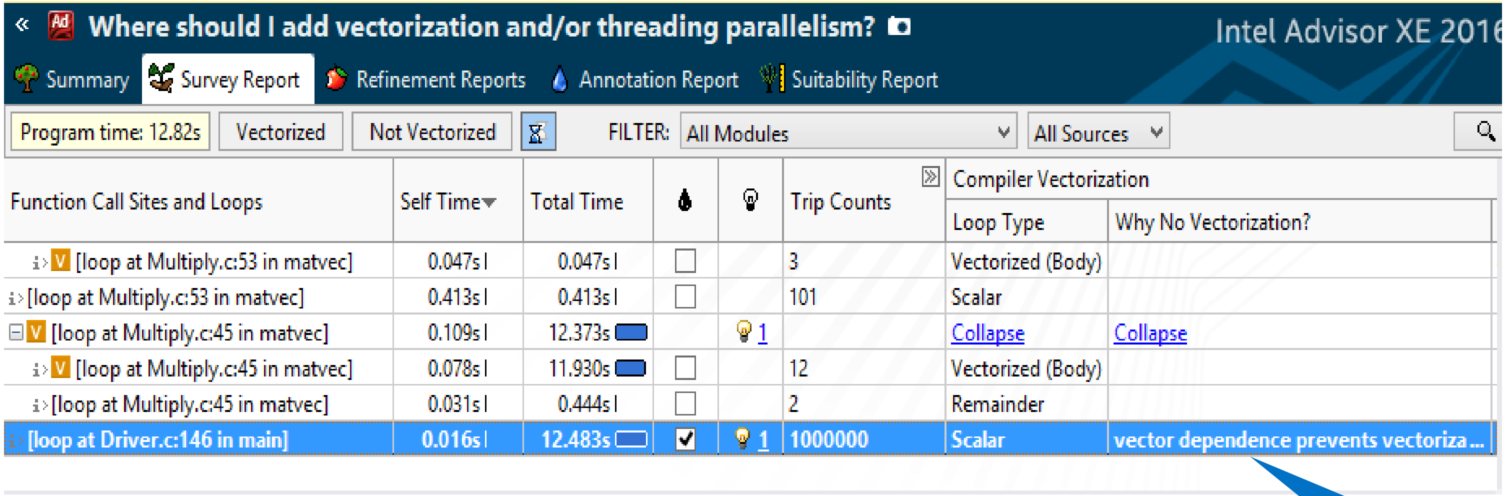


Problems and Messages

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

Is It Safe to Vectorize?

Loop-carried dependencies analysis verifies correctness



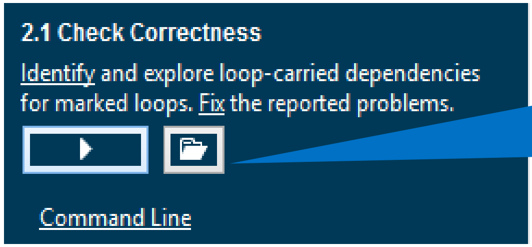
« Where should I add vectorization and/or threading parallelism?

Intel Advisor XE 2016

Program time: 12.82s

FILTER: All Modules | All Sources

Function Call Sites and Loops	Self Time	Total Time			Trip Counts	Compiler Vectorization	
						Loop Type	Why No Vectorization?
V [loop at Multiply.c:53 in matvec]	0.047s	0.047s	□		3	Vectorized (Body)	
[loop at Multiply.c:53 in matvec]	0.413s	0.413s	□		101	Scalar	
⊟ V [loop at Multiply.c:45 in matvec]	0.109s	12.373s		1		Collapse	Collapse
V [loop at Multiply.c:45 in matvec]	0.078s	11.930s	□		12	Vectorized (Body)	
[loop at Multiply.c:45 in matvec]	0.031s	0.444s	□		2	Remainder	
[loop at Driver.c:146 in main]	0.016s	12.483s	☑	1	1000000	Scalar	vector dependence prevents vectoriza...



2.1 Check Correctness

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

Command Line

Select loop for Correct Analysis and press play!

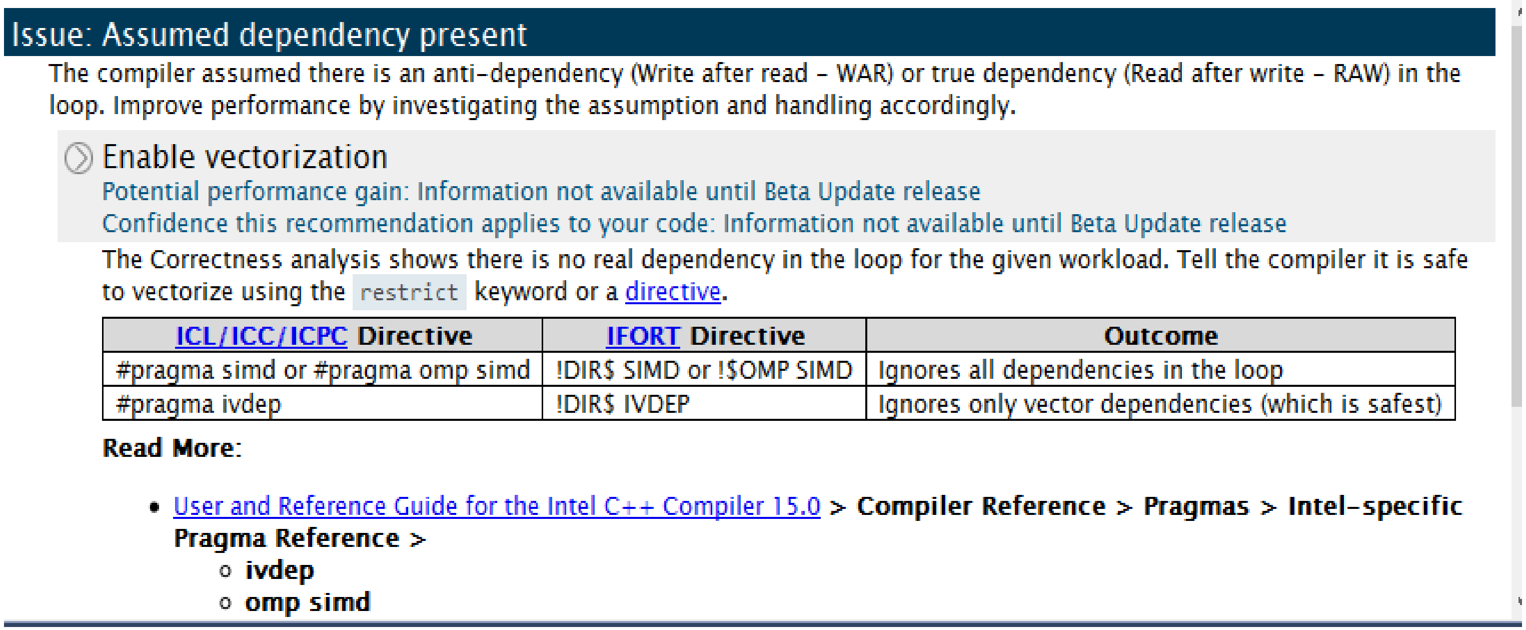
Vector Dependence prevents Vectorization!

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
#pragma simd or #pragma omp simd	!DIR$ SIMD or !$OMP SIMD	Ignores all dependencies in the loop
#pragma ivdep	!DIR$ IVDEP	Ignores only vector dependencies (which is safest)

Read More:

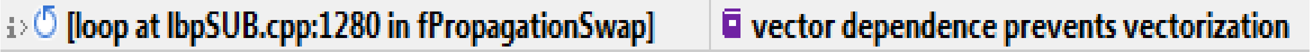
- User and Reference Guide for the Intel C++ Compiler 15.0 > **Compiler Reference > Pragmas > Intel-specific Pragma Reference >**
  - **ivdep**
  - **omp simd**

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]);
```

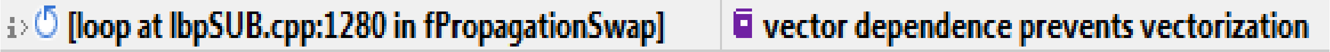
Static Assumption:



[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]);
```

Static Assumption:



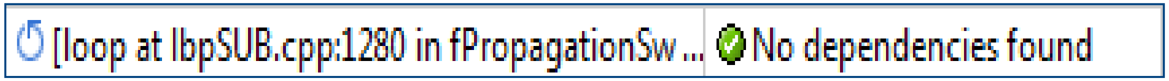
[loop at lbpSUB.cpp:1280 in fPropagationSwap] | vector dependence prevents vectorization

Both loops “equally bad” :
from static analysis perspective

Data Dependencies – Tough Problem #1

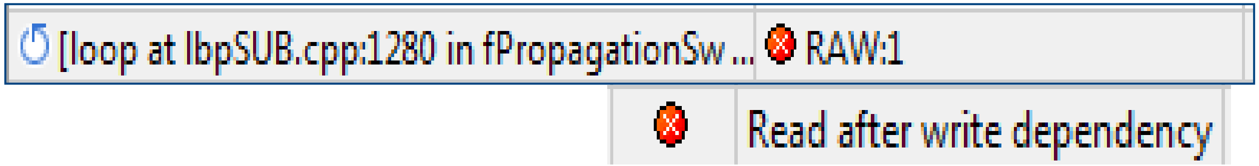
Dynamic check ***knows*** if memory accesses really overlap.

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]);
```



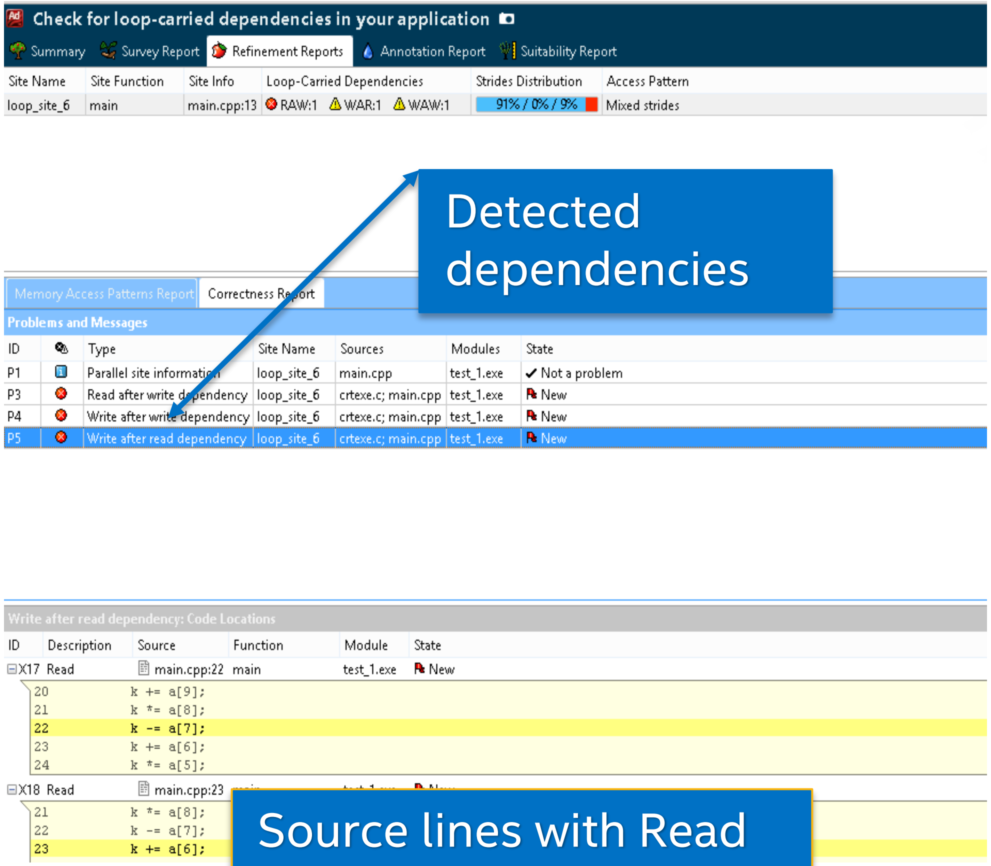
[loop at lbpSUB.cpp:1280 in fPropagationSw ... ☒ RAW:1

☒ Read after write dependency

Correctness Analysis: confirm dependencies are REAL

Correctness – Is It Safe to Vectorize?

Loop-carried dependencies analysis



Check for loop-carried dependencies in your application

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_6	main	main.cpp:13	RAW:1 ⚠ WAR:1 ⚠ WAW:1	91% / 0% / 9%	Mixed strides

Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P1		Parallel site information	loop_site_6	main.cpp	test_1.exe	✓ Not a problem
P3	●	Read after write dependency	loop_site_6	crtexe.c; main.cpp	test_1.exe	New
P4	●	Write after write dependency	loop_site_6	crtexe.c; main.cpp	test_1.exe	New
P5	●	Write after read dependency	loop_site_6	crtexe.c; main.cpp	test_1.exe	New

Write after read dependency: Code Locations

ID	Description	Source	Function	Module	State
⊟X17	Read	main.cpp:22	main	test_1.exe	New

```
20          k += a[9];
21          k *= a[8];
22          k -= a[7];
23          k += a[6];
24          k *= a[5];
```

ID	Description	Source	Function	Module	State
⊟X18	Read	main.cpp:23	main	test_1.exe	New

```
21          k *= a[8];
22          k -= a[7];
23          k += a[6];
```

Detected dependencies

Source lines with Read and Write accesses detected

1. Mark-up the loop and check for the presence of REAL dependencies

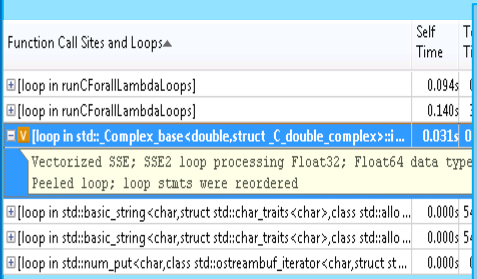
2. Explore dependencies in more details with code snippets

In this example 3 dependencies were detected

- RAW – Read After Write
- WAR – Write After Read
- WAW – Write After Write

3.
Any speed-up out of there? Use SIMD to make your code faster, instead of slower.

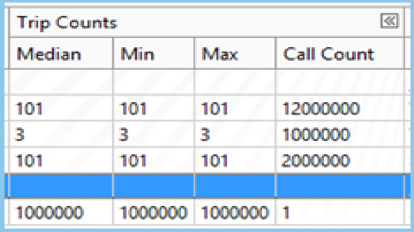
1. Compiler diagnostics + Performance Data + SIMD efficiency information



Function Call Sites and Loops▲	Self Time	T Ti
⊞[loop in runCForallLambdaLoops]	0.094s	0
⊞[loop in runCForallLambdaLoops]	0.140s	3
⊟ V [loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0
⊞[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	54
⊞[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	54
⊞[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	0

```
Vectorized SSE; SSE2 loop processing Float32; Float64 data type
Peeled loop; loop stmts were reordered
```

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



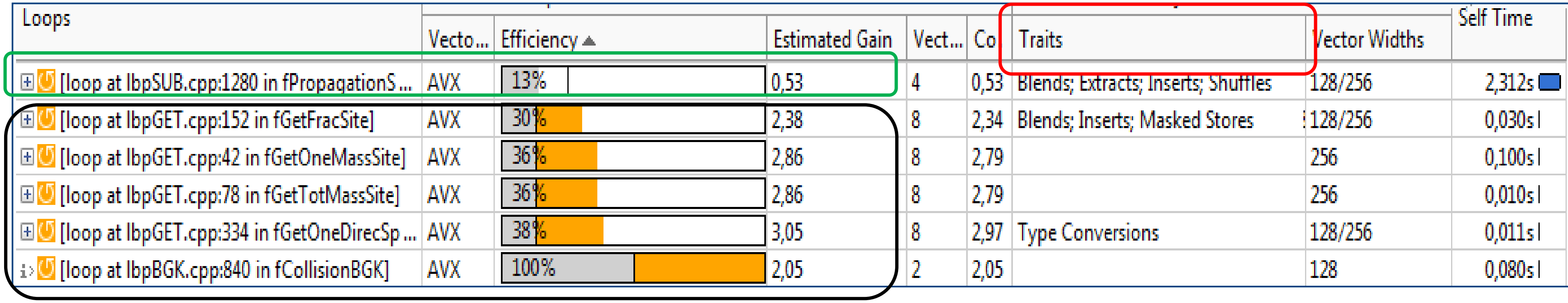
Trip Counts

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

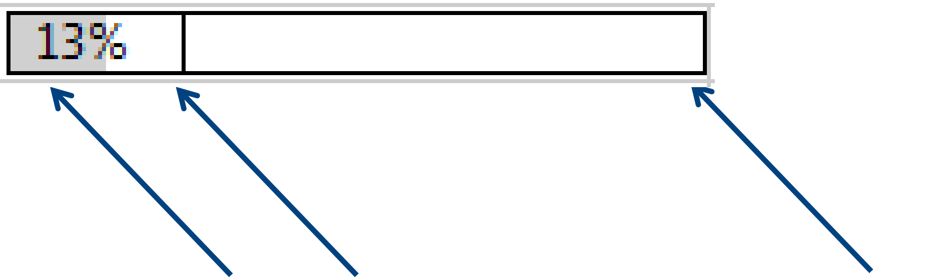
Vector Efficiency: my performance thermometer

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



Achieved Efficiency

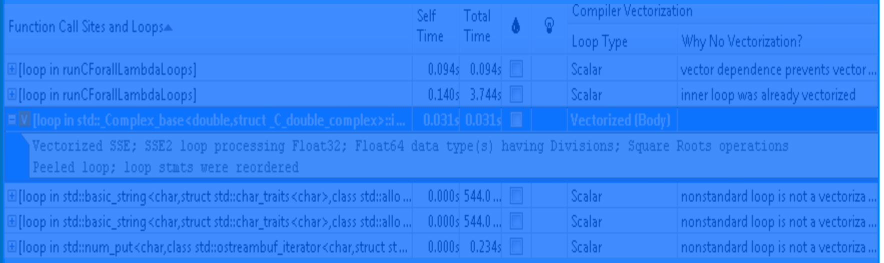
Original (scalar) code efficiency. Corresponds to 1x speed-up.

Upper bound: 100% efficiency 4x gain (VL=4)

- **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

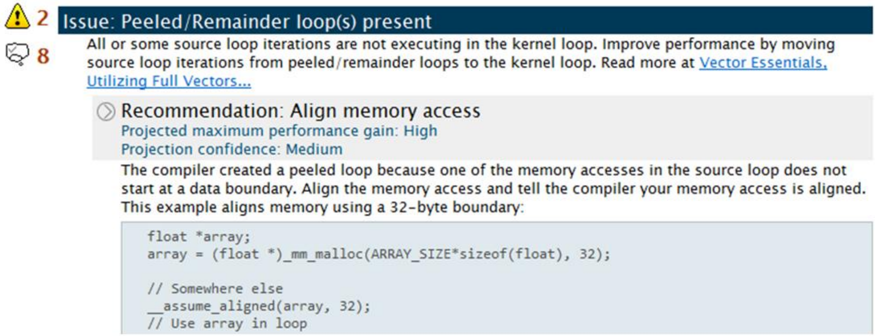
Survey: find out if your code is “undervectorized” and why

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
⊟ V [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 stmts 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



⚠ 2 **Issue: Peeled/Remainder loop(s) present**

8 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!

This is where we want our loops to be executing!

Optional peel part

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

Remainder part

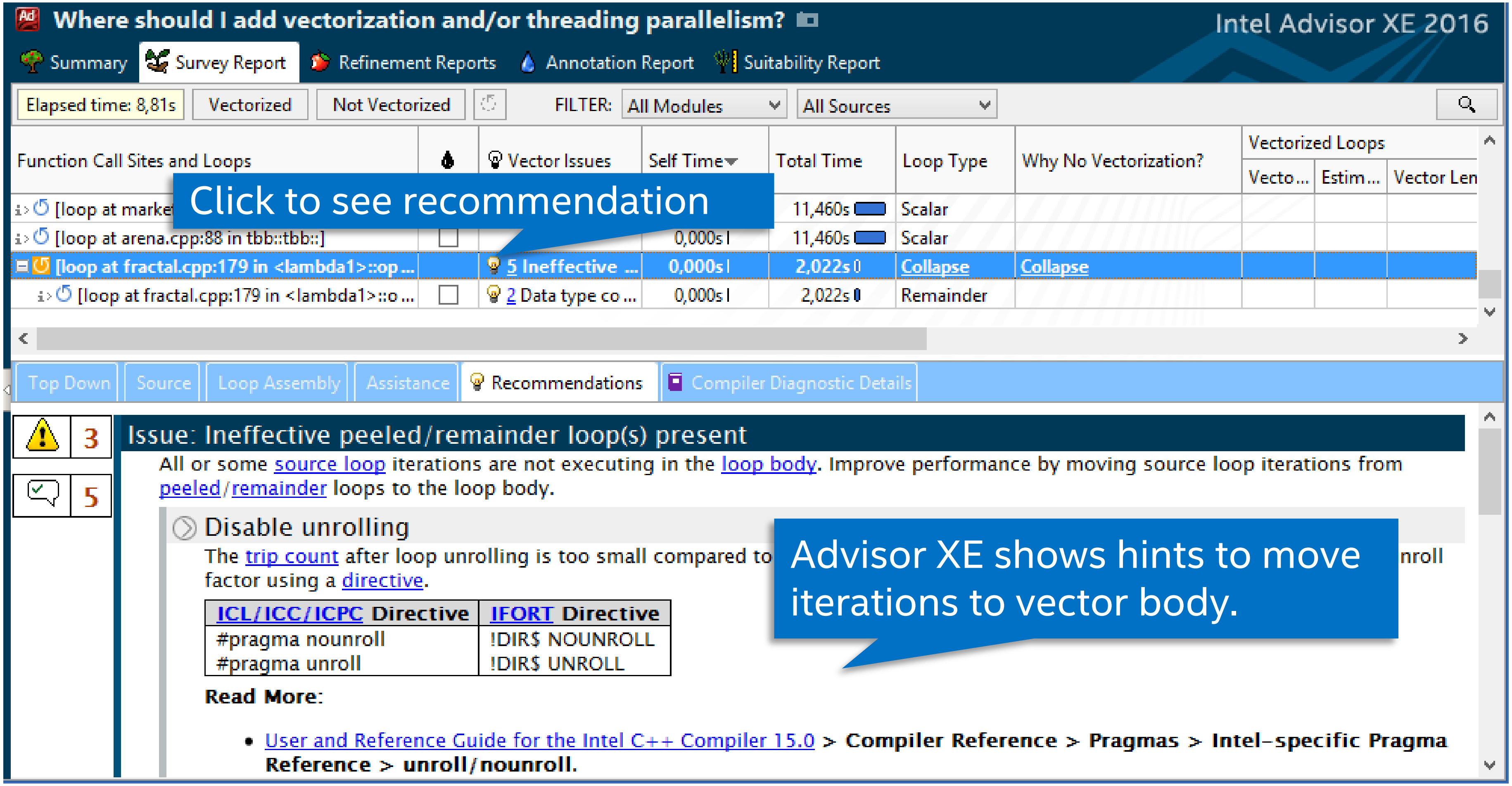
- Due to the number of iterations (trip count) not being divisible by vector length. Uses Scalar or slower vector.

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!

Get Specific Advice For Improving Vectorization

Intel® Advisor XE – Vectorization Advisor



Where should I add vectorization and/or threading parallelism?

Intel Advisor XE 2016

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops Vecto...	 Estim...	 Vector Len
[loop at marke				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				

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 nroll factor using a directive.

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

Read More:

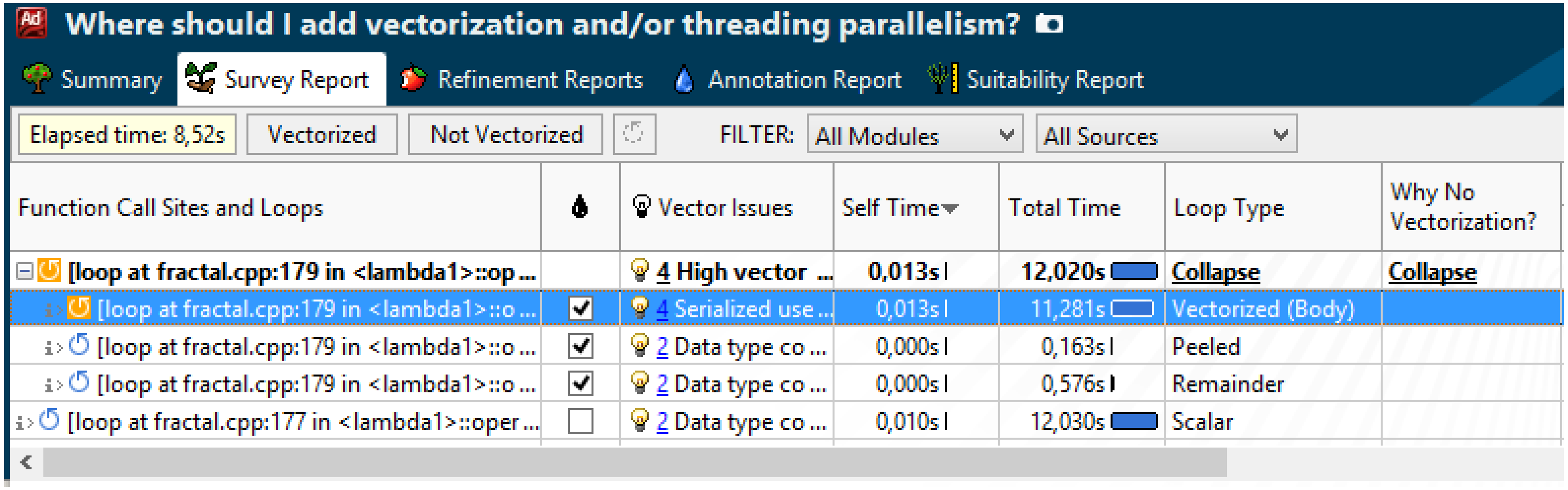
- User and Reference Guide for the Intel C++ Compiler 15.0 > **Compiler Reference > Pragmas > Intel-specific Pragma Reference > unroll/nounroll.**

Click to see recommendation

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?

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	

4.
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
```

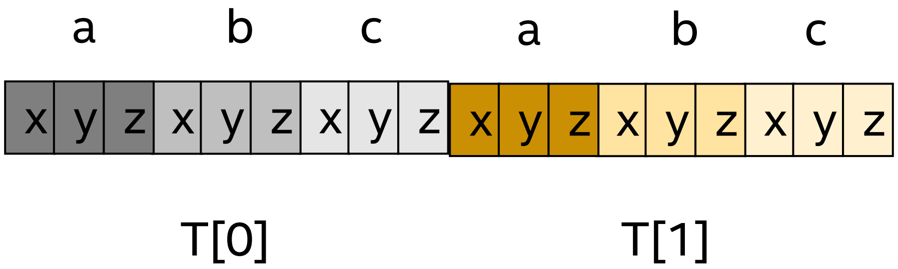
- Non-unit-stride (constant stride) access to arrays

```
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



a			b			c			a			b			c		
x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z

T[0] T[1]

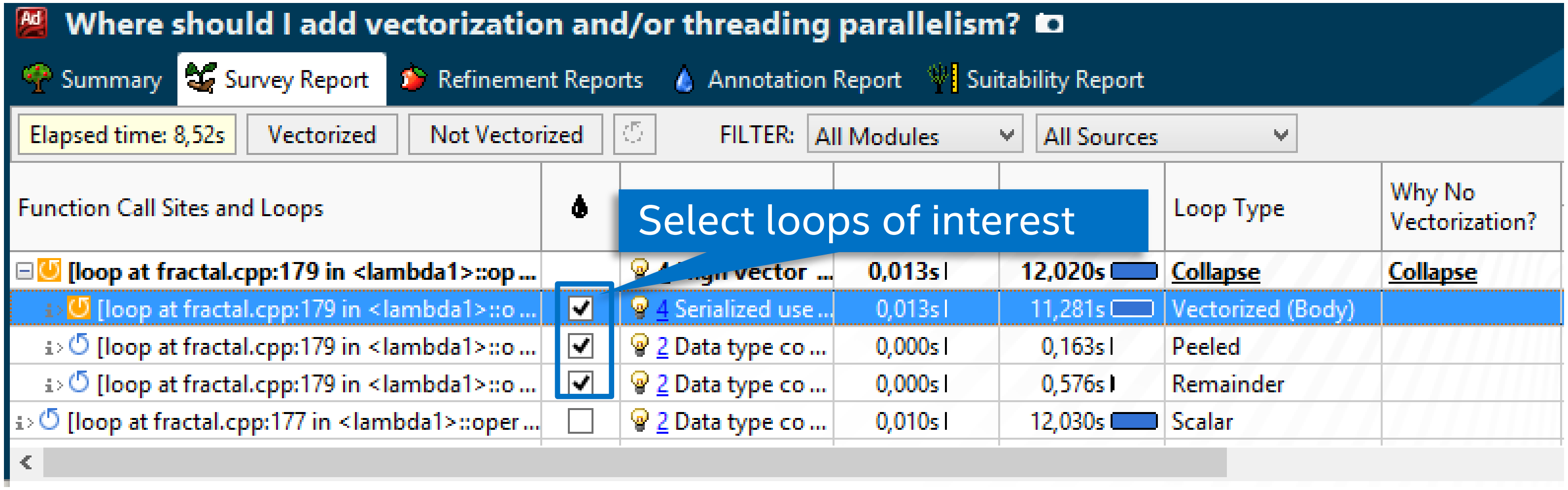
```
Class Point {float
x,y,z;}
Class Triangle {Point
a,b,c;}
Triangle T[100];
Point Cross( const Point& a, const Point& b ) {
    return Point( a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z,
a.x*a.y-a.y-b.x );
}

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

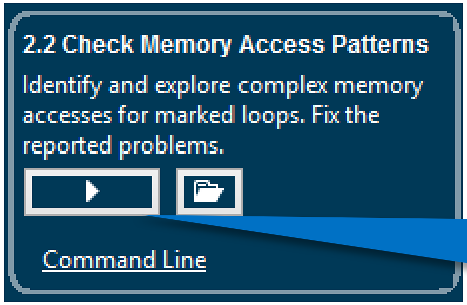
Memory Access pattern analysis



Where should I add vectorization and/or threading parallelism?

Function Call Sites and Loops					Loop Type	Why No Vectorization?
⊟ [loop at fractal.cpp:179 in <lambda1>::op ...		4 [illegible]gn 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	

Select loops of interest

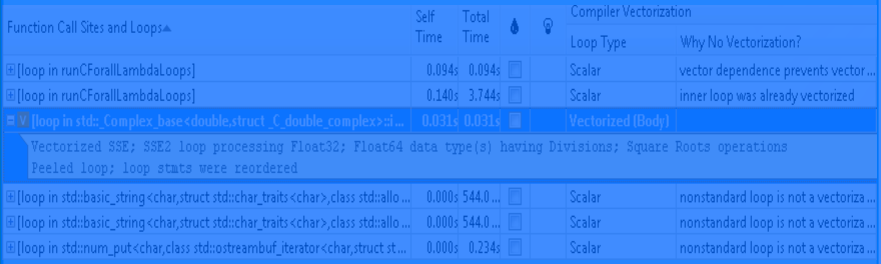


2.2 Check Memory Access Patterns

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

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

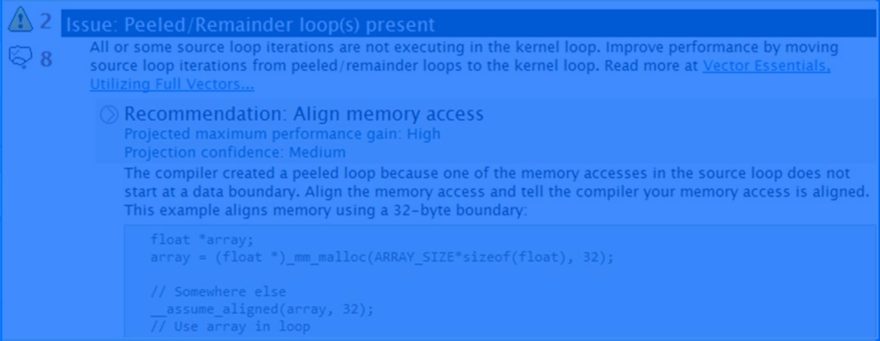


Function Call Sites and Loops▲	Self Time	Total Time			Compiler Vectorization Loop Type	 Why No Vectorization?
⊞[loop in runCForallLambdaLoops]	0.094s	0.094s	□		Scalar	vector dependence prevents vector…
⊞[loop in runCForallLambdaLoops]	0.140s	3.744s	□		Scalar	inner loop was already vectorized
⊟ V [loop 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 stmts 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



2 **Issue: Peeled/Remainder loop(s) present**

8 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

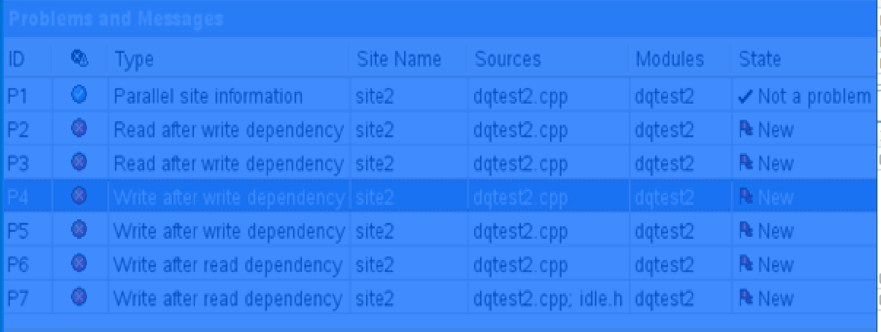
Projection confidence: Medium

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
```

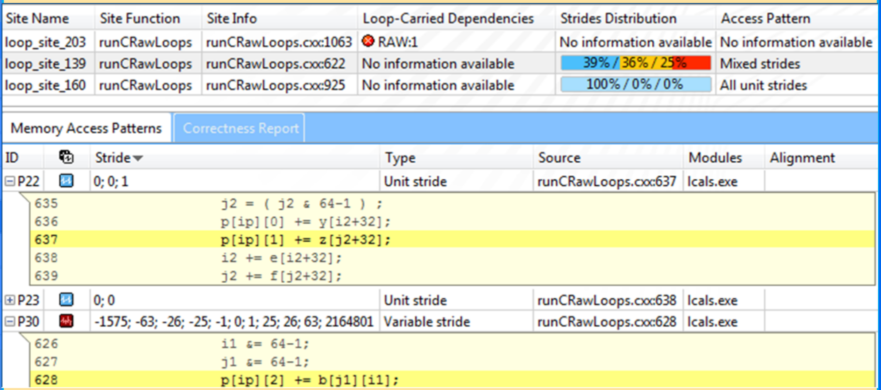
3. Loop-Carried Dependency Analysis



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

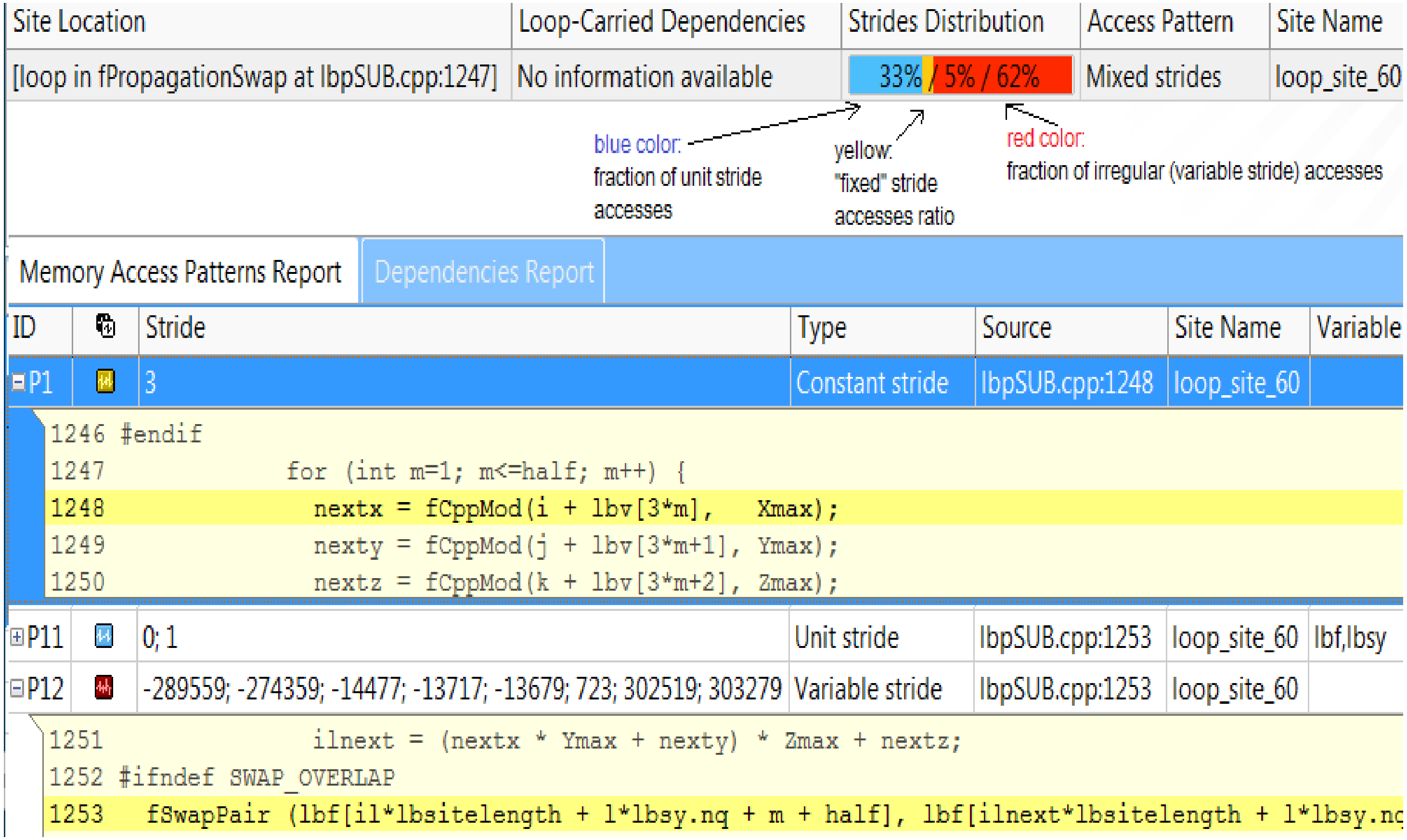
ID		Stride▾	Type	Source	Modules	Alignment
⊟P22		0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	

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

ID		Stride▾	Type	Source	Modules	Alignment
⊞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	

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

Know your access pattern



Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
[loop in fPropagationSwap at lbpSUB.cpp:1247]	No information available	33% / 5% / 62%	Mixed strides	loop_site_60

blue color: fraction of unit stride accesses

yellow: "fixed" stride accesses ratio

red color: fraction of irregular (variable stride) accesses

ID		Stride	Type	Source	Site Name	Variable
⊟P1	▨	3	Constant stride	lbpSUB.cpp:1248	loop_site_60	

```
1246 #endif
1247            for (int m=1; m<=half; m++) {
1248              nextx = fCppMod(i + lbv[3*m],   Xmax);
1249              nexty = fCppMod(j + lbv[3*m+1], Ymax);
1250              nextz = fCppMod(k + lbv[3*m+2], Zmax);
```

ID		Stride	Type	Source	Site Name	Variable
⊞P11	▨	0; 1	Unit stride	lbpSUB.cpp:1253	loop_site_60	lbf,lbsy
⊟P12	▨	-289559; -274359; -14477; -13717; -13679; 723; 302519; 303279	Variable stride	lbpSUB.cpp:1253	loop_site_60	

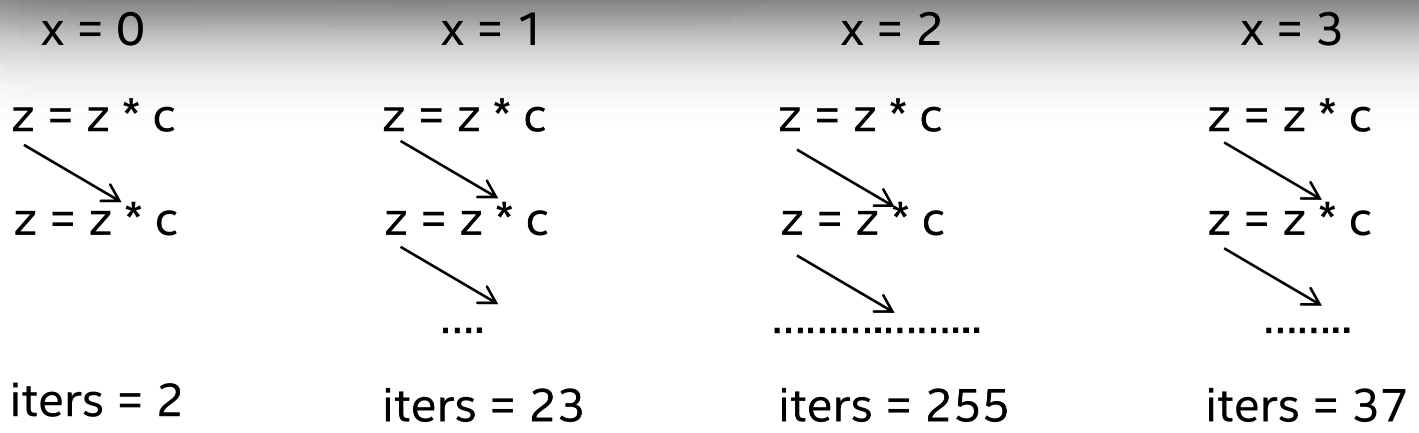
```
1251              ilnext = (nextx * Ymax + nexty) * Zmax + nextz;
1252 #ifndef SWAP_OVERLAP
1253   fSwapPair (lbf[il*lbsitelength + l*lbsy.nq + m + half], lbf[ilnext*lbsitelength + l*lbsy.nq
```

5.
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

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

```
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 (...);
          }
        }
      }
   }
}
```

Here?

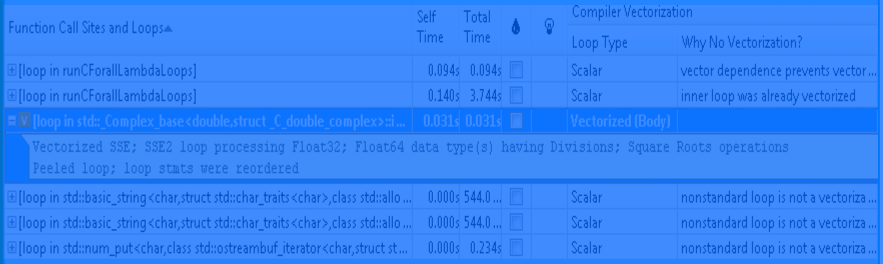
Here????

Here???

Here??

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

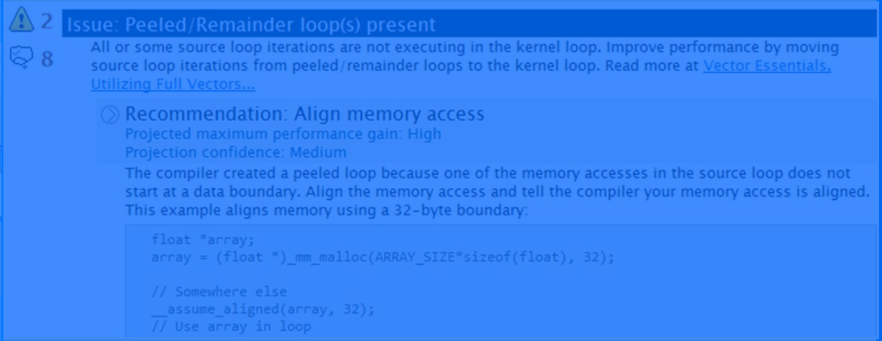


Function Call Sites and Loops	Self Time	Total Time			Compiler Vectorization Loop Type	 Why No Vectorization?
⊞[loop in runCForallLambdaLoops]	0.094s	0.094s	□		Scalar	vector dependence prevents vector…
⊞[loop in runCForallLambdaLoops]	0.140s	3.744s	□		Scalar	inner loop was already vectorized
⊟ V [loop 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 stmts 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

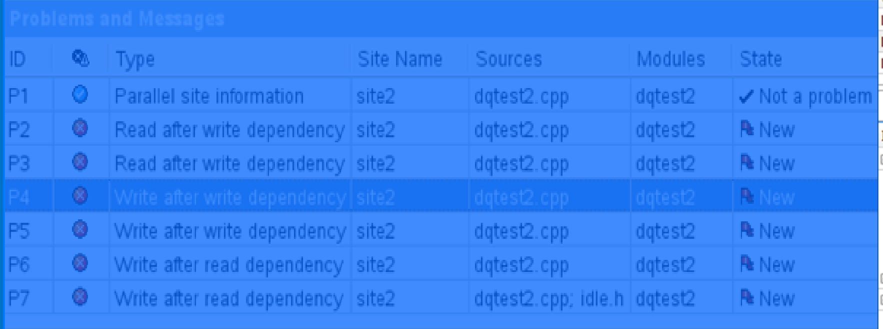
Projection confidence: Medium

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
```

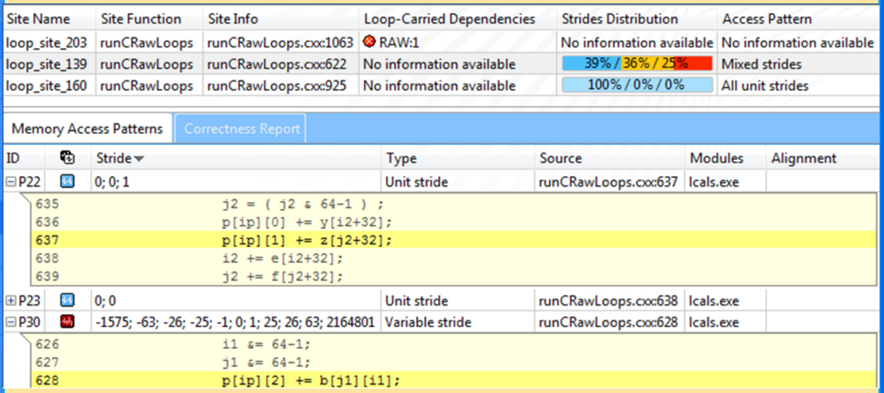
3. Loop-Carried Dependency Analysis



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	

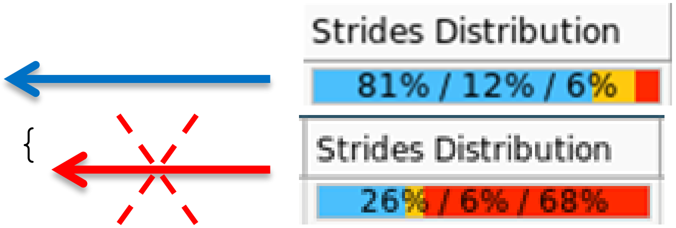
```
635                  j2 = ( j2 & 64-1 ) ;
636                  p[ip][0] += y[i2+32];
637                  p[ip][1] += z[j2+32];
638                  i2 += e[i2+32];
639                  j2 += f[j2+32];
```

⊞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	

```
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%

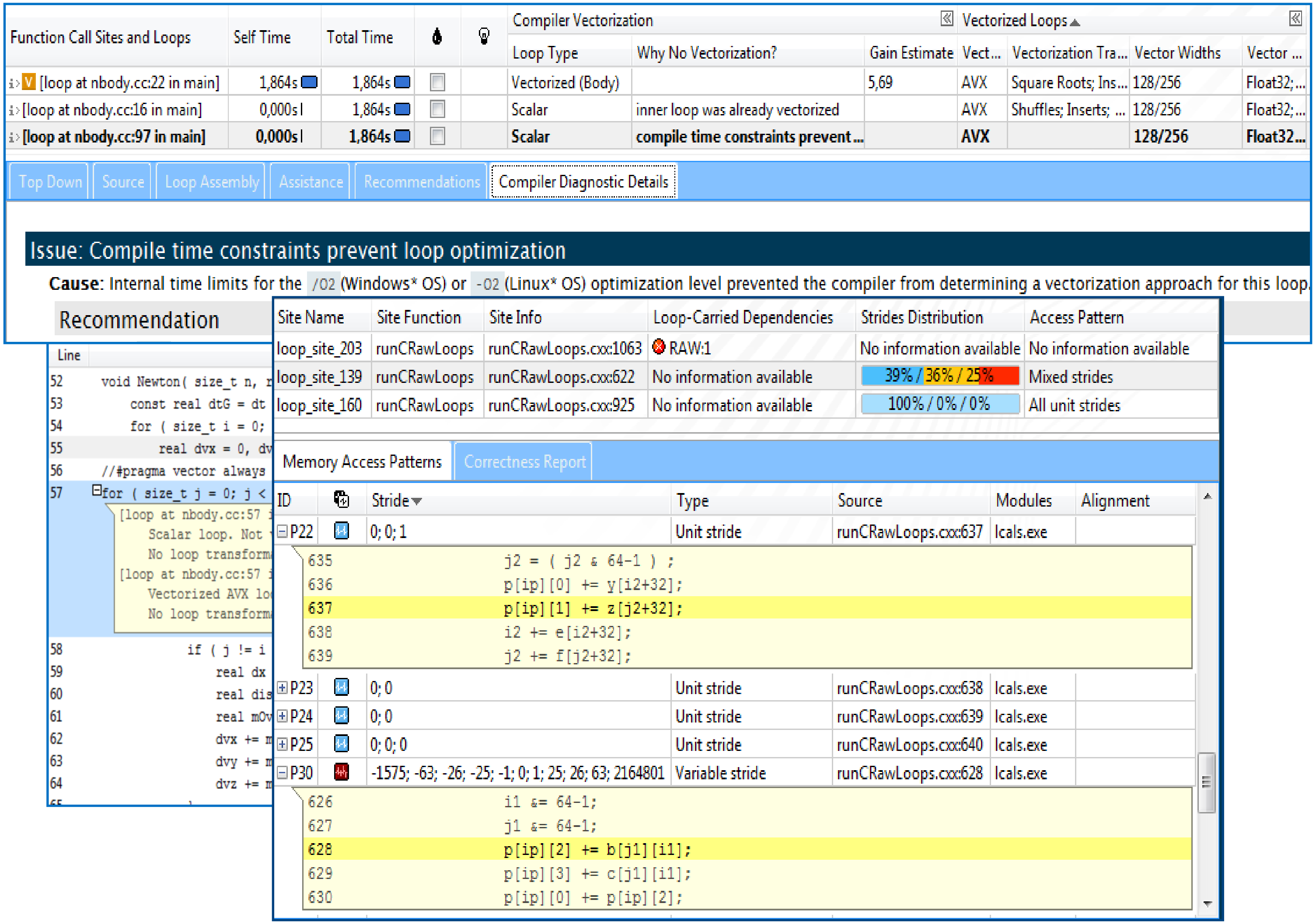
Strides Distribution

26% / 6% / 68%

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

Vector Advisor

- All the data in one place
  (also leveraging Intel Compiler 15.x/16.x reports)
- Guidance and Correctness check
- Deep dive memory analysis



Function Call Sites and Loops	Self Time	Total Time			Compiler Vectorization		Vectorized Loops				
					Loop Type	Why No Vectorization?	Gain Estimate	Vect...	Vectorization Tra...	Vector Widths	Vector ...
V [loop at nbody.cc:22 in main]	1,864s	1,864s	□		Vectorized (Body)		5,69	AVX	Square Roots; Ins...	128/256	Float32;...
[loop at nbody.cc:16 in main]	0,000s	1,864s	□		Scalar	inner loop was already vectorized		AVX	Shuffles; Inserts; ...	128/256	Float32;...
[loop at nbody.cc:97 in main]	**0,000s**	**1,864s**	□		**Scalar**	**compile time constraints prevent...**		**AVX**		**128/256**	**Float32...**

Issue: Compile time constraints prevent loop optimization

Cause: Internal time limits for the /O2 (Windows* OS) or -O2 (Linux* OS) optimization level prevented the compiler from determining a vectorization approach for this loop.

Recommendation

```
    void Newton( size_t n, r
        const real dtG = dt
        for ( size_t i = 0;
            real dvx = 0, dv
    //#pragma vector always
    for ( size_t j = 0; j <
      [loop at nbody.cc:57 i
          Scalar loop. Not
          No loop transform
      [loop at nbody.cc:57 i
          Vectorized AVX lo
          No loop transform
              if ( j != i
                  real dx
                  real dis
                  real mOv
                  dvx += m
                  dvy += m
                  dvz += m
```

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	

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

ID		Stride	Type	Source	Modules	Alignment
⊞P23		0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
⊞P24		0; 0	Unit stride	runCRawLoops.cxx:639	lcals.exe	
⊞P25		0; 0; 0	Unit stride	runCRawLoops.cxx:640	lcals.exe	
⊟P30		-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628	lcals.exe	

```
                    i1 &= 64-1;
                    j1 &= 64-1;
                    p[ip][2] += b[j1][i1];
                    p[ip][3] += c[j1][i1];
                    p[ip][0] += p[ip][2];
```

Threading Advisor XE

Data-Driven Threading Design

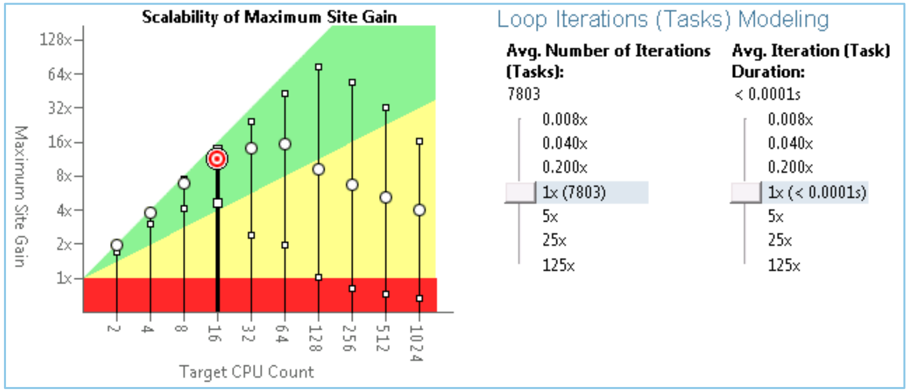
Intel® Advisor XE – Thread Prototyping

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



Loop Iterations (Tasks) Modeling

Avg. Number of Iterations (Tasks): 7803

0.008x
0.040x
0.200x
1x (7803)
5x
25x
125x

Avg. Iteration (Task) Duration: < 0.0001s

0.008x
0.040x
0.200x
1x (< 0.0001s)
5x
25x
125x

Add Parallelism with Less Effort, Less Risk and More Impact

http://intel.ly/advisor-xe

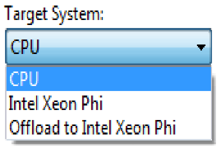
Check Suitability

Is it fast enough?

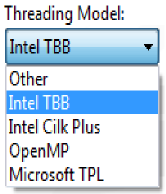
Experiment with modeling by changing:

- Number of tasks
- Task duration
- Runtime modeling
- Threading model
- Target system

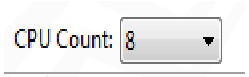
Instantly see impact on scalability



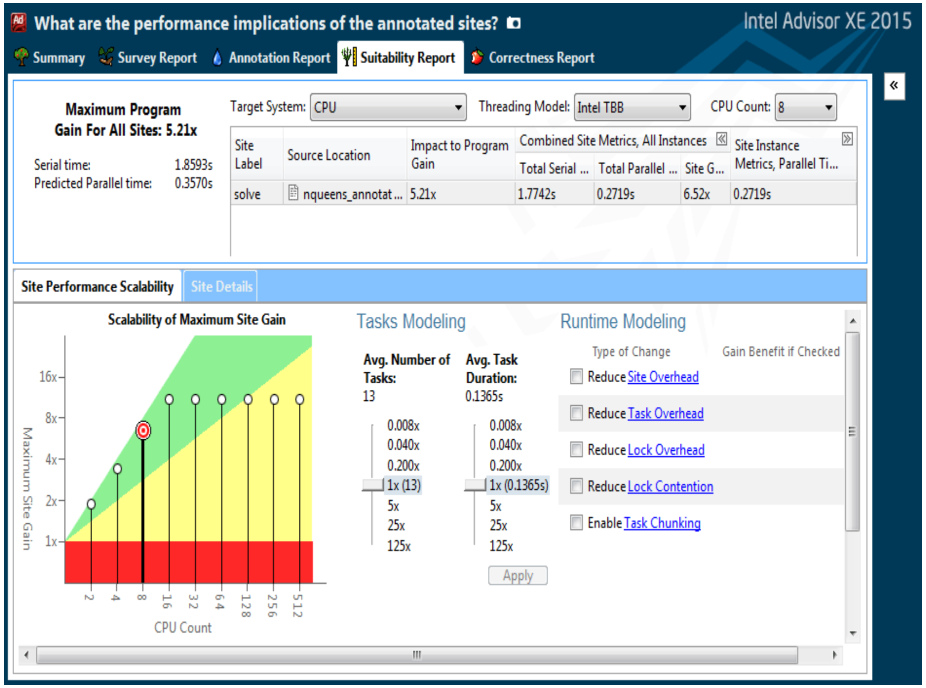
Target System: CPU



Threading Model: Intel TBB



CPU Count: 8



What are the performance implications of the annotated sites?

Maximum Program Gain For All Sites: 5.21x

Serial time: 1.8593s
Predicted Parallel time: 0.3570s

Target System: CPU
Threading Model: Intel TBB
CPU Count: 8

Site Label	Source Location	Impact to Program Gain	Combined Site Metrics, All Instances			Site Instance Metrics, Parallel Ti...
			Total Serial ...	Total Parallel ...	Site G...	
solve	nqueens_annotat ...	5.21x	1.7742s	0.2719s	6.52x	0.2719s

Tasks Modeling

Avg. Number of Tasks: 13

Avg. Task Duration: 0.1365s

Runtime Modeling

Type of Change	Gain Benefit if Checked
□ Reduce Site Overhead	
□ Reduce Task Overhead	
□ Reduce Lock Overhead	
□ Reduce Lock Contention	
□ Enable Task Chunking	

Quickly Evaluate Design Alternatives

Summary

Some Future Plans

KNL ("native") support, including:

- Survey analysis with **AVX512 ISA-specific insights and advices**. Identify cases where migration to AVX512 may give special benefit
- Memory Access Pattern extended to provide **vgather/vscatter and masking utilization** analysis

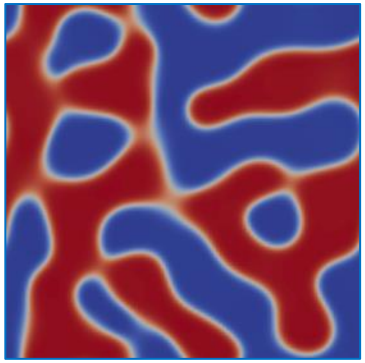
Memory wall vs. Vectorization:

- "Quick and Dirty" **memory-bound (DRAM/LLC/L2/L1) vs. compute-bound** checks and modeling
- **Footprint/latency** - deeper dive in Advisor **MAP**

Back-up

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 Boltzman 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

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

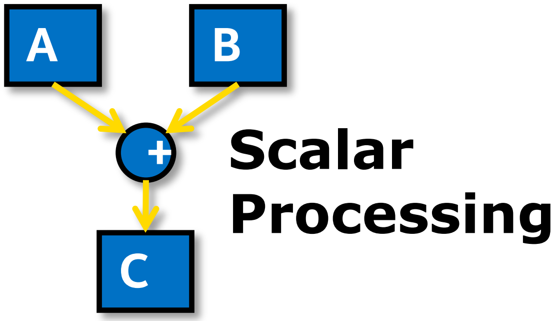
Beta Registration & Download:

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

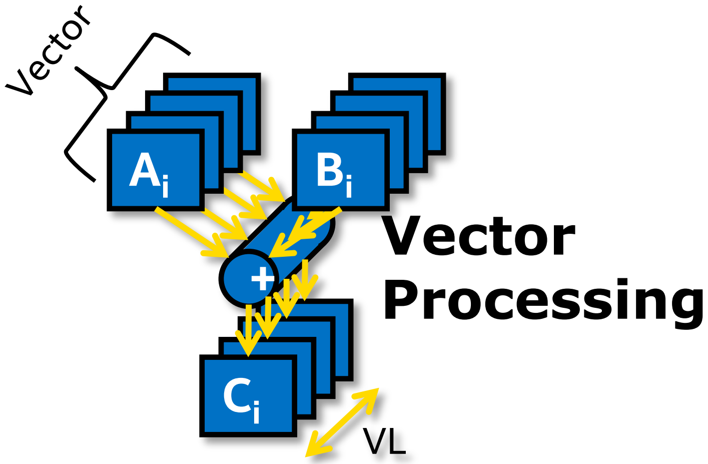
For Intel® Xeon Phi™ coprocessors, but also applicable:
https://software.intel.com/en-us/articles/vectorization-essential
https://software.intel.com/en-us/articles/fortran-array-data-and-arguments-and-vectorization

intel®

Recap



Scalar Processing



Vector Processing

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.

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