intel® Look Inside™

Intel® Advisor XE Future Release

Threading Design & Prototyping
Vectorization Assistant

Parallel is the Path Forward

Intel® Xeon® and Intel® Xeon Phi™ Product Families are both going parallel

	Intel® Xeon® processor 64-bit	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor code-named Sandy Bridge EP	Intel® Xeon® processor code-named Ivy Bridge EP	Intel® Xeon® processor code-named Haswell EP	Intel® Xeon Phi™ coprocessor Knights Corner	Intel® Xeon Phi™ processor & coprocessor Knights Landing[1]
Core(s)	1	2	4	6	8	12	18	61	72
Threads	2	2	8	12	16	24	36	244	288
SIMD Width	128	128	128	128	256	256	256	512	512

*Product specification for launched and shipped products available on ark.intel.com.

1. Not launched or in planning.

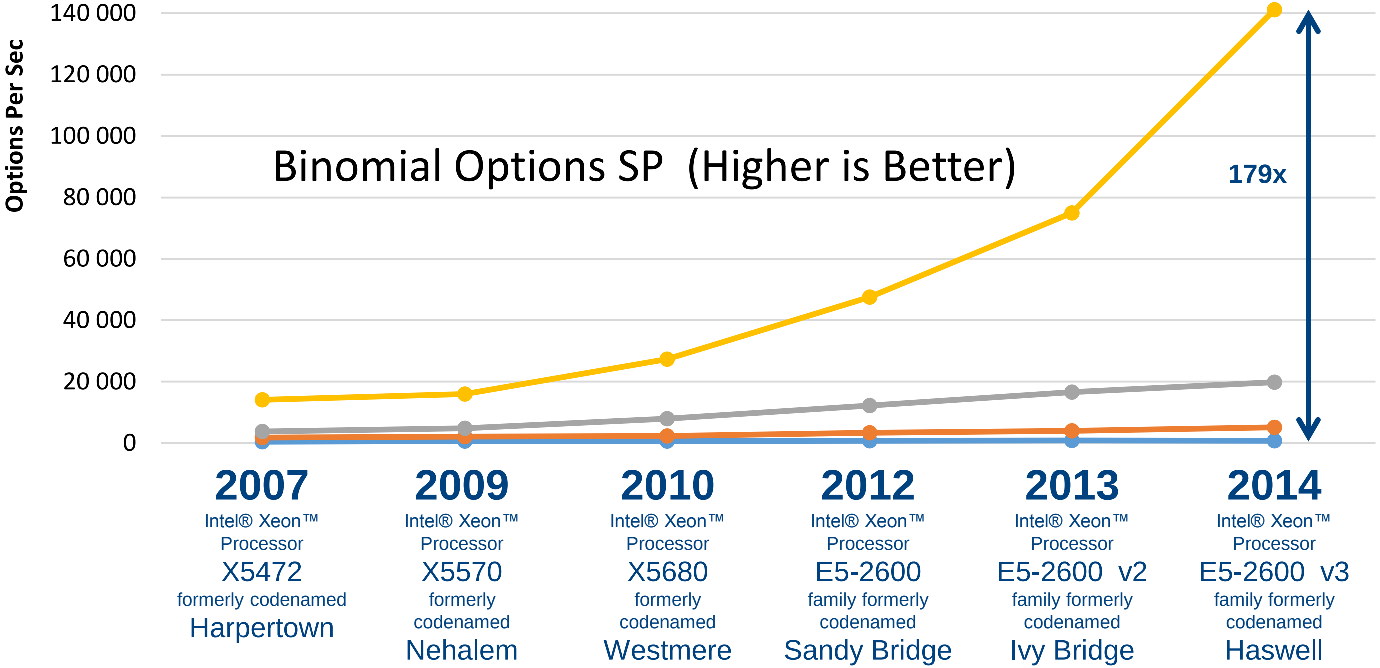
More cores → More Threads → Wider vectors

Don’t use a single Vector lane!

To fully utilize the hardware you need to:

- Parallelize and
- Vectorize

How much potential lies untapped today?



	Parallelized	**Vectorized**
🟨	✓	✓
⬜	✓	Scalar
🟧	Single Thread	✓
🟦	Single Thread	Scalar

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

Parallel + Vectorized is <u>much</u> faster than either one alone

Permission to design for all lanes

Threading and Vectorization

Intel Advisor XE:	Threading	Vectorization
Today	✓	
Future	✓	✓

Data Driven Threading Design

Intel® Advisor XE – Thread Prototyping

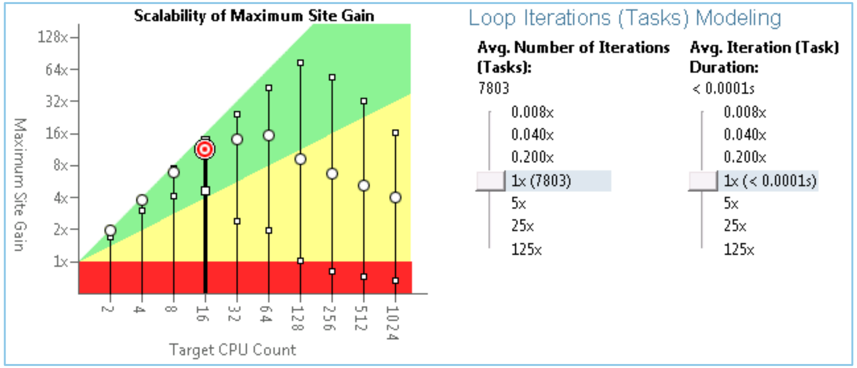
Have you:

- 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



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

Part of Intel® Parallel Studio
For Windows* and Linux* From $1,599

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

Simon Hammond
Senior Technical Staff
Sandia National Laboratories

http://intel.ly/advisor-xe

Data Driven Vectorization Design

Intel® Advisor XE – Vectorization Advisor (future release)

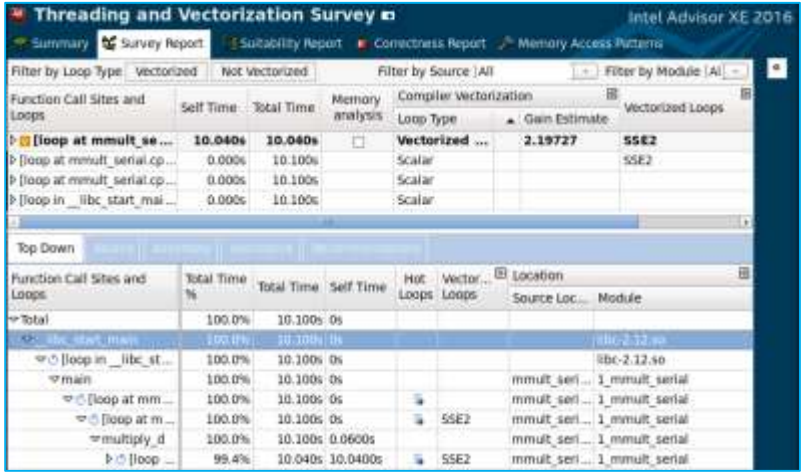
In Beta
soon.
Sign-up!

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?



Function Call Sites and Loops	Self Time	Total Time	Memory analysis	Compiler Vectorization		Vectorized Loops
				Loop Type	Gain Estimate	
[loop at mmult_se…	**10.040s**	**10.040s**	☐	**Vectorized …**	**2.19727**	**SSE2**
[loop at mmult_serial.cp…	0.000s	10.100s		Scalar		SSE2
[loop at mmult_serial.cp…	0.000s	10.100s		Scalar		
[loop in __libc_start_mai…	0.000s	10.100s		Scalar		

Function Call Sites and Loops	Total Time %	Total Time	Self Time	Hot Loops	Vector… Loops	Location	
						Source Loc…	Module
Total	100.0%	10.100s	0s				
__libc_start_main	100.0%	10.100s	0s				libc-2.12.so
[loop in __libc_st…	100.0%	10.100s	0s				libc-2.12.so
main	100.0%	10.100s	0s			mmult_seri…	1_mmult_serial
[loop at mm…	100.0%	10.100s	0s			mmult_seri…	1_mmult_serial
[loop at m…	100.0%	10.100s	0s		SSE2	mmult_seri…	1_mmult_serial
multiply_d	100.0%	10.100s	0.0600s			mmult_seri…	1_mmult_serial
[loop …	99.4%	10.040s	10.0400s		SSE2	mmult_seri…	1_mmult_serial

More Performance
Fewer Machine Dependencies

Vectorization Advisor

Providing 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.
- Convince the compiler to vectorize

Recommendations – How do I fix it?

Correctness via dependency analysis

- Is it safe to vectorize?
- Data on specific variable causing the loop dependency

Memory Access Patterns analysis

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

Vector Advisor Survey: all in one place

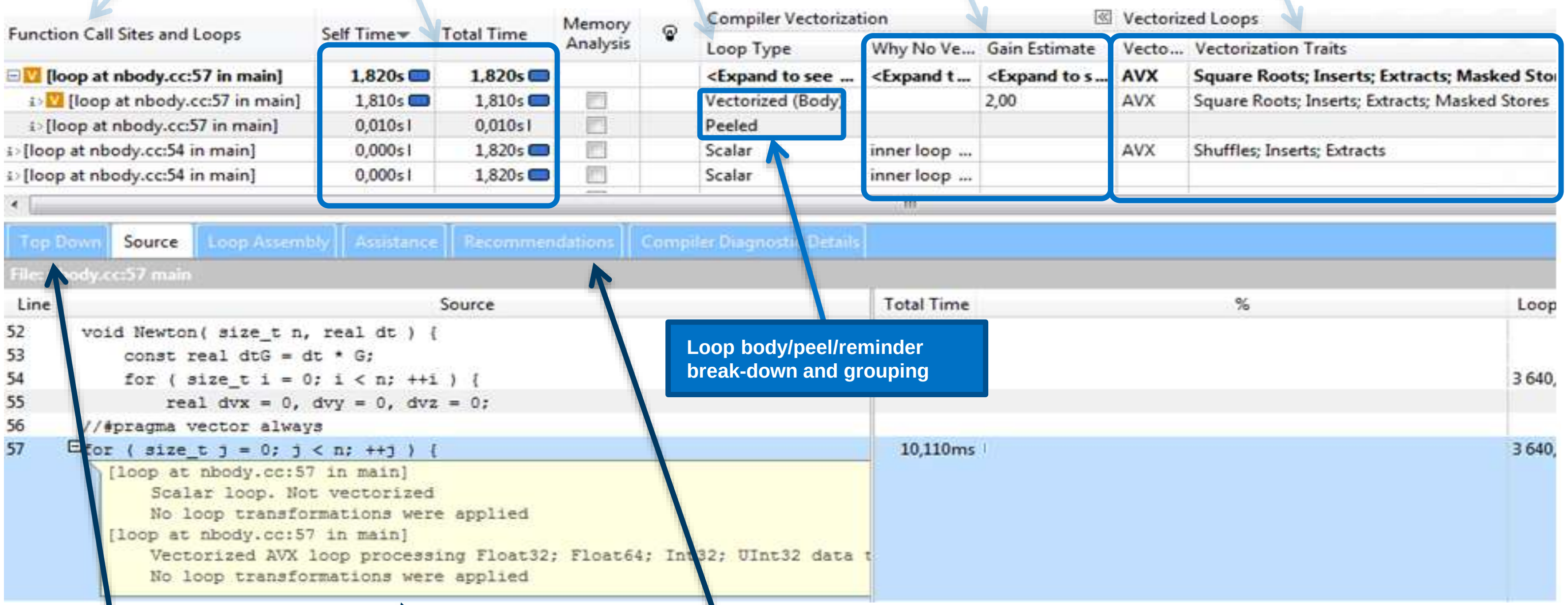
Vector indicator (filter)

Profile data (priority)

Loop type (filter)

Compiler data

Static analysis



Function Call Sites and Loops	Self Time	Total Time	Memory Analysis		Compiler Vectorization			Vectorized Loops	
					Loop Type	Why No Ve...	Gain Estimate	Vecto...	Vectorization 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 ...			

```
void Newton( size_t n, real dt ) {
    const real dtG = dt * G;
    for ( size_t i = 0; i < n; ++i ) {
        real dvx = 0, dvy = 0, dvz = 0;
//#pragma vector always
for ( size_t j = 0; j < n; ++j ) {
```

```
[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
```

Loop body/peel/reminder break-down and grouping

Top-down function-loops tree

Source tab

Suggestions and OpenMP4 snippets

What Makes Vectorization Difficult?

- Non-contiguous memory access – Potential to vectorize but may be inefficient
  - Non-unit strided access to arrays

```
for (i=0;i<N;i+=2) //Incrementing “i” by 2 is not unit-stride
```

  - Indirect reference in a loop

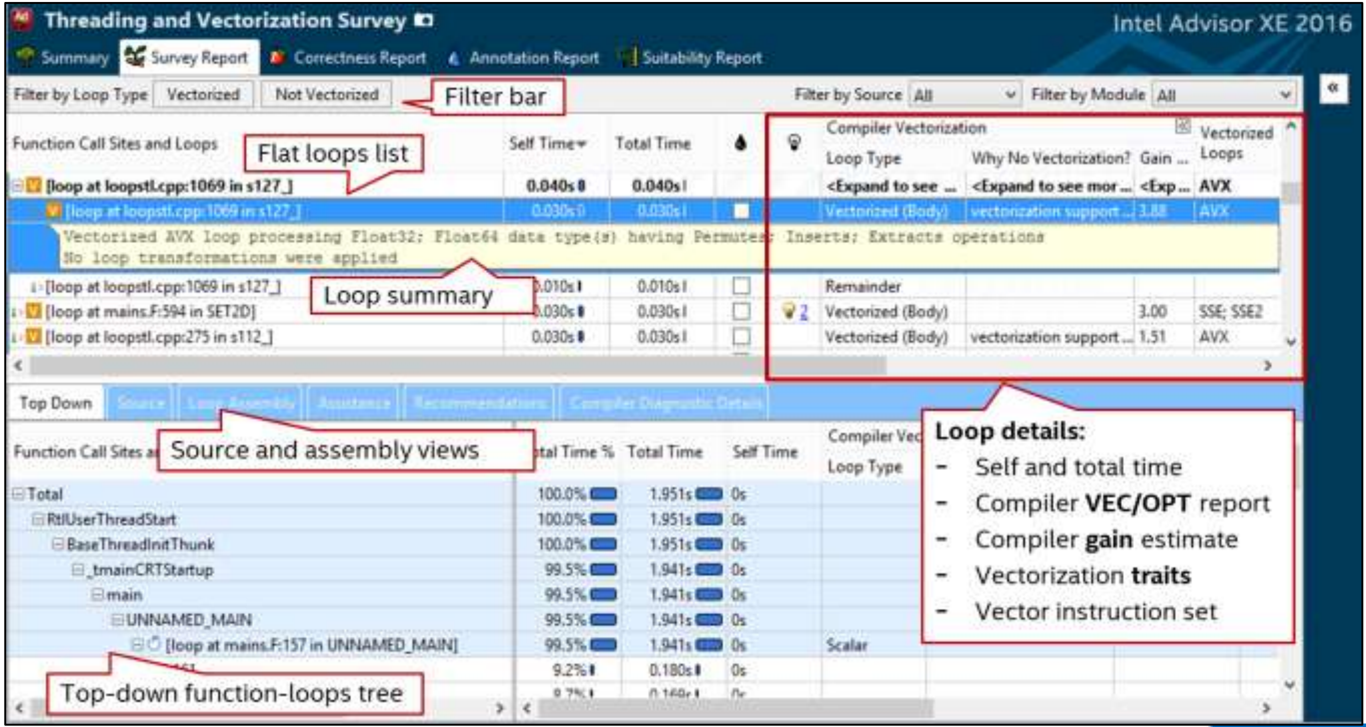
```
for (i=0;i<N;i++)
    A[B[i]] = C[i]*D[i];  //We have to decode B[i] to find out
                           //which element of A to reference
```

- Data dependencies

```
for (i=0;i<N;i++)
    A[i] = A[i-1]*C[i];
```

Compiler diagnostics + Performance Data Find “hot” un-vectorized or “under vectorized” loops

All of the information you require to vectorize available on one screen!



Filter bar

Flat loops list

					Compiler Vectorization			
Function Call Sites and Loops	Self Time	Total Time	💧	💡	Loop Type	Why No Vectorization?	Gain ...	Vectorized Loops
[loop at loopstl.cpp:1069 in s127_]	**0.040s**	**0.040s**			**<Expand to see ...**	**<Expand to see mor ...**	**<Exp ...**	**AVX**
[loop at loopstl.cpp:1069 in s127_]	0.030s	0.030s	☐		Vectorized (Body)	vectorization support ...	3.88	AVX
[loop at loopstl.cpp:1069 in s127_]	0.010s	0.010s	☐		Remainder			
[loop at mains.F:594 in SET2D]	0.030s	0.030s	☐	💡2	Vectorized (Body)		3.00	SSE; SSE2
[loop at loopstl.cpp:275 in s112_]	0.030s	0.030s	☐		Vectorized (Body)	vectorization support ...	1.51	AVX

```
Vectorized AVX loop processing Float32; Float64 data type(s) having Permutes; Inserts; Extracts operations
No loop transformations were applied
```

Loop summary

Source and assembly views

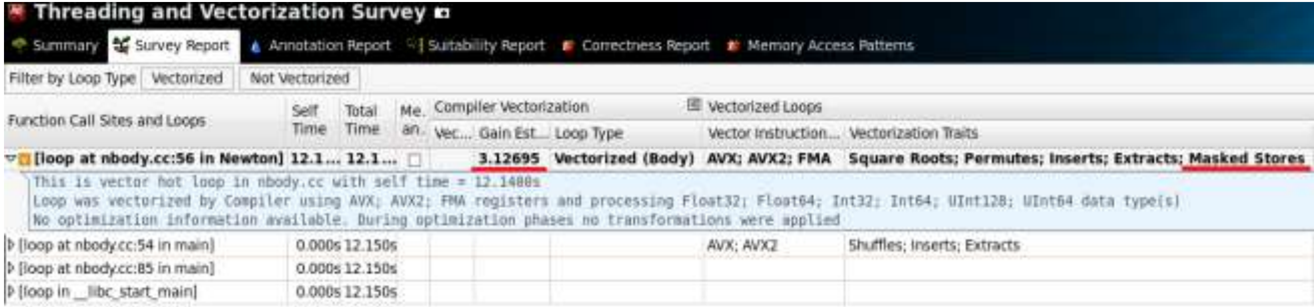
				Compiler Vec
Function Call Sites a	tal Time %	Total Time	Self Time	Loop Type
Total	100.0%	1.951s	0s	
RtlUserThreadStart	100.0%	1.951s	0s	
BaseThreadInitThunk	100.0%	1.951s	0s	
_tmainCRTStartup	99.5%	1.941s	0s	
main	99.5%	1.941s	0s	
UNNAMED_MAIN	99.5%	1.941s	0s	
[loop at mains.F:157 in UNNAMED_MAIN]	99.5%	1.941s	0s	Scalar
	9.2%	0.180s	0s	

Top-down function-loops tree

Loop details:
- Self and total time
- Compiler **VEC/OPT** report
- Compiler **gain** estimate
- Vectorization **traits**
- Vector instruction set

Gives estimated expected gain!

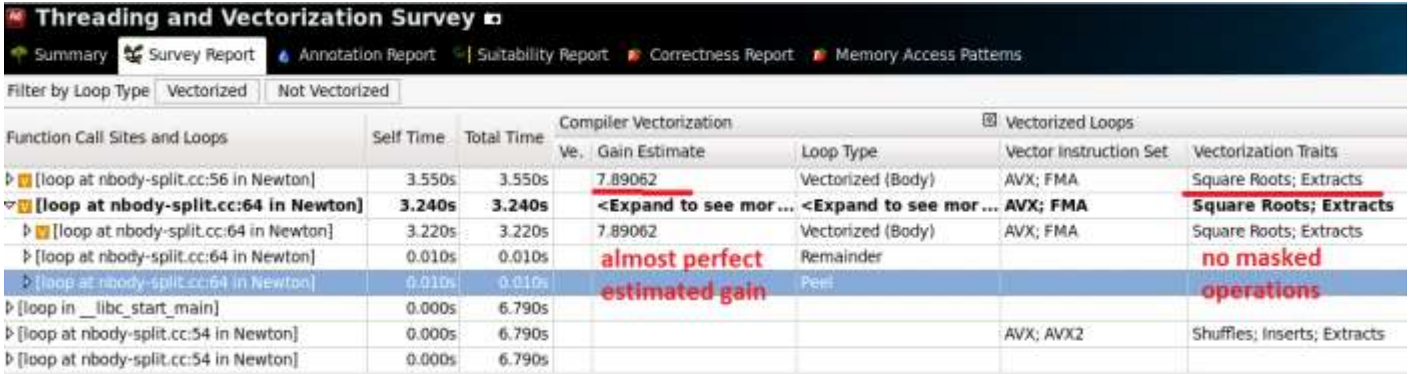
Gain estimates – Gives recommendations and the gain you can expect by using a different vector instruction or rewriting the control flow of your program.



Threading and Vectorization Survey

Function Call Sites and Loops	Self Time	Total Time	Me. an.	Compiler Vectorization Vec...	 Gain Est...	 Loop Type	Vectorized Loops Vector Instruction...	 Vectorization Traits
[loop at nbody.cc:56 in Newton]	**12.1...**	**12.1...**	☐		**3.12695**	**Vectorized (Body)**	**AVX; AVX2; FMA**	**Square Roots; Permutes; Inserts; Extracts; Masked Stores**
[loop at nbody.cc:54 in main]	0.000s	12.150s					AVX; AVX2	Shuffles; Inserts; Extracts
[loop at nbody.cc:85 in main]	0.000s	12.150s						
[loop in __libc_start_main]	0.000s	12.150s						

```
This is vector hot loop in nbody.cc with self time = 12.1480s
Loop was vectorized by Compiler using AVX; AVX2; FMA registers and processing Float32; Float64; Int32; Int64; UInt128; UInt64 data type(s)
No optimization information available. During optimization phases no transformations were applied
```



Threading and Vectorization Survey

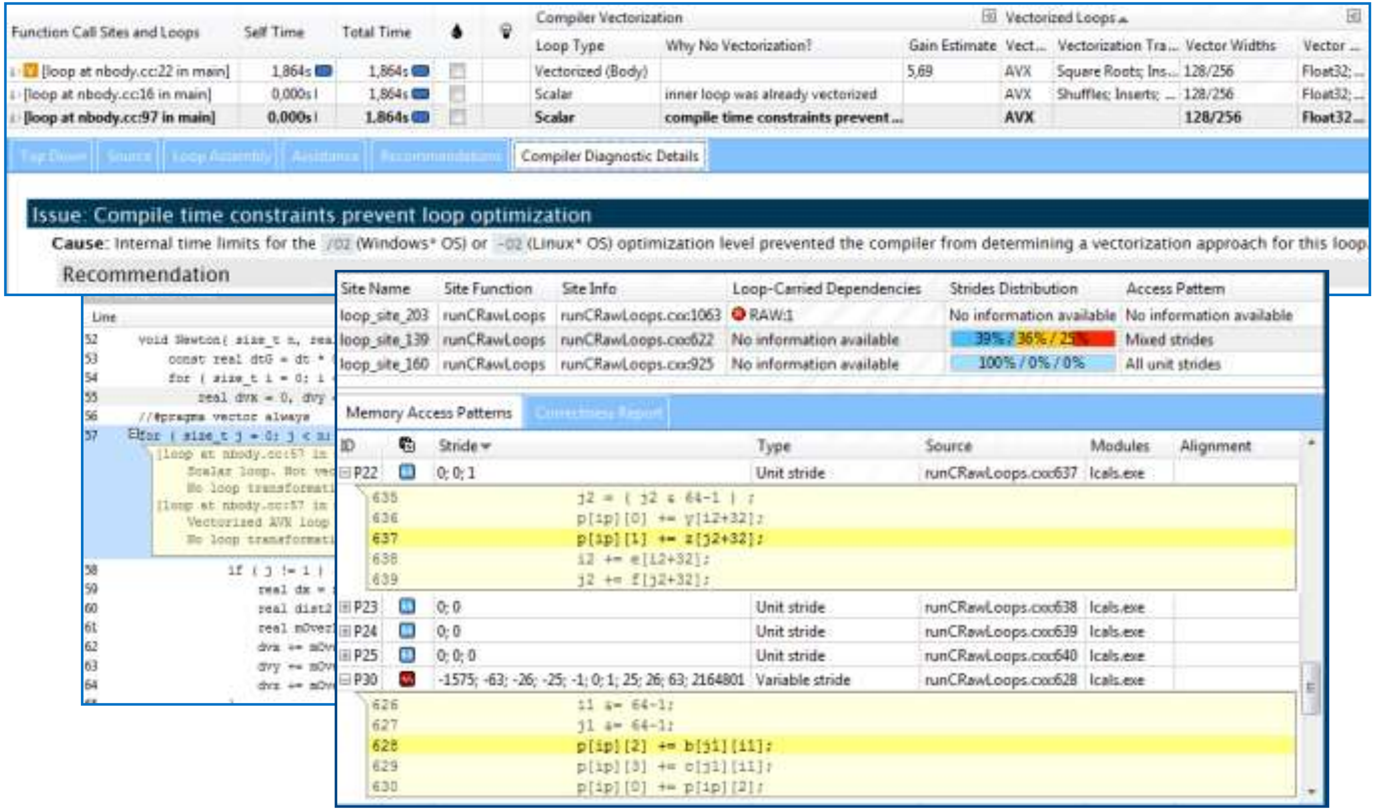
Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization Ve.	 Gain Estimate	 Loop Type	Vectorized Loops Vector Instruction Set	 Vectorization Traits
[loop at nbody-split.cc:56 in Newton]	3.550s	3.550s		7.89062	Vectorized (Body)	AVX; FMA	Square Roots; Extracts
[loop at nbody-split.cc:64 in Newton]	**3.240s**	**3.240s**		**<Expand to see mor...**	**<Expand to see mor...**	**AVX; FMA**	**Square Roots; Extracts**
[loop at nbody-split.cc:64 in Newton]	3.220s	3.220s		7.89062	Vectorized (Body)	AVX; FMA	Square Roots; Extracts
[loop at nbody-split.cc:64 in Newton]	0.010s	0.010s			Remainder		
[loop at nbody-split.cc:64 in Newton]	0.010s	0.010s			Peel		
[loop in __libc_start_main]	0.000s	6.790s					
[loop at nbody-split.cc:54 in Newton]	0.000s	6.790s				AVX; AVX2	Shuffles; Inserts; Extracts
[loop at nbody-split.cc:54 in Newton]	0.000s	6.790s					

almost perfect estimated gain

no masked operations

Vector Advisor:

- All the data in one place
  - Intel Compiler 15.x reports Integration
- Deep dive 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 ...
[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

```
Line
52      void Newton( size_t n, rea
53          const real dtG = dt * (
54          for ( size_t i = 0; i <
55              real dvx = 0, dvy
56      //#pragma vector always
57     ⊟for ( size_t j = 0; j < n
          [loop at nbody.cc:57 in
              Scalar loop. Not vec
              No loop transformati
          [loop at nbody.cc:57 in
              Vectorized AVX loop
              No loop transformati
58                  if ( j != 1 )
59                      real dx =
60                      real dist2
61                      real mOver
62                      dvx += mOv
63                      dvy += mOv
64                      dvz += mOv
```

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];
```

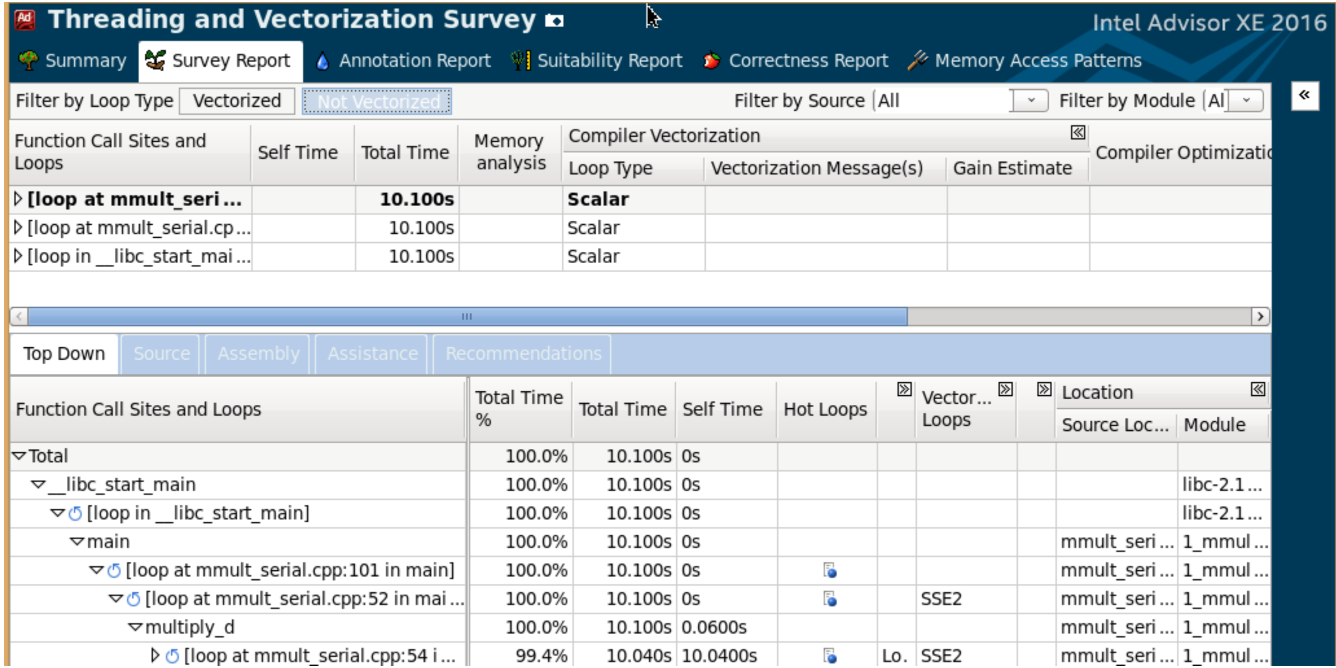
ID		Stride	Type	Source	Modules	Alignment
⊞ P23		0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
⊞ P24		0; 0	Unit stride	runCRawLoops.cxx:639	lcals.exe	
⊞ P25		0; 0; 0	Unit stride	runCRawLoops.cxx:640	lcals.exe	
⊟ P30		-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628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

```
626        i1 &= 64-1;
627        j1 &= 64-1;
628        p[ip][2] += b[j1][i1];
629        p[ip][3] += c[j1][i1];
630        p[ip][0] += p[ip][2];
```

Convince the compiler to vectorize

Unvectorized loops / “under vectorized” loops

- Assumed dependencies
- Control structures preventing vectorization.
- Rewrite loops to vectorize – remove conditions, breaks and returns and many other techniques.



Filter by Loop Type: Vectorized | Not Vectorized

Filter by Source: All

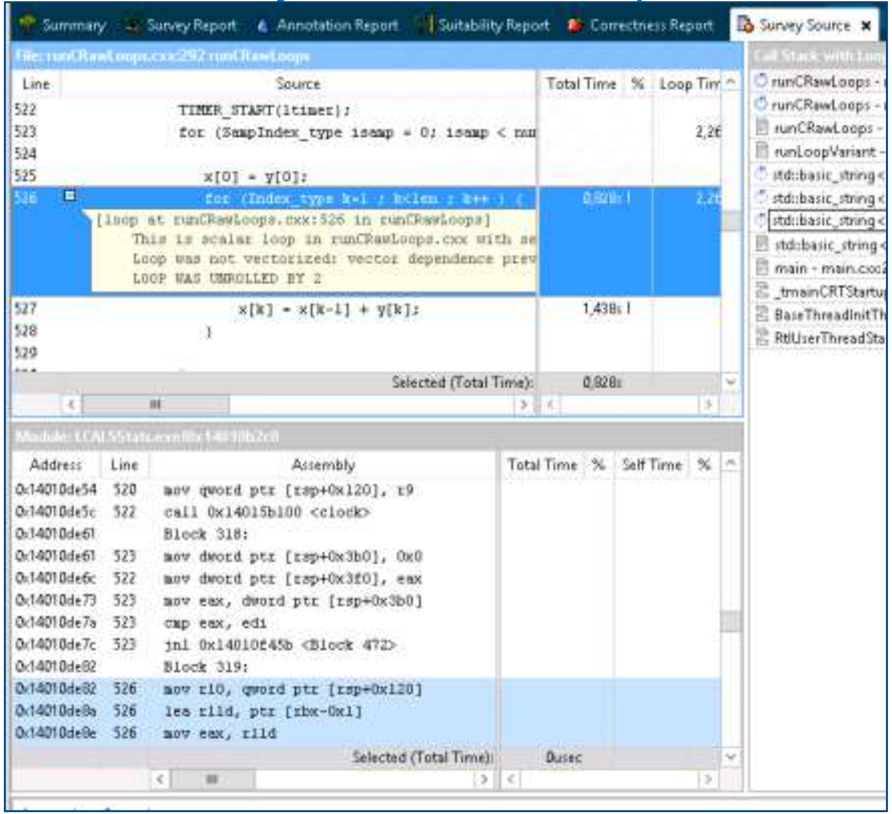
Filter by Module: Al

Function Call Sites and Loops	Self Time	Total Time	Memory analysis	Compiler Vectorization			Compiler Optimizatic
				Loop Type	Vectorization Message(s)	Gain Estimate	
[loop at mmult_seri…		**10.100s**		**Scalar**			
[loop at mmult_serial.cp…		10.100s		Scalar			
[loop in __libc_start_mai…		10.100s		Scalar			

Function Call Sites and Loops	Total Time %	Total Time	Self Time	Hot Loops		Vector… Loops		Location	
								Source Loc…	Module
Total	100.0%	10.100s	0s						
__libc_start_main	100.0%	10.100s	0s						libc-2.1…
[loop in __libc_start_main]	100.0%	10.100s	0s						libc-2.1…
main	100.0%	10.100s	0s					mmult_seri…	1_mmul…
[loop at mmult_serial.cpp:101 in main]	100.0%	10.100s	0s	●				mmult_seri…	1_mmul…
[loop at mmult_serial.cpp:52 in mai…	100.0%	10.100s	0s	●		SSE2		mmult_seri…	1_mmul…
multiply_d	100.0%	10.100s	0.0600s					mmult_seri…	1_mmul…
[loop at mmult_serial.cpp:54 i…	99.4%	10.040s	10.0400s	●	Lo.	SSE2		mmult_seri…	1_mmul…

Deep source and assembly integration

All the data you need in one place



File: runCRawLoops.cxx:292 runCRawLoops

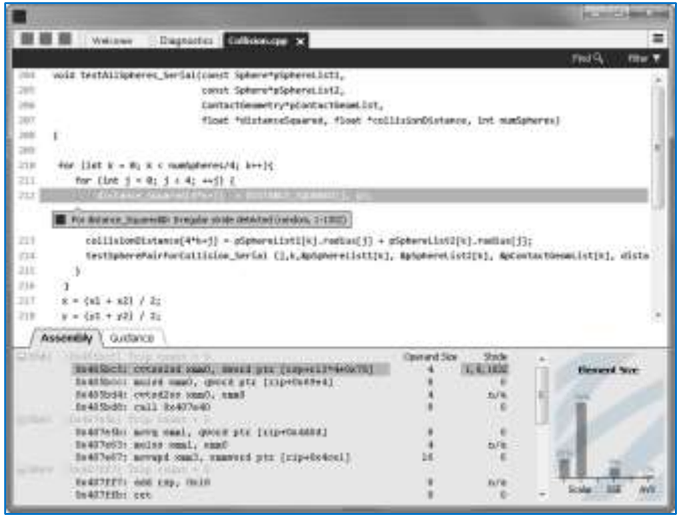
Line	Source	Total Time	%	Loop Tim
522	`TIMER_START(1timer);`			
523	`for (SampIndex_type isamp = 0; isamp < nu`			2,26
524				
525	`x[0] = y[0];`			
526	`for (Index_type k=1 ; k<len ; k++ ) {`	0,928s		2,2
527	`x[k] = x[k-1] + y[k];`	1,438s		
528	`}`			
529				

```
[loop at runCRawLoops.cxx:526 in runCRawLoops]
    This is scalar loop in runCRawLoops.cxx with se
    Loop was not vectorized: vector dependence prev
    LOOP WAS UNROLLED BY 2
```

Selected (Total Time): 0,928s

Address	Line	Assembly	Total Time	%	Self Time	%
0x14010de54	520	`mov qword ptr [rsp+0x120], r9`				
0x14010de5c	522	`call 0x14015b100 <clock>`				
0x14010de61		`Block 318:`				
0x14010de61	523	`mov dword ptr [rsp+0x3b0], 0x0`				
0x14010de6c	522	`mov dword ptr [rsp+0x3f0], eax`				
0x14010de73	523	`mov eax, dword ptr [rsp+0x3b0]`				
0x14010de7a	523	`cmp eax, edi`				
0x14010de7c	523	`jnl 0x14010f45b <Block 472>`				
0x14010de82		`Block 319:`				
0x14010de82	526	`mov r10, qword ptr [rsp+0x120]`				
0x14010de8a	526	`lea r11d, ptr [rbx-0x1]`				
0x14010de8e	526	`mov eax, r11d`				

Selected (Total Time): 0usec

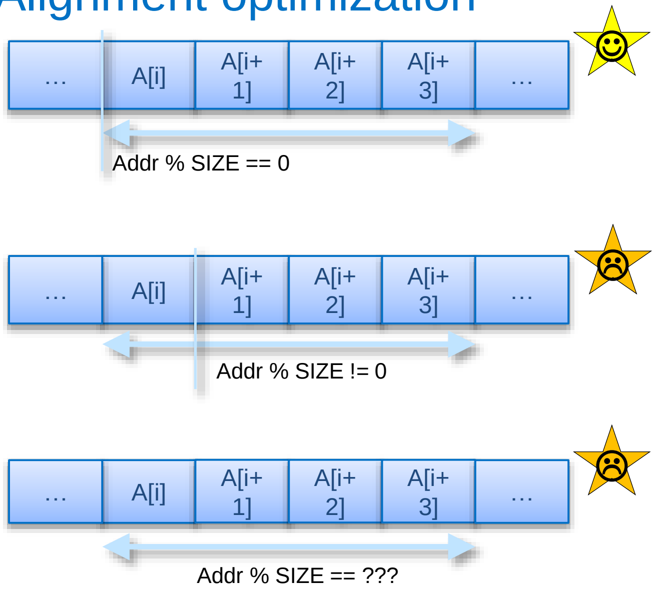


Optimizing…

Recommendations – How do I fix it?

SIZE:
64B for Intel® Xeon Phi™,
32B for AVX1/2,
16B for SSE4.2 and below

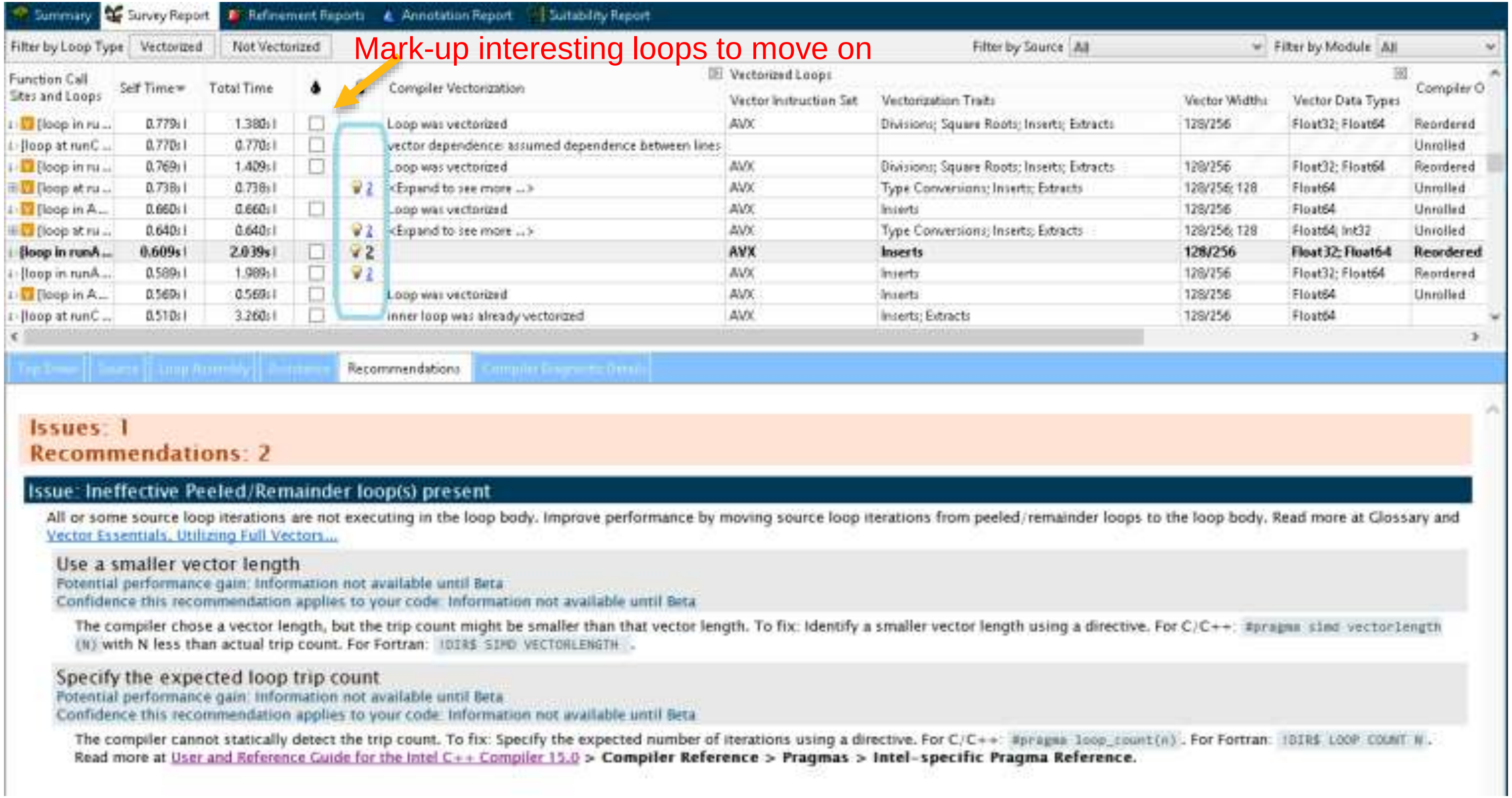
Alignment optimization



Peel/remainder

- Typical vectorized loop consists of
  - Optional “peel” part
    - Needed to improve alignment
    - Scalar or slower vector
  - Main vector part
    - Fastest among the three.
  - “remainder” part
    - Due to trip_count%VL != 0
    - Scalar or slower vector.
- Larger vector register means more iterations in peel/remainder
  - Align your data
  - Block to fight remainders

Recommendations



Mark-up interesting loops to move on

						Vectorized Loops				
Function Call Sites and Loops	Self Time	Total Time			Compiler Vectorization	Vector Instruction Set	Vectorization Traits	Vector Widths	Vector Data Types	Compiler O
[loop in ru...	0.779s	1.380s	☐		Loop was vectorized	AVX	Divisions; Square Roots; Inserts; Extracts	128/256	Float32; Float64	Reordered
[loop at runC...	0.770s	0.770s	☐		vector dependence: assumed dependence between lines					Unrolled
[loop in ru...	0.769s	1.409s	☐		Loop was vectorized	AVX	Divisions; Square Roots; Inserts; Extracts	128/256	Float32; Float64	Reordered
[loop at ru...	0.738s	0.738s		2	<Expand to see more ...>	AVX	Type Conversions; Inserts; Extracts	128/256; 128	Float64	Unrolled
[loop in A...	0.660s	0.660s	☐		Loop was vectorized	AVX	Inserts	128/256	Float64	Unrolled
[loop at ru...	0.640s	0.640s		2	<Expand to see more ...>	AVX	Type Conversions; Inserts; Extracts	128/256; 128	Float64; Int32	Unrolled
[loop in runA...	**0.609s**	**2.039s**	☐	**2**		**AVX**	**Inserts**	**128/256**	**Float32; Float64**	**Reordered**
[loop in runA...	0.589s	1.989s	☐	2		AVX	Inserts	128/256	Float32; Float64	Reordered
[loop in A...	0.569s	0.569s	☐		Loop was vectorized	AVX	Inserts	128/256	Float64	Unrolled
[loop at runC...	0.510s	3.260s	☐		inner loop was already vectorized	AVX	Inserts; Extracts	128/256	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...

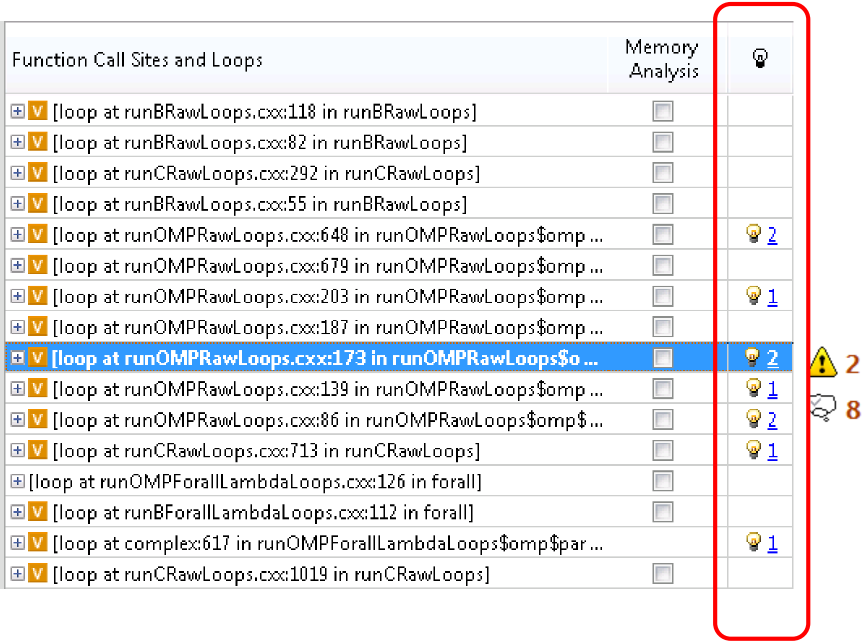
Use a smaller vector length
Potential performance gain: Information not available until Beta
Confidence this recommendation applies to your code: Information not available until Beta

The compiler chose a vector length, but the trip count might be smaller than that vector length. To fix: Identify a smaller vector length using a directive. For C/C++: `#pragma simd vectorlength (N)` with N less than actual trip count. For Fortran: `!DIR$ SIMD VECTORLENGTH` .

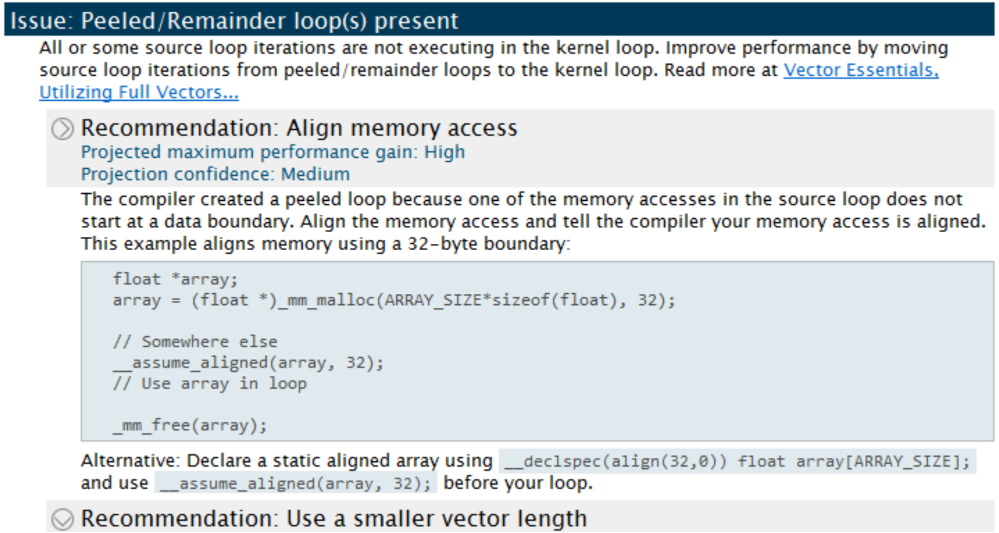
Specify the expected loop trip count
Potential performance gain: Information not available until Beta
Confidence this recommendation applies to your code: Information not available until Beta

The compiler cannot statically detect the trip count. To fix: Specify the expected number of iterations using a directive. For C/C++: `#pragma loop_count(n)` . For Fortran: `!DIR$ LOOP COUNT N` .
Read more at User and Reference Guide for the Intel C++ Compiler 15.0 > **Compiler Reference > Pragmas > Intel-specific Pragma Reference.**

End-user recommendations, performance penalties



Function Call Sites and Loops	Memory Analysis	💡
⊞ V [loop at runBRawLoops.cxx:118 in runBRawLoops]	☐	
⊞ V [loop at runBRawLoops.cxx:82 in runBRawLoops]	☐	
⊞ V [loop at runCRawLoops.cxx:292 in runCRawLoops]	☐	
⊞ V [loop at runBRawLoops.cxx:55 in runBRawLoops]	☐	
⊞ V [loop at runOMPRawLoops.cxx:648 in runOMPRawLoops$omp ...	☐	💡 2
⊞ V [loop at runOMPRawLoops.cxx:679 in runOMPRawLoops$omp ...	☐	
⊞ V [loop at runOMPRawLoops.cxx:203 in runOMPRawLoops$omp ...	☐	💡 1
⊞ V [loop at runOMPRawLoops.cxx:187 in runOMPRawLoops$omp ...	☐	
⊞ V **[loop at runOMPRawLoops.cxx:173 in runOMPRawLoops$o ...**	☐	💡 2
⊞ V [loop at runOMPRawLoops.cxx:139 in runOMPRawLoops$omp ...	☐	💡 1
⊞ V [loop at runOMPRawLoops.cxx:86 in runOMPRawLoopsomp ...	☐	💡 2
⊞ V [loop at runCRawLoops.cxx:713 in runCRawLoops]	☐	💡 1
⊞ [loop at runOMPForallLambdaLoops.cxx:126 in forall]	☐	
⊞ V [loop at runBForallLambdaLoops.cxx:112 in forall]	☐	
⊞ V [loop at complex:617 in runOMPForallLambdaLoops$omp$par ...		💡 1
⊞ V [loop at runCRawLoops.cxx:1019 in runCRawLoops]	☐	



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:

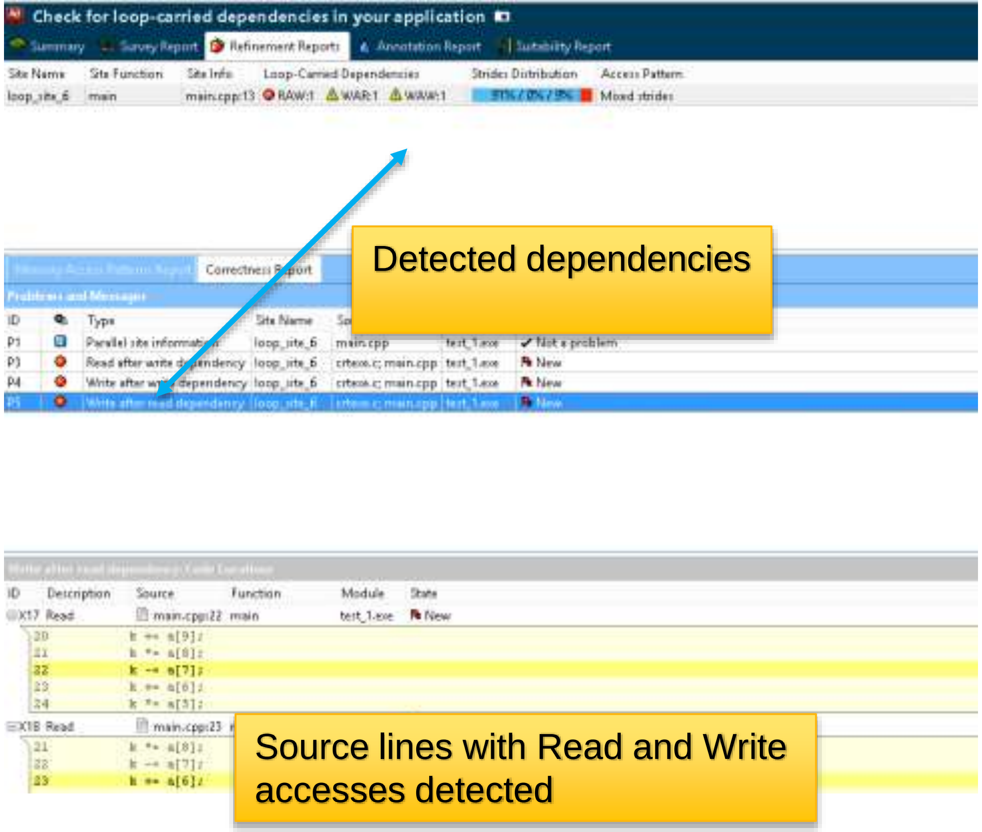
```
float *array;
array = (float *)_mm_malloc(ARRAY_SIZE*sizeof(float), 32);

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

_mm_free(array);
```

Alternative: Declare a static aligned array using `__declspec(align(32,0)) float array[ARRAY_SIZE];` and use `__assume_aligned(array, 32);` before your loop.

Recommendation: Use a smaller vector length

Correctness – Is It Safe to Vectorize?

Loop-carried dependencies analysis



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

ID	Type	Site Name	So		
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
X18	Read	main.cpp:23

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

Detected dependencies

Source lines with Read and Write accesses detected

Got recommendations to enforce vectorization of the loop:

1. Mark-up the loop and check for the presence of REAL dependencies
2. Explore dependencies in more details with code snippets

Are there dependencies in your loop preventing vectorization?

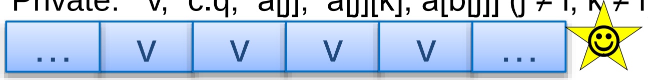
(if you force the compiler to vectorize this could generate incorrect code)

Optimizing…

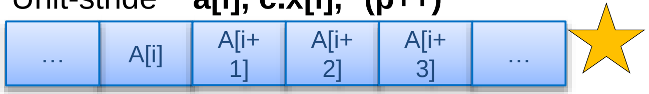
Memory Access Patterns – Data Layout Is Key

Vectorizing i-loop

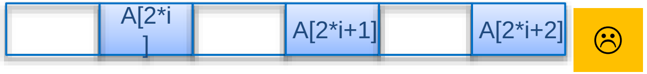
Private: v, c.q, a[j], a[j][k], a[b[j]] (j ≠ i, k ≠ i)



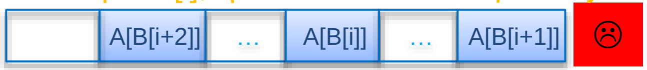
Unit-stride **a[i], c.x[i], *(p++)**



Non-unit-stride: a[2*i], c[i].x, a[i][j], a[i][0]
F90 Arrays in most cases
if not “contiguous”



Gather/scatter: j = b[i]; a[j], a[b.x[i]]
p = a[i]; *p – Intel® IMCI: especially slow

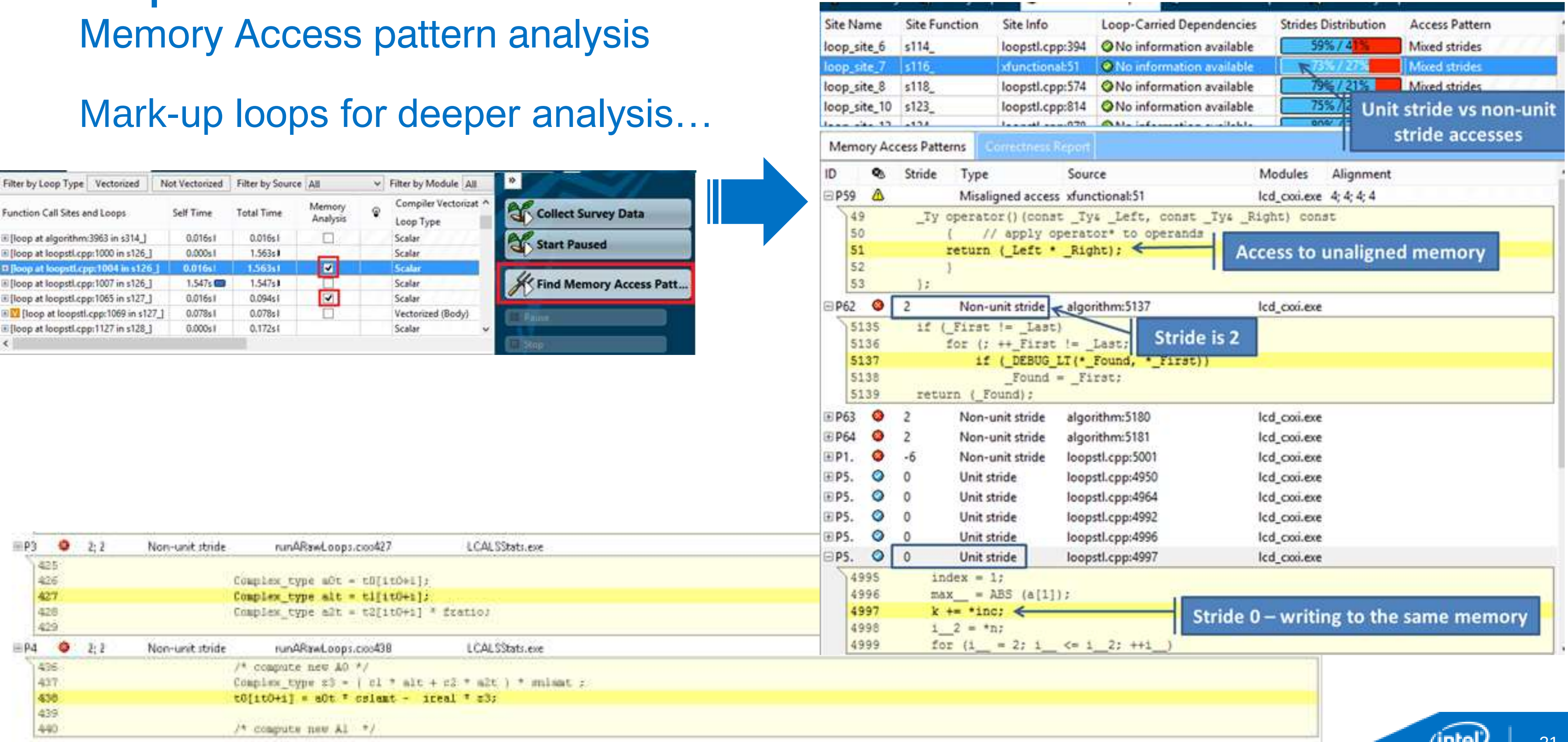


- Private – good
  - Almost no alignment requirements
  - Any addressing if…
    - Not depend on vectorizable loop index
- Unit-Stride
  - Good – with one exception.
  - Subject to vector alignment
  - Out-of-order cores won’t store-forward masked (unit-stride) store. ☹
  - On Intel® Xeon Phi™ correctness prevents efficient implementation of masked (unit-stride) store.
- Strided, Gather/Scatter is less efficient
  - Perf varies on micro-arch and the actual index patterns.
  - Big latency is exposed if you have these on the critical path
  - Better if done at outer loop level if loop nest is vectorized

Improve Vectorization

Memory Access pattern analysis

Mark-up loops for deeper analysis…



Function Call Sites and Loops	Self Time	Total Time	Memory Analysis	Compiler Vectorizat Loop Type
[loop at algorithm:3963 in s314_]	0.016s	0.016s	☐	Scalar
[loop at loopstl.cpp:1000 in s126_]	0.000s	1.563s		Scalar
[loop at loopstl.cpp:1004 in s126_]	0.016s	1.563s	☑	Scalar
[loop at loopstl.cpp:1007 in s126_]	1.547s	1.547s	☐	Scalar
[loop at loopstl.cpp:1065 in s127_]	0.016s	0.094s	☑	Scalar
[loop at loopstl.cpp:1069 in s127_]	0.078s	0.078s	☐	Vectorized (Body)
[loop at loopstl.cpp:1127 in s128_]	0.000s	0.172s		Scalar

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_6	s114_	loopstl.cpp:394	No information available	59% / 41%	Mixed strides
loop_site_7	s116_	xfunctional:51	No information available	73% / 27%	Mixed strides
loop_site_8	s118_	loopstl.cpp:574	No information available	79% / 21%	Mixed strides
loop_site_10	s123_	loopstl.cpp:814	No information available	75% /	

Unit stride vs non-unit stride accesses

ID	Stride	Type	Source	Modules	Alignment
P59		Misaligned access	xfunctional:51	lcd_cxxi.exe	4; 4; 4; 4

```
_Ty operator()(const _Ty& _Left, const _Ty& _Right) const
    {   // apply operator* to operands
    return (_Left * _Right);
    }
};
```

Access to unaligned memory

ID	Stride	Type	Source	Modules	Alignment
P62	2	Non-unit stride	algorithm:5137	lcd_cxxi.exe	

```
if (_First != _Last)
    for (; ++_First != _Last; )
        if (_DEBUG_LT(*_Found, *_First))
            _Found = _First;
return (_Found);
```

Stride is 2

ID	Stride	Type	Source	Modules	Alignment
P63	2	Non-unit stride	algorithm:5180	lcd_cxxi.exe	
P64	2	Non-unit stride	algorithm:5181	lcd_cxxi.exe	
P1.	-6	Non-unit stride	loopstl.cpp:5001	lcd_cxxi.exe	
P5.	0	Unit stride	loopstl.cpp:4950	lcd_cxxi.exe	
P5.	0	Unit stride	loopstl.cpp:4964	lcd_cxxi.exe	
P5.	0	Unit stride	loopstl.cpp:4992	lcd_cxxi.exe	
P5.	0	Unit stride	loopstl.cpp:4996	lcd_cxxi.exe	
P5.	0	Unit stride	loopstl.cpp:4997	lcd_cxxi.exe	

```
index = 1;
max__ = ABS (a[1]);
k += *inc;
i__2 = *n;
for (i__ = 2; i__ <= i__2; ++i__)
```

Stride 0 – writing to the same memory

ID	Stride	Type	Source	Modules
P3	2; 2	Non-unit stride	runARawLoops.cxx:427	LCALSStats.exe

```
Complex_type a0t = t0[it0+i];
Complex_type a1t = t1[it0+i];
Complex_type a2t = t2[it0+i] * fratio;
```

ID	Stride	Type	Source	Modules
P4	2; 2	Non-unit stride	runARawLoops.cxx:438	LCALSStats.exe

```
/* compute new A0 */
Complex_type z3 = ( c1 * a1t + c2 * a2t ) * snlamt ;
t0[it0+i] = a0t * cslamt -  ireal * z3;

/* compute new A1  */
```

Vectorization Advisor

Providing 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.
- Convince the compiler to vectorize

Recommendations – How do I fix it?

Correctness via dependency analysis

- Is it safe to vectorize?
- Data on specific variable causing the loop dependency

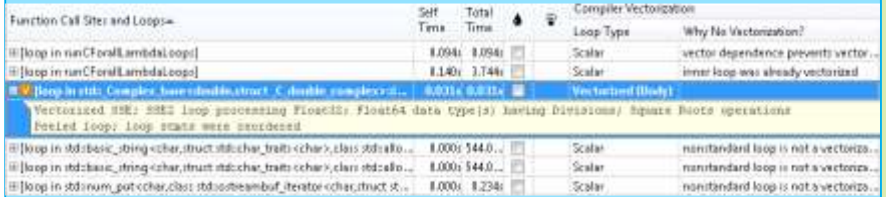
Memory Access Patterns analysis

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

Summary: Vector Advisor Alpha

4 Analysis Features for Efficient Vectorization

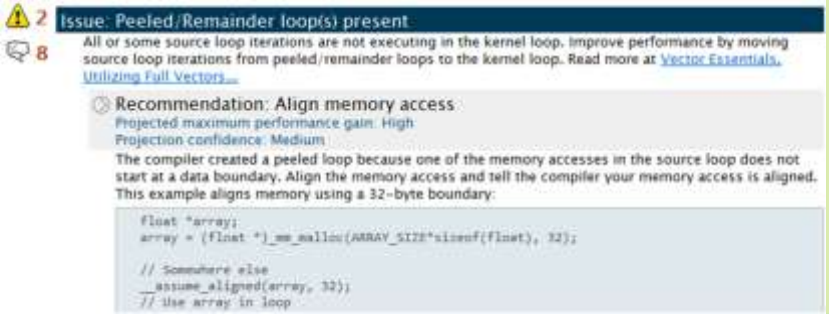
1. Compiler diagnostics with Performance Data



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>::d...	0.031s	0.031s	☐		Vectorized (Body)	
[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...

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

2. Recommendations on how to improve vectorization



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

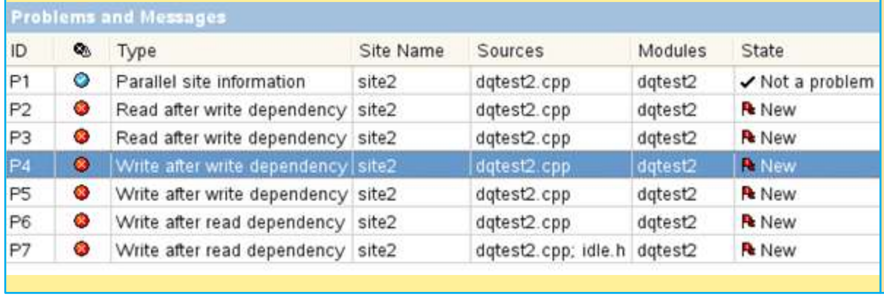
Projected maximum performance gain: High

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
```

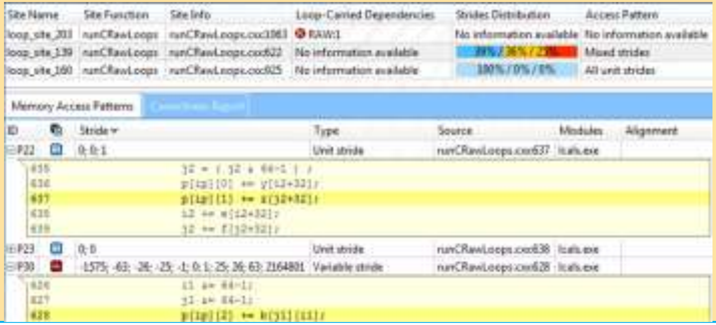
3. Correctness Dependency Analysis



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	

P22:

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

P30:

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

Intel® Advisor XE is part of Intel® Parallel Studio XE

Intel® Parallel Studio XE 2015 Composer Edition	Intel® Parallel Studio XE 2015 Professional Edition	Intel® Parallel Studio XE 2015 Cluster Edition
Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*	Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*	Intel® C++ Compiler Intel® Fortran Compiler Intel® Threading Building Blocks Intel® Integrated Performance Primitives Intel® Math Kernel Library Intel® Cilk™ Plus Intel® OpenMP*
	Intel® Advisor XE Intel® Inspector XE Intel® VTune™ Amplifier XE	**Intel® Advisor XE** Intel® Inspector XE Intel® VTune™ Amplifier XE Intel® MPI Library Intel® Trace Analyzer and Collector

For more information: http://intel.ly/perf-tools

Join the beta!

Intel® Advisor XE – Vectorization Advisor

Send e-mail to vector_advisor@intel.com to participate in the Vectorization Advisor beta.

Limited alpha access is available now under NDA

Public beta is coming late Q1 or Q2 2015

Sign-up now and we will contact you when we have more details.

Legal Disclaimer & Optimization Notice

Optimization Notice