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

Zakhar A. Matveev, PhD,

Intel Russia, Intel Software and Services Group

November‘ 2015

Code modernization: Intel® Parallel Studio XE 2016

Intel® Parallel Studio XE 2016 Suite

Vectorization – Boost Performance By Utilizing Vector Instructions / Units

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

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

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

Big Data Analytics – Easily Build IA Optimized Data Analytics Application

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

Standards – Scaling Development Efforts Forward

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

Intel® Advisor XE

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

Where is a problem?

Loop-carried dependencies

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

Function calls

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

Pointer aliasing

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

Unknown loop iteration count

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

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

Indirect memory access

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

Outer loops

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

Vectorization Advisor: part of Intel® Advisor XE

Assist code modernization for x86 SIMD

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	T
[loop in runCForallLambdaLoops]	0.094s	
[loop in runCForallLambdaLoops]	0.140s	
[loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0
Vectorized SSE; SSE2 loop processing Float32; Float64 data type		
Peeled loop; loop stmts were reordered		
[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	

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

Issue: Peeled/Remainder loop(s) present

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

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

...oat), 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

Memory Access Patterns | Correctness Report

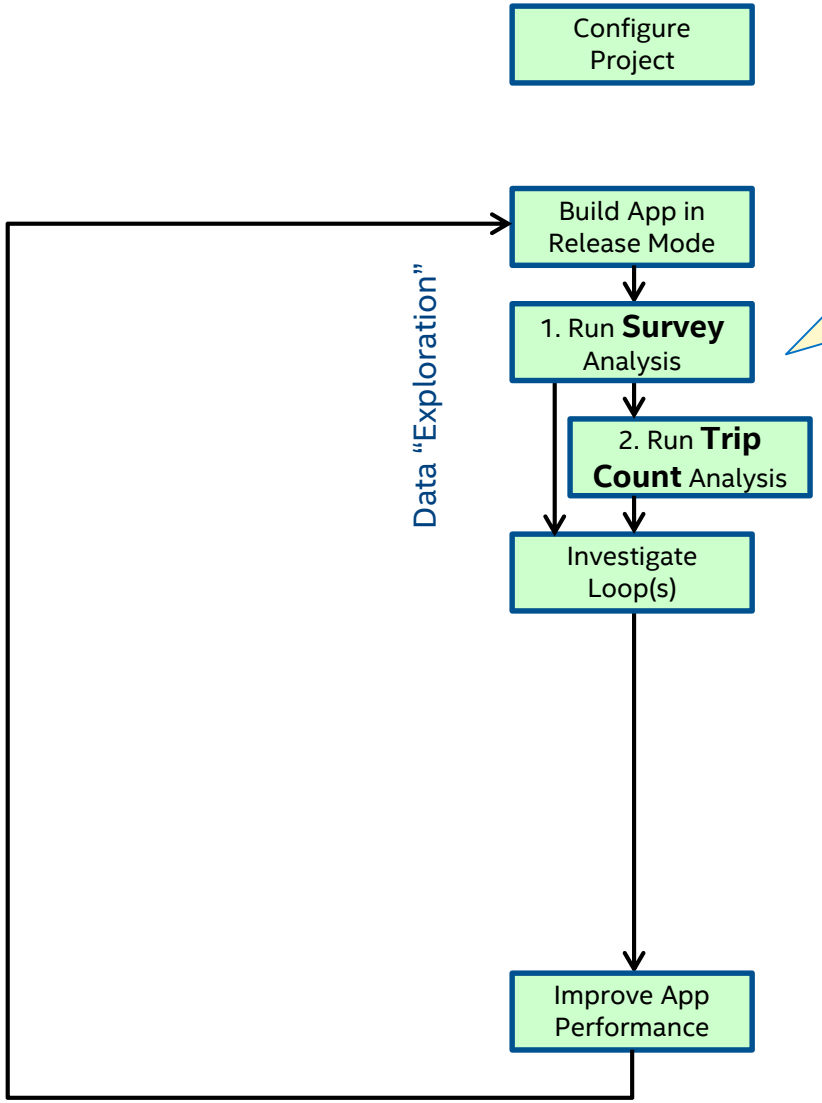
ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	

```
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];
```

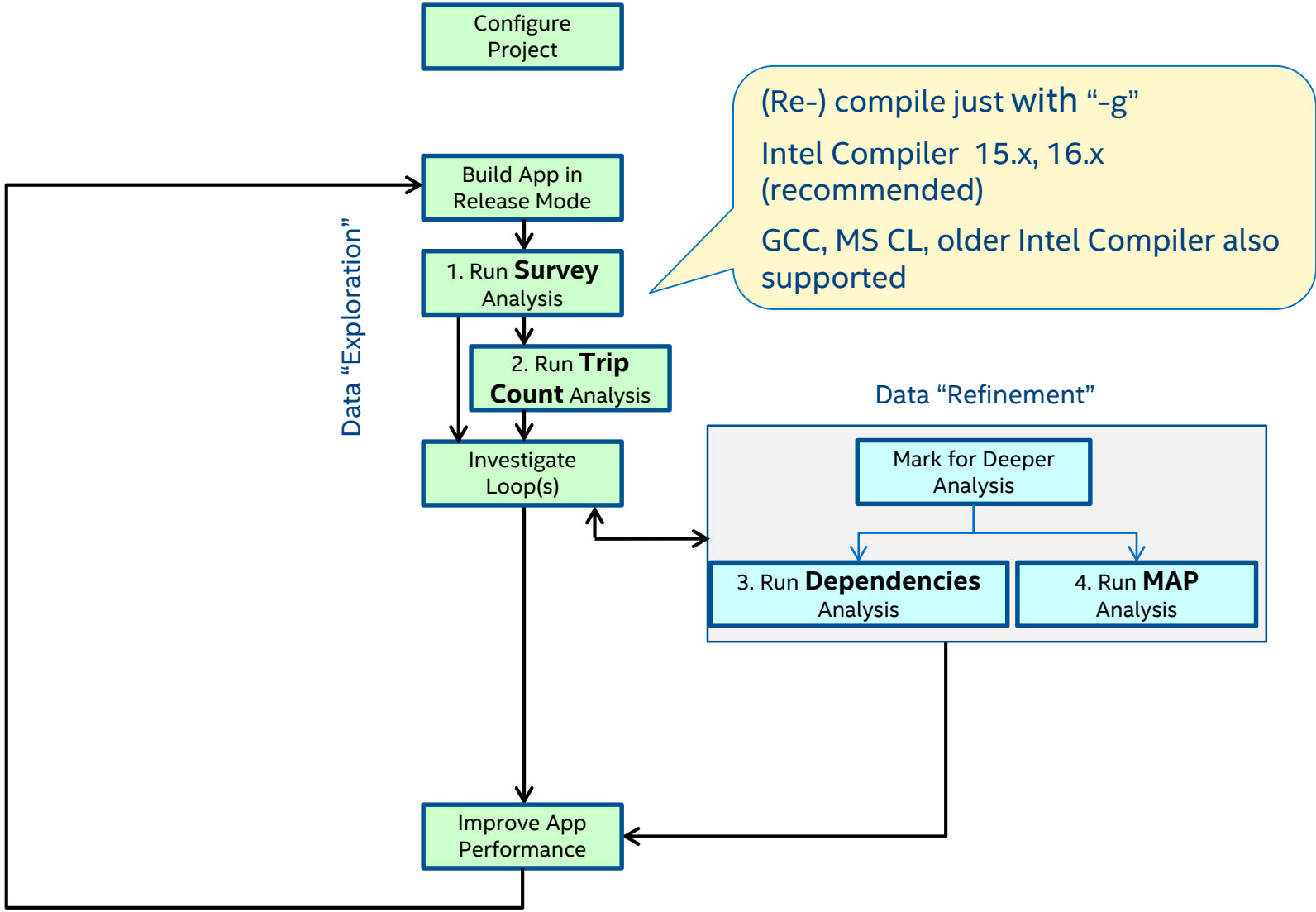
0.
Workflow



(Re-) compile just with "-g"

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

GCC, MS CL, older Intel Compiler also supported



Vector Advisor

C, C++, Fortran, C#

Windows, Linux

- 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: Loop Type	Why No Vectorization?	Gain Estimate	Vectorized Loops: 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...**

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

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, r
53          const real dtG = dt
54          for ( size_t i = 0;
55              real dvx = 0, dv
56      //#pragma vector always
57      for ( size_t j = 0; j <
        [loop at nbody.cc:57 i
            Scalar loop. Not v
            No loop transforma
        [loop at nbody.cc:57 i
            Vectorized AVX lo
            No loop transforma
58              if ( j != i
59                  real dx
60                  real dis
61                  real mOv
62                  dvx += m
63                  dvy += m
64                  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

Memory Access Patterns | Correctness Report

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	
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	

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

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

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
[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 ...

Loop Vectorization: loop versions

A typical vectorized loop consists of

Main vector body

- Fastest among the three!

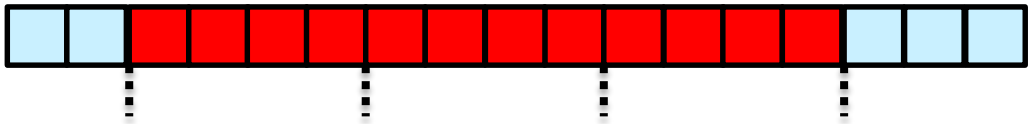
This is where we want our loops to be executing!

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



data is aligned since iteration 3

Peel Vector iteration Vector iteration Vector iteration Remainder

- Alignment: make sure you Align your data!
- Padding: make number of iterations divisible by the vector length!

The Right Data At Your Fingertips

Get all the data you need for high impact vectorization

Filter by which loops are vectorized!

Trip Counts

What prevents vectorization?

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

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

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

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops: Vecto...	Vectorized Loops: Efficiency	Vectorized Loops: Vector L...
[loop at stl_algo.h:4740 in std::tr...		0.170s	0.170s		Scalar	non-vectorizable loop ins...			
[loop at loopstl.cpp:2449 in s234_]	2 Ineffective peeled/rem..	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop 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 prev...			

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

Don't Just Vectorize, Vectorize Efficiently

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

Where should I add vectorization and/or threading parallelism?

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

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

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

Get **Specific Advice** For Improving Vectorization

Intel® Advisor XE – Vectorization Advisor

Click to see recommendation

[loop in fGetOneMassSite at I...		1 Ineffective peeled/remainder loop(s) present	AVX	~33%	1,31x	4	1,31x	0,267s	8,3%
[loop in fGetSpeedSite at lbp		1 Data type conversions present	AVX2	~100%	5,14x	4	5,14x	0,249s	7,7%

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

Issue: Ineffective peeled/remainder loop(s) present

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

Recommendation: Collect trip counts data — Confidence: Need More Data

Recommendation: Specify the expected loop trip count — Confidence: Low

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

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

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

Read More:

- loop_count
- Getting Started with Intel Compiler Pragmas and Directives and ... Resources for Intel® Advisor users

Advisor XE shows hints how to decrease vectorization overhead

Recommendation: Enforce vectorized remainder — Confidence: Low

Recommendation: Use a smaller vector length — Confidence: Low

Recommendation: Align data — Confidence: Low

Recommendation: Add data padding — Confidence: Low

Vector Efficiency: my performance thermometer

all the data in one place

Elapsed time: 4.31s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources

Loops	Self Time	Total Time	Loop Type	Vectorized Loops: Vect...	Efficiency	Gain...	VL (...	Vector Issues	Trip Counts	Instruction Set...: Traits	Data T...	Advanced
[loop in fCollisionBGK at lbp ...	0.016s	0.016s	Vectorized (Remainder)	AVX	80%	1.59x	2	1 Ineffecti ...	6		Float64	
[loop in fGetEquilibriumF at lb ...	0.516s	0.703s	Vectorized (Body; Peeled; Remainder)	AVX	39%	1.55x	4	2 Possible i ...	4; 1; 2	Inserts; ...	Float64...	Unrolled by 1
[loop in fGetFracSite at lbpGE ...	0.047s	0.047s	Vectorized Versions	AVX	21%							
[loop in fGetOneDirecSpeedSi ...	0.016s	0.031s	Vectorized (Body; Peeled)	AVX	14%							
[loop in fGetOneMassSite at l ...	0.047s	0.078s	Vectorized (Body; Peeled; Remainder)	AVX	8%							

39%

39%: Achieved Vectorization Efficiency

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

Estimated Gain = 1.55x

Vector Length = 4

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

(25%): Reference Efficiency for original scalar loop

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

(100%): Theoretical Maximum Vectorization Efficiency

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

Theoretical Maximum Gain = Currently selected Vector Length = 4

- **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

Fortran Example

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

Summary | Survey Report | Survey Source: loops90.f | Refinement Reports | Annotation Report | Suitability Report

Elapsed time: 46.30s | Vectorized | Not Vectorized | FILTER: All Modules | All Sources

Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?
[loop at mains.F:587 in SET2D]		0.000s	0.040s	550	Scalar	
[loop at loops90.f:2034 in s3110_ompparallel@2033]		0.000s	0.123s	1000	Scalar	
[loop at mains.F:775 in nindex_]		0.000s	0.006s	67	Scalar	loop control variable .
[loop at loops90.f:2034 in s3110_ompparallel@2033]		n/a	n/a		Scalar Completely ...	vectorization possible
[loop at loops90.f:1773 in S2102]						
[loop at loops90.f:2301 in S343]						
[loop at loops90.f:448 in S126]						
[loop at loops90.f:573 in s141_ompparallel_for@570]						
[loop at loops90.f:908 in S222]						
[loop at loops90.f:2716 in S442]						
[loop at loops90.f:213 in S114]						
[loop at loops90.f:1313 in S258]						

Top Down | Source | Loop Assembly | Assistance | Recomme

Function Call Sites and Loops	Total Time %	T
Total	100.0%	
RtlUserThreadStart	99.9%	
BaseThreadInitThunk	99.9%	
_tmainCRTStartup	83.3%	
main	83.3%	
UNNAMED_MAIN	83.3%	
S2101	7.8%	
S252	6.4%	
S343	5.2%	
S126	4.6%	
S451	3.2%	
S471	2.1%	
S442	1.9%	
S352	1.8%	
S222	1.6%	
S118	1.4%	0.705s 0s
S353	1.2%	0.610s 0s

File: loops90.f:2301 S343

Line	Source	Total Time	%	Loop Time	%
2293	real t1,t2,chksum,ctime,dtime,cs1d,array				
2294	parameter(nn= 1000)				
2295	common/cdata /array(nn*nn)				
2296	call init(ld,n,a,b,c,d,e,aa,bb,cc,'s343 ')				
2297	call forttime(t1)				
2298	do nl= 1,ntimes/n			2.647s	
2299	k= 0				
2300	do i= 1,n			2.647s	
2301	do j= 1,n	0.040s		2.647s	
2302	if(bb(i,j) > 0)then	1.364s			
2303	k= k+1	0.043s			
2304	array(k)= aa(i,j)	1.200s			
2305	endif				
2306	enddo				
2307	enddo				
2308	call dummy(ld,n,a,b,c,d,e,aa,bb,cc,1.)				
2309	enddo				
2310	call forttime(t2)				
	Selected (Total Time):	0s			

Why no vectorization?

Assumed dependencies – Run correctness check to verify dependency

Function call prevents vectorization

Loop with multiple exits

Inner loop already vectorized

- Potential to force vectorization of outer loop

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization: Loop Type	Compiler Vectorization: Why No Vectorization?
[loop in runCForallLambdaLoops]	0.094s	0.094s	Scalar	vector dependence prevents vector ...
[loop in runCForallLambdaLoops]	0.140s	3.744s	Scalar	inner loop was already vectorized
[loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop 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
```

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

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

Data Dependencies – Tough Problem #1

Is it safe to force the compiler to vectorize?

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

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

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**

Is It Safe to Vectorize?

Loop-carried dependencies analysis verifies correctness

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

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

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

Function Call Sites and Loops	Self Time	Total Time			Trip Counts	Compiler Vectorization	
						Loop Type	Why No Vectorization?
[loop at Multiply.c:53 in matvec]	0.047s	0.047s	☐		3	Vectorized (Body)	
[loop at Multiply.c:53 in matvec]	0.413s	0.413s	☐		101	Scalar	
[loop at Multiply.c:45 in matvec]	0.109s	12.373s		1		Collapse	Collapse
[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.

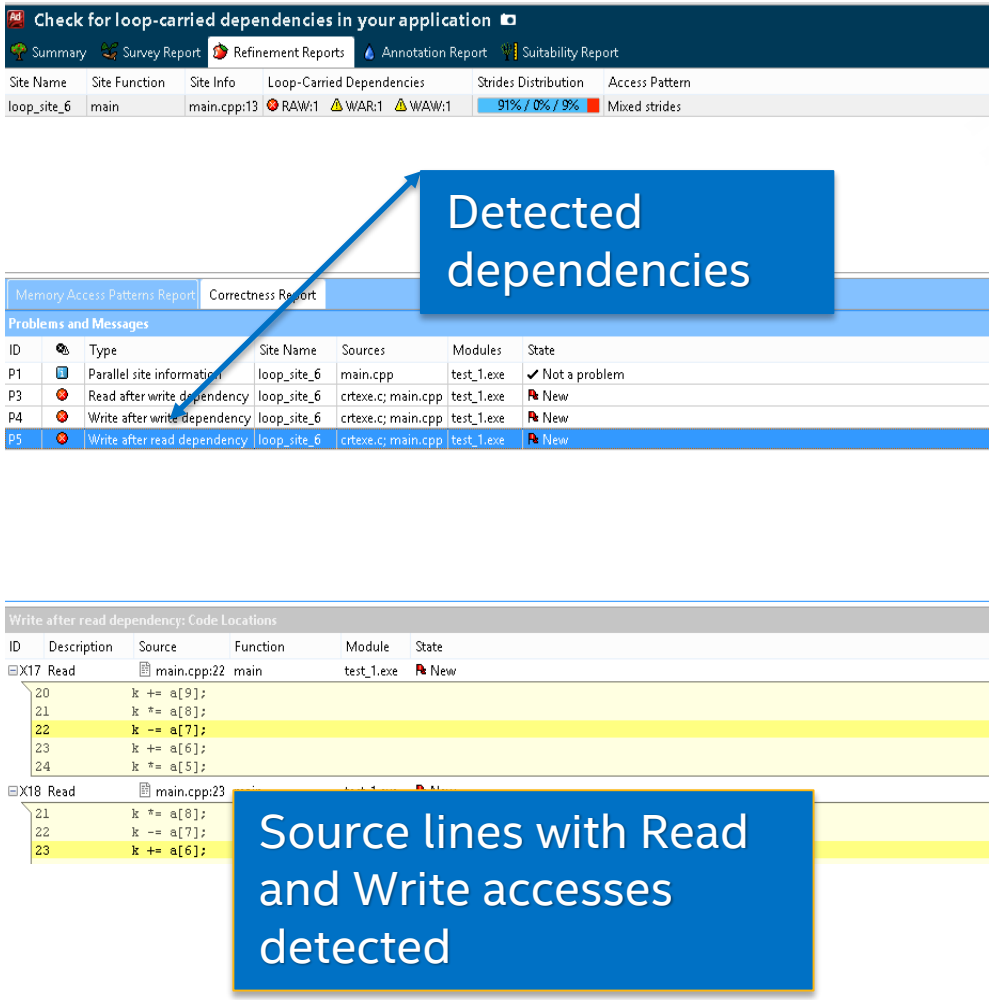
ICL/ICC/ICPC Directive	IFORT Directive	Outcome
#pragma simd or #pragma omp simd	!DIR$ SIMD or !$OMP SIMD	Ignores all dependencies in the loop
#pragma ivdep	!DIR$ IVDEP	Ignores only vector dependencies (which is safest)

Read More:

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

Correctness – Is It Safe to Vectorize?

Loop-carried dependencies analysis



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

Almost half loops checked did not have actual dependencies

We should investigate using pragma simd to force vectorization

Check memory access patterns in your application

Summary | Survey Report | Survey Source: loops90.f | Refinement Reports | Annotation Report | Suitability Report

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
loop at loops90.f:1261 in S256	No dependencies found	No information available	No information available	loop_site_365
loop at loops90.f:1287 in S257	RAW:1	No information available	No information available	loop_site_372
loop at loops90.f:1313 in S258	No dependencies found	No information available	No information available	loop_site_373
loop at loops90.f:2127 in S321	RAW:1	No information available	No information available	loop_site_509
loop at loops90.f:2150 in S322	RAW:1	No information available	No information available	loop_site_515
loop at loops90.f:2150 in S322	RAW:1	No information available	No information available	loop_site_514
loop at loops90.f:2173 in S323	RAW:1	No information available	No information available	loop_site_516
loop at loops90.f:2276 in S342	No dependencies found	No information available	No information available	loop_site_528
loop at loops90.f:2301 in S343	No dependencies found	No information available	No information available	loop_site_534
loop at loops90.f:2712 in S442	No information available	100% / 0% / 0%	All unit strides	loop_site_501
loop at loops90.f:2712 in s442_ompparallel_for@2710	No information available	100% / 0% / 0%	All unit strides	loop_site_102
loop at loops90.f:2840 in S471	No dependencies found	No information available	No information available	loop_site_611
loop at loops90.f:2840 in s471_ompparallel_for@2839	No dependencies found	No information available	No information available	loop_site_99
loop at loops90.f:369 in S123	No dependencies found	No information available	No information available	loop_site_183
loop at loops90.f:448 in S126	RAW:1	No information available	No information available	loop_site_196
loop at loops90.f:573 in s141_ompparallel_for@570	No dependencies found	No information available	No information available	loop_site_77
loop at loops90.f:884 in S221	RAW:1	No information available	No information available	loop_site_286
loop at loops90.f:908 in S222	RAW:1	No information available	No information available	loop_site_289
loop at loops90.f:959 in S232	RAW:1	No information available	No information available	loop_site_296
loop at loops90.f:959 in S232	RAW:1	No information available	No information available	loop_site_300
loop at loops90.f:959 in s232_ompparallel_for@956	No dependencies found	No information available	No information available	loop_site_88
loop at loops90.f:959 in s232_ompparallel_for@956	RAW:1	No information available	No information available	loop_site_711

Dependency analysis

```
1259      do nl= 1,ntimes/n                                   0.010s
1260        do j= 2,n                                         0.010s
1261      do i= 1,n                                           0.039s
          [Scalar loop at loops90.f:1261 in S256]
            Scalar Loop. Not vectorized: vector dependence prevents vec
            Loop was unrolled by 2
1262        a(j)= aa(i,j)-a(j-1)                              0.120s
1263        aa(i,j)= a(j)+bb(i,j)                             0.151s
1264          enddo
1265        enddo
```

Advisor dynamic analysis found no dependencies

```
1287      do i= 2,n                                           0.050s
1288        a(i)= aa(i,j)-a(i-1)
1289        aa(i,j)= a(i)+bb(i,j)                             0.224s
1290          enddo
1291        enddo
1292      call dummy(ld,n,a,b,c,d,e,aa,bb,cc,1.)
1293    enddo
1294      call forttime(t2)
1295      t2= t2-t1-ctime-(dtime*float(ntimes/n))
1296      chksum= cs1d(n,a)+cs2d(n,aa)                          0.020s
```

Advisor found a RAW dependency!

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**

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization: Loop Type	Compiler Vectorization: Why No Vectorization?
[loop in runCForallLambdaLoops]	0.094s	0.094s	Scalar	vector dependence prevents vector ...
[loop in runCForallLambdaLoops]	0.140s	3.744s	Scalar	inner loop was already vectorized
[loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop 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.
Tough problem #1 for already vectorized codes

Non-Contiguous Memory Access – Tough Problem #2

Potential to vectorize but may be inefficient

Unit-Stride access

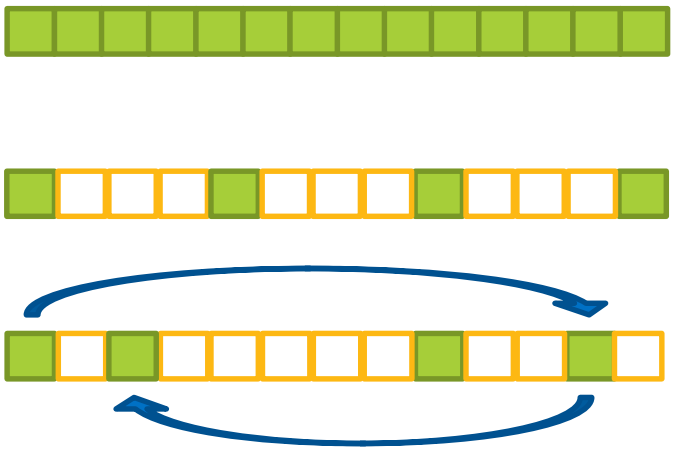
```
for (i=0; i<N; i++)
A[i] = C[i]*D[i]
```

Constant stride access

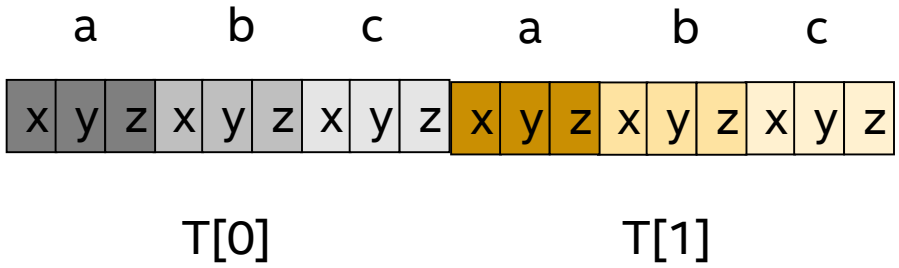
```
for (i=0; i<N; i++)
point[i].x = x[i]
```

Variable stride access

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



Object-oriented programming




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

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

Object oriented programming may inhibit SIMD code generation

Improve Vectorization

Memory Access pattern analysis

Where should I add vectorization and/or threading parallelism?

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

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

Function Call Sites and Loops					Loop Type	Why No Vectorization?
[loop at fractal.cpp:179 in <lambda1>::op ...		2 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	

Select loops of interest

2.2 Check Memory Access Patterns

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

Command Line

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

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization: Loop Type	Compiler Vectorization: Why No Vectorization?
[loop in runCForallLambdaLoops]	0.094s	0.094s	Scalar	vector dependence prevents vector ...
[loop in runCForallLambdaLoops]	0.140s	3.744s	Scalar	inner loop was already vectorized
[loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop 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

Memory Access Patterns | Correctness Report

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

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

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

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

Memory Access Patterns Report | Dependencies Report

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);
```

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
```

4.
It's time for explicit parallelism choices to make your code faster, not slower.

Example of Outer Loop Vectorization

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

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

x = 0	x = 1	x = 2	x = 3
z = z * c	z = z * c	z = z * c	z = z * c
z = z * c	z = z * c	z = z * c	z = z * c
	....	.....	.....
iters = 2	iters = 23	iters = 255	iters = 37

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

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

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

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization: Loop Type	Compiler Vectorization: Why No Vectorization?
[loop in runCForallLambdaLoops]	0.094s	0.094s	Scalar	vector dependence prevents vector ...
[loop in runCForallLambdaLoops]	0.140s	3.744s	Scalar	inner loop was already vectorized
[loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s	Vectorized (Body)	
Vectorized SSE; SSE2 loop processing Float32; Float64 data type(s) having Divisions; Square Roots operations Peeled loop; loop 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

Memory Access Patterns | Correctness Report

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

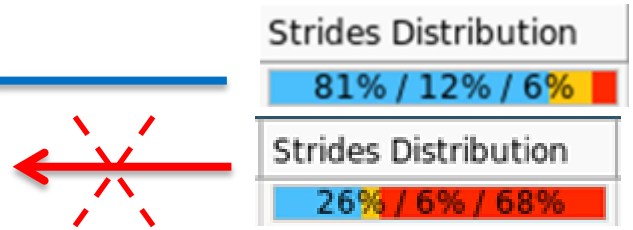
```
635    j2 = ( j2 & 64-1 ) ;
636    p[ip][0] += y[i2+32];
637    p[ip][1] += z[j2+32];
638    i2 += e[i2+32];
639    j2 += f[j2+32];
```

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

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

```
for(int i=0; i<Xmax; i++)
   for(int j=0; j<Ymax; j++)
      for(int k=0; k<Zmax; k++) {

         for (int l=0; l<qdim; l++) {
            for (int m=1; m<=half; m++) {
               //...
               fSwapPairM (...);
            }
         }
      }
```

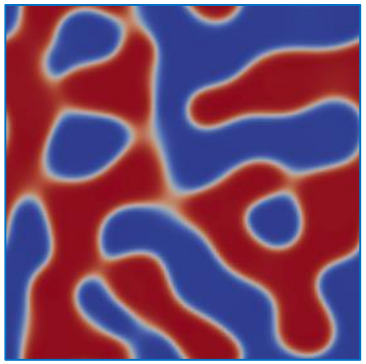


Memory Access Patterns analysis (+ Trip Counts and Dependencies) to drive decision wrt most appropriate parallelism level

Some use cases

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

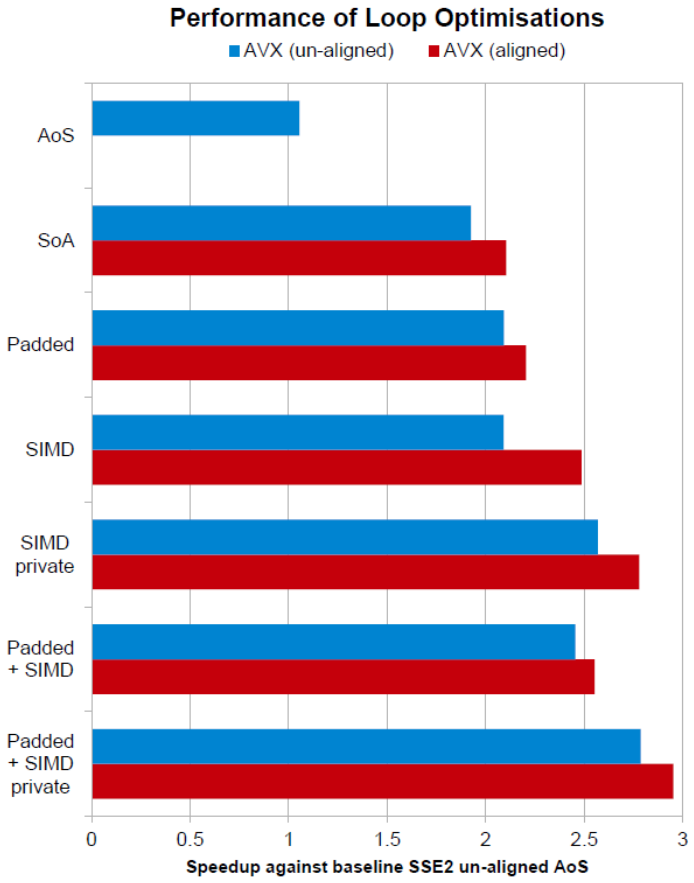
Workload characteristics:

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

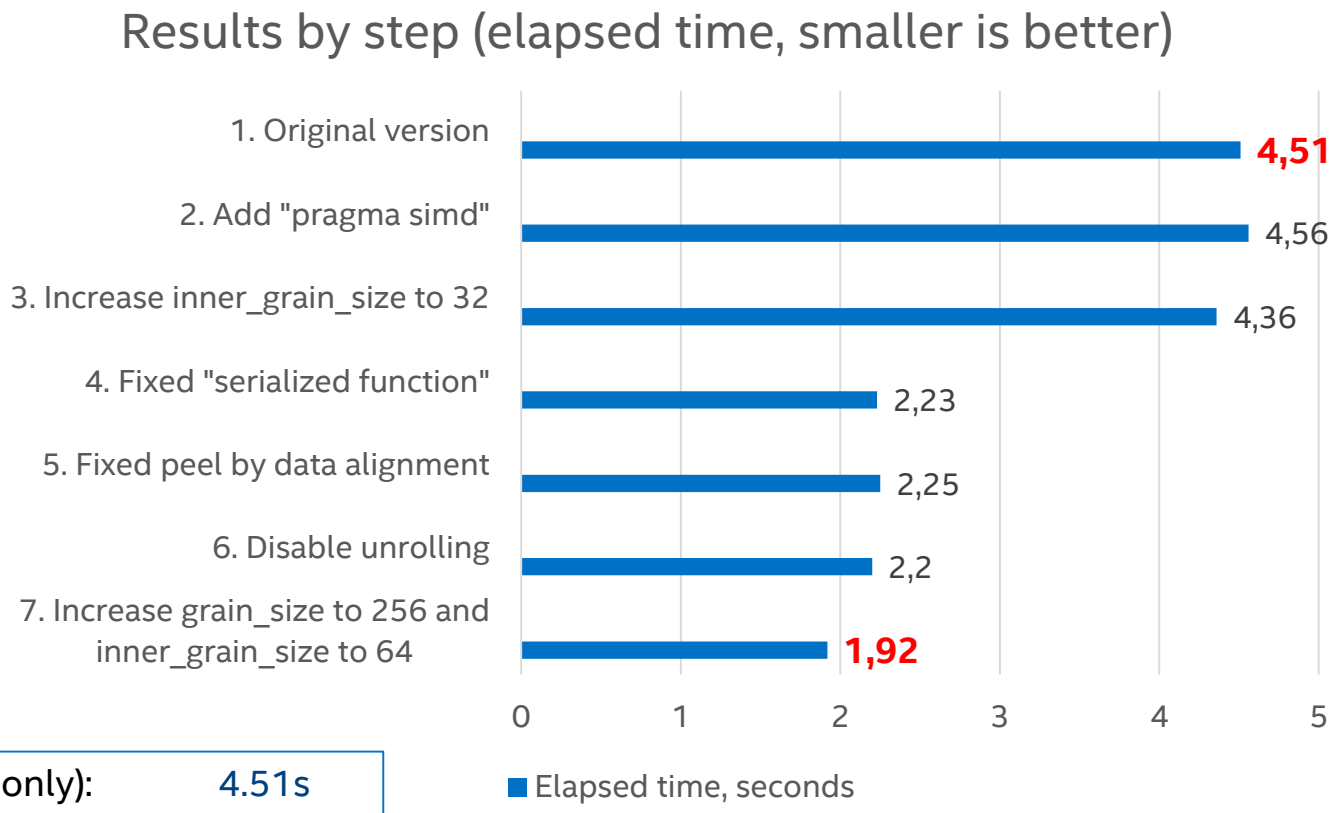
DL_MESO: one kernel results (ISC'15, Xeon Phi BoF, Hartree talk)

Performance - fGetEquilibriumF

- V-Advisor recommendations
  - AVX not enabled by default
  - MAP analysis points to AoS -> SoA.
  - Remove Scalar remainders.
  - Align data accesses.
- SoA allowed aligned access and removing peel loops.
- Array padding and #pragma loop count removes remainder loops.
- Additional optimization
  - #pragma SIMD
  - Private SIMD clause allowed additional compiler optimizations.
- Xeon speed up x2.95
- Phi speed up x4.05



TBB Fractal: more than 2x performance!



Original elapsed time (TBB only):	4.51s
Optimized time (TBB+vectorization):	1.92s
Speedup:	**2.3x**

"**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

Vector Advisor

C, C++, Fortran, C#

Windows, Linux

- 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: Loop Type	Why No Vectorization?	Gain Estimate	Vectorized Loops: 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...**

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

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, r
53          const real dtG = dt
54          for ( size_t i = 0;
55              real dvx = 0, dv
56      //#pragma vector always
57      for ( size_t j = 0; j <
        [loop at nbody.cc:57 i
            Scalar loop. Not v
            No loop transforma
        [loop at nbody.cc:57 i
            Vectorized AVX lo
            No loop transforma
58              if ( j != i
59                  real dx
60                  real dis
61                  real mOv
62                  dvx += m
63                  dvy += m
64                  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

Memory Access Patterns | Correctness Report

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	
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	

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

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

Effective usage of ISA (AVX, AVX2, AVX512)

Intel® AVX

Intel® Advanced Vector Extensions (Intel® AVX)

KEY FEATURES	*BENEFITS*
• **Wider Vectors** ▪ **Increased from 128 bit to 256 bit**	• Up to 2x peak FLOPs output with good power efficiency
• Enhanced Data Rearrangement – Use the new 256 bit primitives to broadcast, mask loads and permute data	• Organize, access and pull only necessary data more quickly and efficiently
• Flexible unaligned memory access support	• More opportunities to fuse load and compute operations

Intel® AVX is a general purpose architecture,

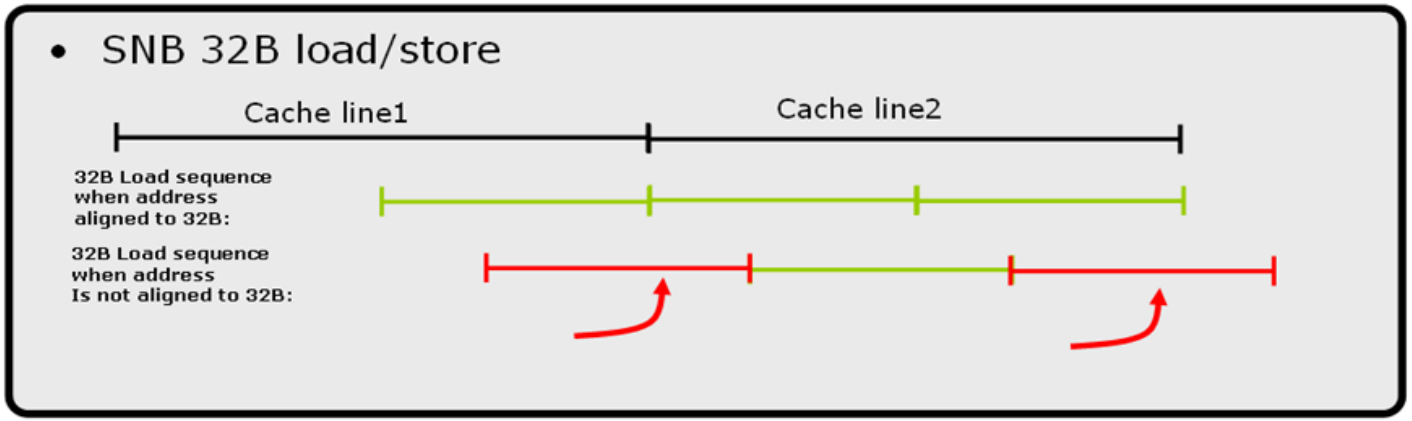
expected to supplant SSE in all applications used today

Intel® AVX Data Alignment Tuning Tips

Align Data to Vector Length

Intel® SSE - Align data to 16 Bytes (Intel® SSE vector length)

Intel AVX (Intel® microarchitecture (Sandy Bridge))- Align Data to 32 Bytes (minimize cache line split)



- Cache line length is 64 bytes
- Intel AVX register length is 32 bytes
- Sandy Bridge microarchitecture has 2 load ports, 1 store port
- 32 Bytes accesses are single micro-op using a "double pump"
- Unaligned data will cause every second load on consecutive memory accesses to be a cache line split

How to Align Data

- Allocate memory on heap aligned to n byte boundary:

```
void* _mm_malloc(int size, int n)
int posix_memalign(void **p, size_t n, size_t size)
[NEW Intel Compiler 15] #include <aligned_new>
```

- Alignment for variable declarations:

```
__attribute__((aligned(n)))  var_name
```

(C++11 `alignas` also OK)

And tell the compiler…

```
#pragma vector aligned
```

- Asks compiler to vectorize, overriding cost model, and assuming all array data accessed in loop are aligned for targeted processor
  - May cause fault if data are not aligned

```
__assume_aligned(array, n)
```

- Compiler may assume array is aligned to n byte boundary

```
typedef double* __attribute__((align_value(32))) A_DBL;
```

n=64 for Intel® Xeon Phi™ coprocessors, **n=32** for AVX, **n=16** for SSE

Applications likely to benefit from rebuilding for Intel® AVX

- Significant time spent in floating-point vectorizable loops with iteration count > vector length (8 floats, 4 doubles)
  - Vectorization with SSE is a good initial indication
- Calls to optimized performance libraries, e.g. MKL
  - Might not even need rebuilding

Less likely to benefit:

- Scalar or integer code;
- Heavy use of double precision division or square root
- Applications that are memory bound

Intel® AVX2

Intel® AVX2: Key Features

1. Extends 128-bit integer vector instructions to 256-bit
   - Including*: Intel® SSE2, Intel SSE3, SSSE3 and Intel SSE4 (some special instructions excepted)
2. Floating Point Fused Multiply Add: A*B + C
   - Increased FLOPS potential
   - Increased accuracy – Only a single rounding
3. Enhanced vectorization with Gather, Shifts and powerful permutes

Uses the same 256-bit YMM registers as Intel AVX

Intel AVX2 completes the 256-bit extensions started with Intel AVX: 256-bit integer , cross-lane permutes, gather, FMA

Other Features of Haswell (wrt AVX2)

Improved cache bandwidth to feed wide vector units and FMAs

- 32-byte load/store for L1 -> 2X bandwidth
- 2x L2 bandwidth

2 new ports:

- additional ALU and new branch unit
- new AGU (address generation unit) for stores;

New Gather Instructions

```
c[i] = a[b[i]]                    //indirect reference
```

```
if (p[i] == q[i])  c[i] = a[b[i]]   // also masked
```

p[i] q[i]

```
VPCMPEQQ       ymm3, ymm2, ymm1
VPGATHERQQ     ymm1, ptr [rax+ymm0], ymm3
```

Base of "a" b[i] mask

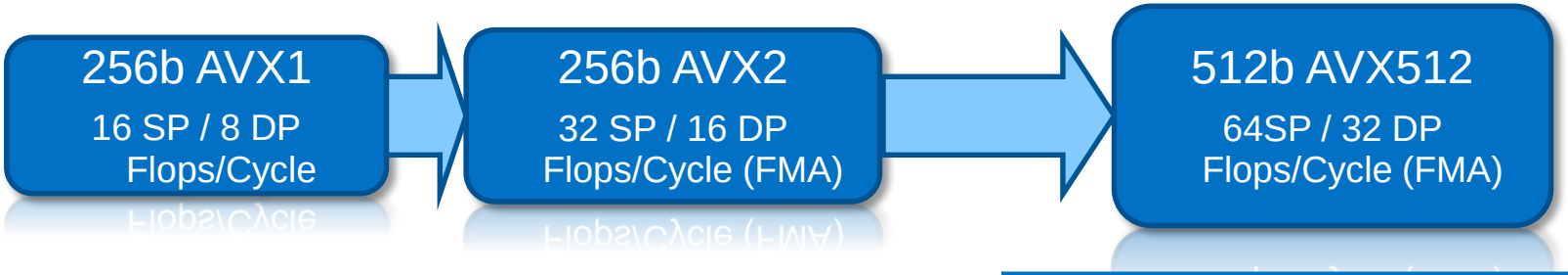
Fundamental building block for sparse or indirect memory accesses, easing vectorization

Applications Likely to Benefit from recompiling for Intel® AVX2

- CPU bound
- Significant time spent in vectorizable loops with
  - iteration count ≥ vector width (8 ints, 8 floats, 4 doubles) and
  - integer arithmetic & bit manipulation (e.g. video processing) or
  - floating-point that can make use of FMAs (e.g. linear algebra) or
  - non-contiguous memory access, if new gather & permute instructions make vectorization more efficient
- **Less likely to benefit:**
  - apps bound by memory bandwidth won't benefit directly from the instruction set
  - Iteration counts << vector width

Intel® AVX512

Intel® AVX Technology



AVX	AVX2
256-bit basic FP	Float16 (IVB 2012)
16 registers	256-bit FP FMA
NDS (and AVX128)	256-bit integer
Improved blend	PERMD
MASKMOV	Gather
Implicit unaligned	

SNB
2011

HSW
2013

AVX512
512-bit FP/Integer
32 registers
8 mask registers
Embedded rounding
Embedded broadcast
Scalar/SSE/AVX "promotions"
Transcendental support
Gather/Scatter

Future Processors (KNL & Future Xeon)

Math Support

Package to aid with Math library writing
- Good value upside in financial applications
- Available in PS, PD, SS and SD data types
- Great in combination with embedded RC

Instruction 30		
VGETXEXP$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Obtain exponent in FP format
VGETMANT$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Obtain normalized mantissa
VRNDSCALE$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2, imm8	Round to scaled integral number
VSCALEF$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2, zmm3	$X*2^{y}$, X <= getmant, Y <= getexp
VFIXUPIMM$_{\{PS,PD,SS,SD\}}$	zmm1, zmm2, zmm3, imm8	Patch output numbers based on inputs
VRCP14$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Approx. reciprocal() with rel. error 2^{-14}
VRSQRT14$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Approx. rsqrt() with rel. error 2^{-14}
VDIV$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2, zmm3	IEEE division
VSQRT$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	IEEE square root

Additional HPC AVX-512 superset for 2nd generation Intel Xeon Phi

CPUID	Instructions	Description
AVX512-PF	PREFETCHWT1	Prefetch cache line into the L2 cache with intent to write
	VGATHERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache
	VSCATTERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache with intent to write
AVX512-ER	VEXP2{PS,PD}	Computes approximation of 2^x with maximum relative error of 2^{-23}
	VRCP28{PS,PD}	Computes approximation of reciprocal with max relative error of 2^{-28} before rounding
	VRSQRT28{PS,PD}	Computes approximation of reciprocal square root with max relative error of 2^{-28} before rounding
AVX512-CD	VPCONFLICT{D,Q}	Detect duplicate values within a vector and create conflict-free subsets
	VPLZCNT{D,Q}	Count the number of leading zero bits in each element
	VPBROADCASTM{B2Q,W2D}	Broadcast vector mask into vector elements

Why True Masking?

Memory fault suppression

- Vectorize code without touching memory that the correspondent scalar code would not touch
  - Typical examples are if-conditional statements or loop remainders
  - AVX is forced to use VMASKMOV*

MXCSR flag updates and fault handlers

- Avoid spurious floating-point exceptions without having to inject neutral data

Zeroing/merging

- Use zeroing to avoid false dependencies in OOO architecture

```
float32 A[N], B[N], C[N];

for(i=0; i<16; i++)
{
    if(B[i] != 0) {
        A[i] = A[i] / B[i];
    else {
        A[i] = A[i] / C[i];
    }
}
```

```
VMOVUPS zmm2, A
VCMPPS k1, zmm0, B
VDIVPS  zmm1 {k1}{z}, zmm2, B
KNOT k2, k1
VDIVPS  zmm1 {k2}, zmm2, C
VMOVUPS A, zmm1
```

Vectorization Analysis for AVX-512 platforms

Running Vectorization Advisor "Survey" analysis on next generation Intel® Xeon Phi (KNL)

AVX-512 ERI – specific to Intel® Xeon Phi

Loops	Vector Issues	Self Time	Loop Type	Vectorized Loops: Vector ISA	Efficiency	Gain Esti...	VL (V...	Traits	Vector ...	Instruction Sets	Advanced: Vectorizat
[loop	3 Possible i...	35.226s 5.4%	Vectorized+Threaded (Body; Peeled; Re...	AVX512	~28	2.21x	8	Divisions; FMA; Gathers	Float32; ...	256/512	AVX; AVX2; AVX512ER_512; ...	Masked L
[loc	2 Possible in...	26.025s 4.0%	Vectorized (Body)+Threaded (OpenMP)	AVX512			8	Divisions; Gathers; FMA	Float32; ...	256/512	AVX; AVX512ER_512; AVX512F...	
[loc	1 High vecto...	5.876s	Vectorized (Peeled)+Threaded (OpenMP)	AVX512			8	Divisions; Gathers; FMA	Float32; ...	256/512	AVX2; AVX512ER_512; AVX512...	Masked Lo
[loc	1 High vecto...	3.324s	Vectorized (Remainder)+Threaded (Open...	AVX512			8	Divisions; Gathers; FMA	Float32; ...	256/512	AVX2; AVX512ER_512; AVX512...	Masked Lo
[loop		34.599s 5.3%	Vectorized (Body; Remainder)	AVX512	~70%	5.64x	8	Divisions; FMA				
[loop	1 Possible in...	33.849s 5.2%	Vectorized (Body; Peeled; Remainder)	AVX512	~28%	2.24x	8	Divisions; FMA				
[loop		19.839s 3.1%	Vectorized (Body; Remainder)	AVX512	~72%	11.48x	16; 8					

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

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

Issue: Possible inefficient memory access patterns present

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

Recommendation: Confirm inefficient memory access patterns — Confidence: Need More Data

There is no confirmation inefficient memory access patterns are present. To confirm: Run a Memory Access Patterns analysis.

Issue: Ineffective peeled/remainder loop(s) present

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

Recommendation: Collect trip counts data

The Survey Report lacks trip counts data that might generate more precise recommendations. To fix: Run a Trip Counts analysis.

Recommendation: Align data — Confidence: Low

Recommendation: Add data padding — Confidence: Low

The trip count is not a multiple of vector length. To fix: Do one of the following:

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

Windows* OS	Linux* O
/Qopt-assume-safe-padding	-qopt-assume-sa

Performance optimization problem and advice how to fix it

Program metrics

Elapsed Time: 142.79s

Vector Instruction Set: AVX, AVX2, AVX512, SSE, SSE2 — Number of CPU Threads: 4

Loop metrics

Total CPU time	454.08s	100.0%
Time in **88** vectorized loops	41.86s	9.2%

No access to AVX-512 Hardware yet? Explore AVX-512 code with Advisor!

Experimental feature. Contact **vector_advisor@intel.com**

1. **Use -axCOMMON-AVX512 -xAVX compilation flags. Compiler will generate two code-paths:**
   - AVX-512 code path (not executed on your Xeon/Core machine)
   - AVX(2) code path (executed when run on your Haswell Xeon or earlier)
2. **Use special mode of Advisor analysis:**
   - Compare AVX and AVX-512 code characteristics on Xeon!

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

AVX2 variant: executed (on Haswell), 0.47 seconds

Speed-up estimate: 31.6x (AVX-512) vs. 13.5x (AVX2)

AVX512 variant: not executed (on Haswell)

Loops	Self Time	Loop Type	Vectorized Loops: Vect...	Efficiency	Gain...	VL (...	Compiler Es...	Instruction Set Analysis: Traits	D...	W...	Instruction Sets	Advanced: Vectorization De
[loop in s352_ at loopstl.cpp:5939]	0,641s	Vectorized (Body)	AVX2	~54%	2,15x	4	2,15x	FMA; Inserts		128	AVX; FMA	
[loop in s352_ at loopstl.cpp:5939]	n/a	Remainder [Not Executed]				4		FMA	Float32			
[loop in s352_ at loopstl.cpp:5939]	0,641s	Vectorized (Body)	AVX2			4	2,15x	Inserts; FMA	Float32	128	AVX; FMA	
[loop in		d]	AVX512			16	3,20x	Gathers; FMA				
[loop in		ecuted]	AVX512			16	2,70x	Gathers; FMA				
[loop in s1			AVX2	~100%	13,54x	8	<13,54x	FMA; NT-stores				
[loop in						8		FMA				
[loop in s125_.Aompparallel_for...	n/a	Remainder [Not Executed]				8		FMA	Float32			
[loop in s125_.Aompparallel_for...	0,465s	Vectorized (Body)	AVX2			8	13,54x	NT-stores; FMA		256	AVX; FMA	Unaligned Acce
[loop in s125_.Zompparallel_for...	n/a	Vectorized (Peeled) [Not Executed]	AVX512			16	6,77x	FMA	Float32...	512	AVX512F_512	Masked Loop Ve
[loop in s125_.Zompparallel_for...	n/a	Vectorized (Body) [Not Executed]	AVX512			32	30,61x	NT-stores; FMA	Float32	512	AVX512F_512	Unaligned Acce
[loop in s125_.Zompparallel_for...	n/a	Vectorized (Remainder) [Not Executed]	AVX512			16	9,78x	FMA	Float32...	512	AVX512F_512	Masked Loop Ve

Source | Top... | ...ostic Details

File: loopstl.cp...

Line	Source	Total Time	%	Loop Time	%	Traits
5939	for (i__ = 1; i__ <= i__2; i__ += 5)	0,093s		0,641s		
	[loop in s352_ at loopstl.cpp:5939] Vectorized AVX; FMA loop processes Float32 data type(s) and includes Inserts; FMA Loop was unrolled by 1					
	[loop in s352_ at loopstl.cpp:5939] Vectorized AVX512F_512 loop [not executed] processes Float32 data type(s) and includes Gathers; FMA Loop was unrolled by 1					
	[loop in s352_ at loopstl.cpp:5939] Vectorized AVX512F_512 remainder loop [not executed] processes Float32; Int32; UInt32 data type(s) and inclu... No loop transformations applied					
	[loop in s352_ at loopstl.cpp:5939] Scalar remainder loop [not executed]. Not vectorized Loop was unrolled by 1					
5940	dot += a[i__] * b[i__] + a[i__ + 1] * b[i__ + 1] + a[i__ + 2]	0,563s				FMA; Gath...
5941	* b[i__ + 2] + a[i__ + 3] * b[i__ + 3] + a[i__ + 4] * b[i__ + 4];					
5942	dummy_ (ld, n, &a[1], &b[1], &c__[1], &d__[1], &e[1], &aa[aa_offset],	0,016s				
5943	&bb[bb_offset], &cc[cc_offset], &c_b3);					
5944	}					

1 loop in source code:

4 loops in binary:

- 2 AVX2 versions and
- 2 AVX-512 versions

Threading Advisor XE

Data-Driven Threading Design

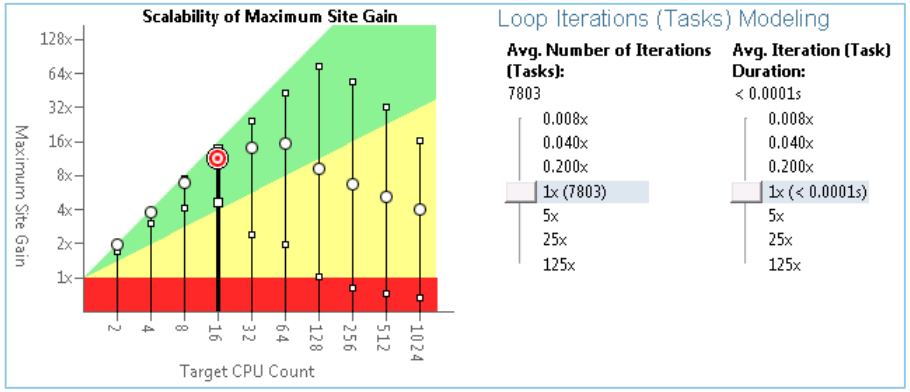
Intel® Advisor XE – Thread Prototyping

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



Add Parallelism with Less Effort, Less Risk and More Impact

http://intel.ly/advisor-xe

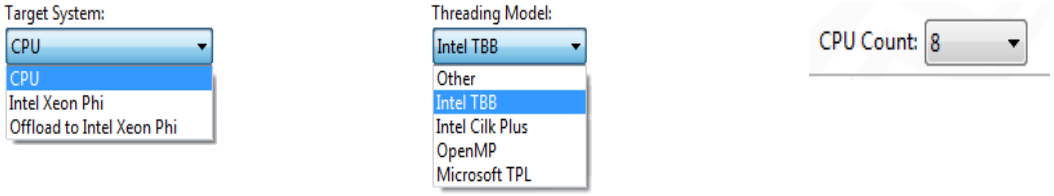
Check Suitability

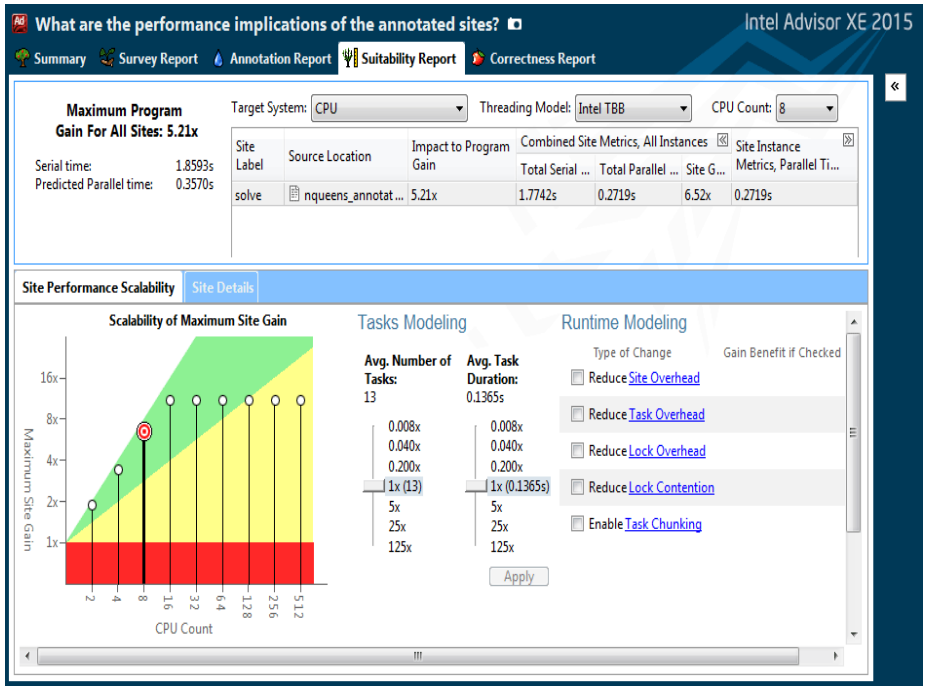
Is it fast enough?

Experiment with modeling by changing:

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

Instantly see impact on scalability





Quickly Evaluate Design Alternatives

Summary

Back-up

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

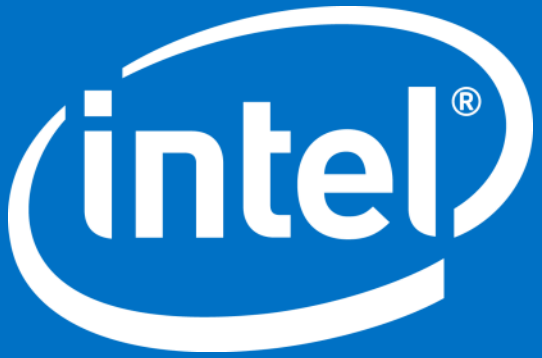
Beta Registration & Download:

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

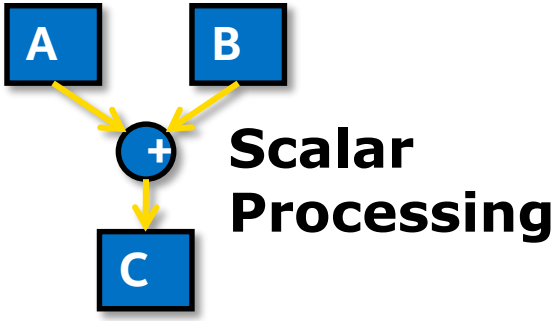
For Intel® Xeon Phi™ coprocessors, but also applicable:

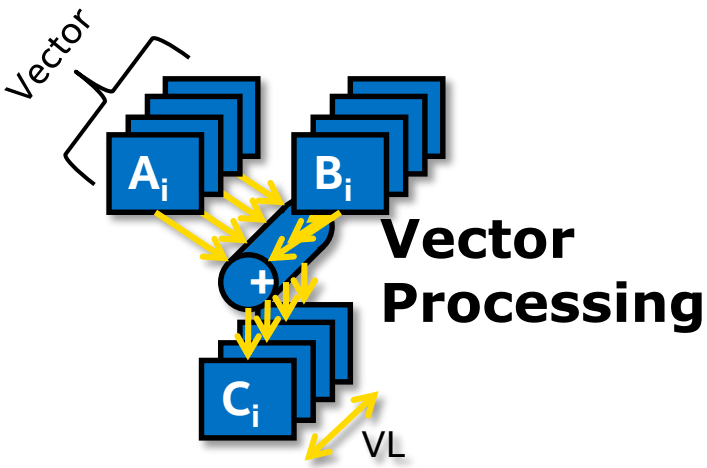
https://software.intel.com/en-us/articles/vectorization-essential

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


intel®

Recap





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

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