



INTEL ADVISOR

Intel Software and Services, May' 2016

Zakhar Matveev, PhD, Product Architect



INTEL (VECTORIZATION “AND MEMORY”) ADVISOR

Intel Software and Services, May' 2016

Zakhar Matveev, PhD, Product Architect

Agenda [Training Part]

- Vectorization with Intel Advisor
 - Vectorization bottlenecks and methodology overview
 - Characterize and improve your SIMD code with help of Advisor
 - Analyze Memory Bound codes
- Beta capabilities
 - AVX-512 specific Advisor analysis capabilities
 - FLOPs and Masks Profiler
- Roofline analysis

EFFECTIVE VECTORIZATION WITH INTEL ADVISOR 2016

- Vectorization Bottlenecks overview
- Vector Advisor High Level methodology and workflow overview

Factors that prevent Vectorizing your code

1. Loop-carried dependencies

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

1.A Pointer aliasing (compiler-specific)

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

2. Function calls (incl. indirect)

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

3. Loop structure, boundary condition

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

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

4 Outer vs. inner loops

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

5. Cost-benefit (compiler specific..)

And others.....

Factors that **slow-down** your **Vectorized** code

1.A. Indirect memory access

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

1.B Memory sub-system Latency / Throughput

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

2. Serialized or “sub-optimal” function calls

```
for (i = 1; i < nx; i++) {  
    sumx = sumx +  
        serialized_func_call(x,  
y, xp);  
}
```

3. Small trip counts not multiple of VL

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

4. Branchy codes, *outer vs. inner loops*

```
for(i = 0; i <= MAX; i++) {  
    if ( D[i] < N)  
        do_this(D);  
    else if (D[i] > M)  
        do_that();  
    //...  
}
```

5. **MANY** others: spill/fill, fp accuracy trade-offs, FMA, DIV/SQRT, Unrolling, even AVX throttling..

Get Fast Code Fast! Intel® Advisor

Vectorization Optimization and Thread Prototyping

Software Must Vectorize & Thread or Performance Dies (Underutilizes the Processor)

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

Vectorization - Have you:

- Recompiled for **AVX2/AVX-512** with little **gain**
- Struggled with **compiler** reports?
- Wondered where to vectorize?

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

Dr. Luigi Lapichino
Scientific Computing Expert
Leibniz Supercomputing Centre

Threading - Have you:

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

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

Simon Hammond
Senior Technical Staff
Sandia National Laboratories

5 Steps to Efficient Vectorization - Vector Advisor

(part of Intel® Advisor, Parallel Studio, Cluster Studio 2016)

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization
			Loop Type Why No Vectorization?
[loop in runCforallLambdaLoops]	0.094s	0.094s	Scalar vector dependence prevents vector...
[loop in runCforallLambdaLoops]	0.140s	3.744s	Scalar inner loop was already vectorized
[loop in std::Complex_base<double,struct _C_double_complex>::...]	0.031s	0.031s	Vectorized (Both)

Vectorized SSE: SSE2 loop processing Float32; Float64 data type
Peeled loop: loop stats were reordered

Function Call Sites and Loops	Self Time	Total Time
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allocat...	0.000s	0.000s
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allocat...	0.000s	0.000s
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	0.000s

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

Issue: Peeled/Remainder loop(s) present

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

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

Use one of the memory accesses in the source loop does not align on a byte boundary and tell the compiler your memory access is aligned to a byte boundary.

```
SIZE*sizeof(float), 32);
```

3. "Accurate" Trip Counts + FLOPs: understand utilization, parallelism granularity & overheads

Total Time	Trip Counts			Iteration Duration	Call Count
	Median	Min	Max		
3.151s	1	1	1	3,150ns	1
0.440s	1	1	1	< 0.0001s	2408000
0.010s	1	1	2	< 0.0001s	207596
0.010s	1	2	1	< 0.0001s	1173619
0.010s	1	3	1	< 0.0001s	1312315

4. Loop-Carried Dependency Analysis

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	runRawLoops	runRawLoops.cxx:1063	RAW:1	No information available	No information available
loop_site_139	runRawLoops	runRawLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runRawLoops	runRawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

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

```

635 j2 = ( j2 + 64 - 1 ) ;
636 p[ip][0] += z[j2+32];
637 p[ip][1] += z[j2+32];
638 i2 += e[12+32];
639 j2 += f[j2+32];
    
```

```

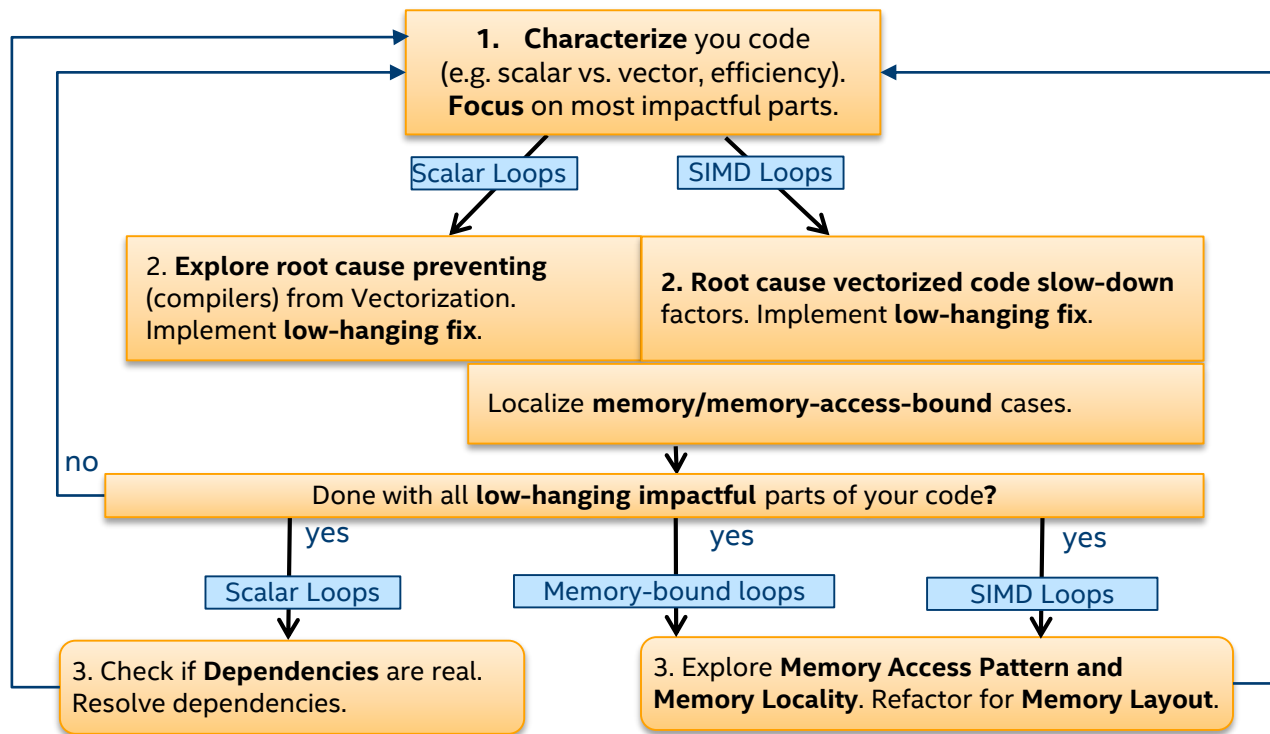
626 i1 = 64-1;
627 j1 = 64-1;
628 p[ip][2] += b[j1][11];
    
```

Optimization Notice

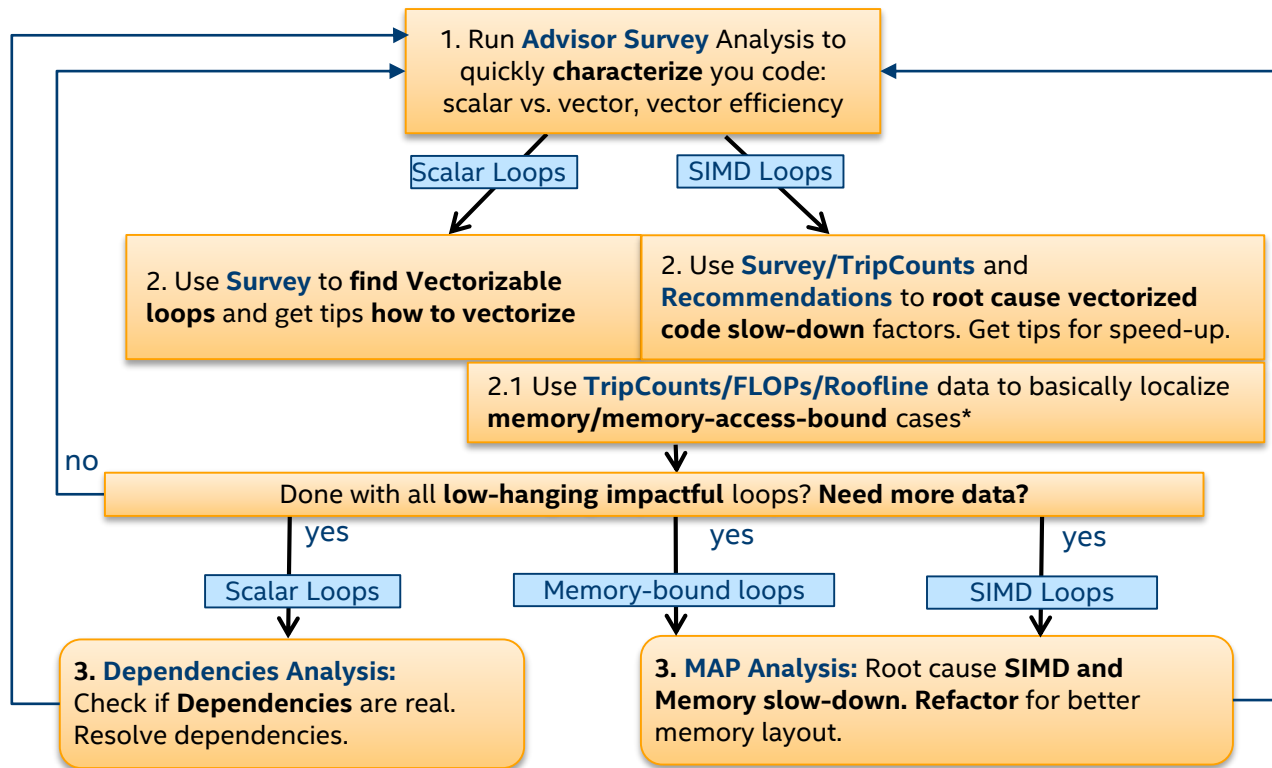
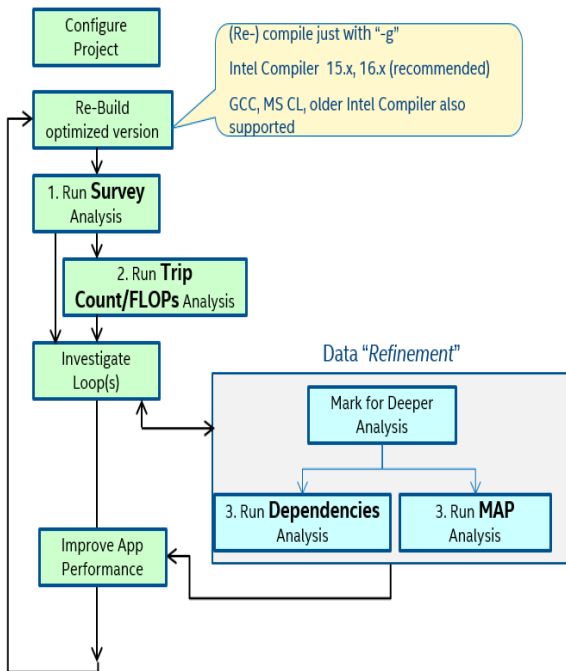
Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



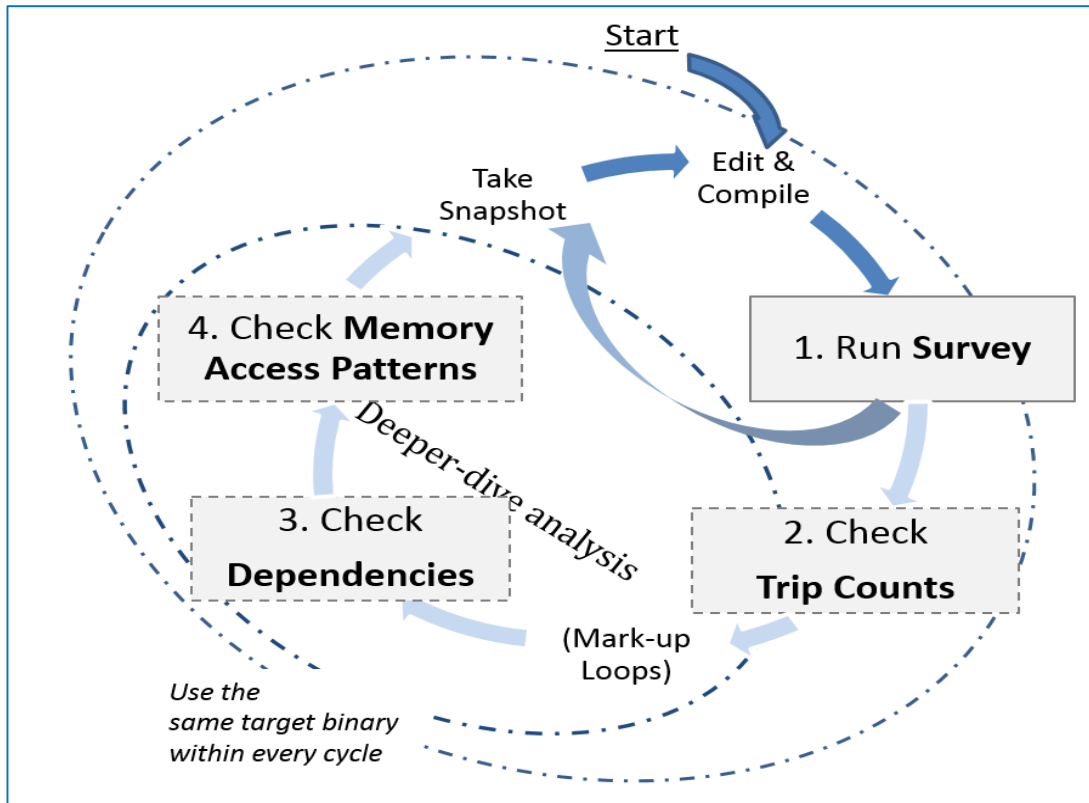
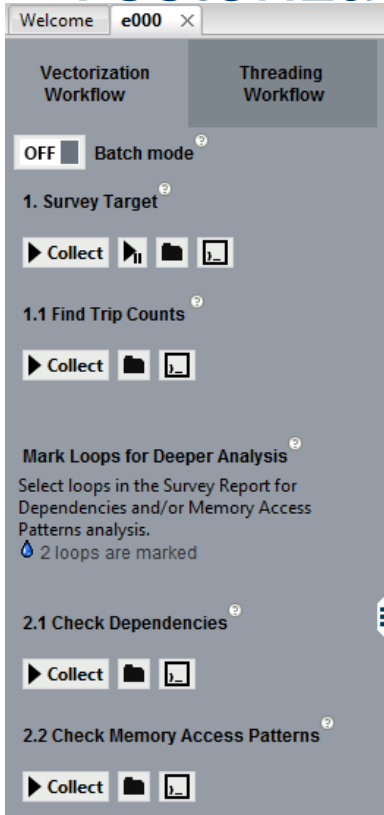
Using Advisor and SIMD methodology



Using Advisor to address these factors



Vectorization Analysis Workflow



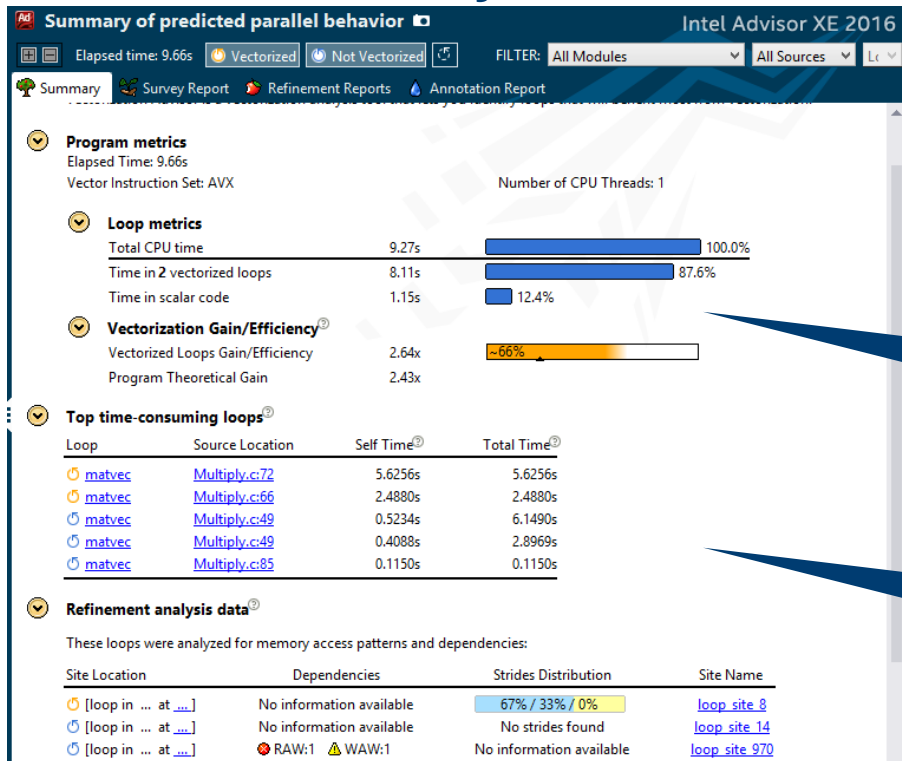
Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

CHARACTERIZE YOUR CODE, FOCUS ON MOST IMPACTFUL CODE.

- Using Advisor Summary and Survey

Quickly **characterize** the efficiency of your code: Advisor Summary.



Use the summary view to quickly characterize your program

Time in **Scalar vs. Vector** loops. SIMD Efficiency.

Focus on **Hottest** kernels

Advisor Survey: Focus + Characterize.

One stop shop.

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

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

Summary Survey Report Refinement Reports Annotation Report

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops			
						Vect...	Efficiency	Gain...	VL (...)
[loop in matvec at Multiply.c:72]	1 Inefficient ...	5.281s	5.281s	Vectorized (Bo...		AVX	~55%	2.19x	4
[loop in matvec at Multiply.c:66]	1 Ineffective ...	2.828s	2.828s	Vectorized (Bo...		AVX	~91%	3.65x	4
[loop in matvec at Multiply.c:49]		0.531s	5.812s	Scalar					
[loop in matvec at Multiply.c:49]		0.516s	3.344s	Scalar					
[loop in matvec at Multiply.c:85]	1 Assumed d...	0.063s	0.063s	Scalar	vector dependence...				
[loop in main at driver.c:151]		0.016s	9.297s	Scalar	loop in function ...				
[loop in matvec at Multiply.c:49]			0.000s	Scalar					

hotspots

HOW to improve performance

ISA

Efficiency

What prevents vectorization?

All the data you need for effective vectorization

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Advisor Survey: Focus + Characterize.

HPC (medium size) workload example:

- 1) **1000** (user code) loops potentially executable (including multi-versioning, etc)
- 2) Reports = **30 000 lines of text**
- 3) Only **250** loops were really executed
- 4) Only **150** loops took >0.1% of time
- 5) Only **50** loops were **hotspots** (make noticeable difference in performance)

Advisor helps to immediately focus on important 50 lines (instead of original 30 000), still preserving compiler diagnostics.

- Further narrow data down using filtering (e.g by source, by scalar vs. vector) and sorting dynamic+static data.

Advisor Survey: Focus + Characterize.

Focus and order hottest loops

Function Call Sites and Loops	Self Time	Type	Vectorized Loops	
			Vector ...	Efficiency
[loop in fPropagation ...]	1,078s	Inside vectorized		
[loop in fGetEquilibr...	0,906s	Vectorized (Body)	AVX2	~55%
[loop in fGetSpeedSite ...]	0,890s	Scalar		
[loop in fPropagationS ...]	0,438s	Scalar		
[loop in fSiteFluidCollis ...]	0,423s	Scalar		
[loop in I10_OUTPUT a ...]	0,125s	Scalar		
[loop in fSiteFluidCollis ...]	0,094s	Scalar		
[loop in fGetOneMassS ...]	0,078s	Vectorized (Remainder)	AVX	~33%
[loop in fCollisionBGK ...]	0,062s	Vectorized (Remainder)	AVX2	~65%
[loop in fPropagationS ...]	0,047s	Scalar		
[loop in write_string at ...]	0,031s	Scalar		
[loop in fGetSpeedSite ...]	0,031s	Scalar		
[loop in fGetFracSite at ...]	0,031s	Vectorized (Body)	AVX2	~24%
[loop in I10_OUTPUT a ...]	0,016s	Scalar		
[loop in cropzeros_I at ...]	0,016s	Scalar		
[loop in cropzeros_I at ...]	0,016s	Scalar		
[loop in I10_OUTPUT a ...]	0,016s	Scalar		
[loop in strcpy_s at tcs ...]	0,016s	Scalar		
[loop in write_nolock a ...]	0,016s	Scalar		
[loop in output_s_I at o ...]	0,016s	Scalar		
[loop in _intel_sse4_str ...]	0,016s	Scalar		
[loop in fCollisionBGK ...]	0,016s	Scalar		
[loop in fsBGK at lbpR ...]	0,000s	Scalar		
[loop in std::basic_ofstr ...]	0,000s	Scalar		
[loop in std::basic_ofstr ...]	0,000s	Scalar		
[loop in std::basic_ofstr ...]	0,000s	Scalar		
[loop in std::num_put< ...]	0,000s	Scalar		
[loop in write_nolock a ...]	0,000s	Scalar		
[loop in std::basic_ofstr ...]	0,000s	Scalar		
[loop in std::basic_ofstr ...]	0,000s	Scalar		
[loop in std::basic_ofstr ...]	0,000s	Scalar		
[loop in std::basic_ofstr ...]	0,000s	Scalar		
[loop in std::basic_ofstr ...]	0,000s	Scalar		

- Maximize your optimization ROI

Function Call Sites and Loops	Self Time	Type	Vectorized Loops	
			Vector ...	Efficiency
[loop in fPropagation ...]	1,078s	Inside vectorized		
[loop in fGetEquilibr...	0,906s	Vectorized (Body)	AVX2	~55%
[loop in fGetSpeedSite ...]	0,890s	Scalar		
[loop in fPropagationS ...]	0,438s	Scalar		
[loop in fSiteFluidCollis ...]	0,423s	Scalar		

Optimization Notice

Advisor Survey: Focus + Characterize.

Focus and order non-vectorized loops



- “Compiler diagnostics” sub-tab to explain root cause

Function Call Sites and Loops	Self Time	Why No Vectorization?▼
[loop in CS2D at mains.F:685]	0,100s	precise FP model implied by the command line o ...
[loop in CS1D at mains.F:668]	0,030s	precise FP model implied by the command line o ...
[loop in s118_at_loopstl.cpp:569]	0,030s	outer loop was not auto-vectorized: consider usin ...
[loop in s114_Somp\$parallel_for@389 ...]	0,010s	outer loop was not auto-vectorized: consider usin ...
[loop in s114_Somp\$parallel_for@389 ...]	0,000s	outer loop was not auto-vectorized: consider usin ...
[loop in s481_at_loopstl.cpp:7189]	0,344s	loop with multiple exits cannot be vectorized unle ...
[loop in s482_at_loopstl.cpp:7251]	0,300s	loop with multiple exits cannot be vectorized unle ...
[loop in s392_at_loopstl.cpp:5625]	0,140s	loop with multiple exits cannot be vectorized unle ...
[loop in s141_Somp\$parallel_for@1 ...]	0,000s	loop control variable was not identified. Explici ...

Source | Top Down | Loop Analytics | Loop Assembly | Recommendation | **Compiler Diagnostic Details**

```
for (int i = 0; i < dim.X; i++) {
```

Recommendations

- For **Example 1**, where the loop iteration count is not available before the loop execution:
 - Move the calculation outside the loop using an additional variable.
 - Rewrite the loop to avoid `goto` statements or other early exits from the loop.
 - Identify the loop iterations lower bound using a constant.

For example, introduce the new `limit` variable:

```
void foo(float *A) {
    int i;
    int OuterCount = 90;
    int limit = bar(int(A[0]));
    while (OuterCount > 0) {
        for (i = 1; i < limit; i++) {
            A[i] = i + 4;
        }
        OuterCount--;
    }
}
```

Do not use global variables or indirect accesses as loop boundaries unless you also

- Directive to ignore vector dependencies
- | Target | ICL/ICC/ICPC Directive |
|-------------|----------------------------|
| Source loop | <code>#pragma ivdep</code> |
- `restrict` keyword.

Read More:

- C++ information at <https://software.intel.com/en-us/articles/cdiag15523>

Advisor Survey: Focus + Characterize.

Focus and order vectorized loops

Function Call Sites and Loops	Vector Issues	Vectorized Loops			Instruction Set Analysis		
		Vect...	Efficiency	Gain...	VL ..	Traits	Data T.
[loop in s241_ at lo ...]		AVX	~97%	7,76x	8		Float32
[loop in s152s_ at lo ...]		AVX2	~96%	7,71x	8	FMA	Float32
[loop in s452_ at lo ...]	1 Data type conversions present	AVX2	~96%	7,71x	8	FMA; Type Con...	Float32
[loop in s413_ at lo ...]	1 Ineffective peeled/remainder ...	AVX2	~96%	7,69x	4; 8	FMA	Float32
[loop in s273_ at lo ...]	1 Possible inefficient memory a ...	AVX2	~96%	7,69x	8	FMA; Masked St...	Float32
[loop in s279_ at lo ...]	3 Possible inefficient memory a ...	AVX2	~95%	7,56x	8	Blends; FMA	Float32
[loop in s253_ at lo ...]	2 Possible inefficient memory a ...	AVX2	~91%	7,30x	8	Blends; FMA	Float32
[loop in s251_ at lo ...]		AVX2	~90%	7,23x	8	FMA	Float32
[loop in s271_ at lo ...]	2 Possible inefficient memory a ...	AVX2	~90%	7,16x	4; 8	FMA; Masked St...	Float32
[loop in vif_ at loop ...]	1 Possible inefficient memory a ...	AVX	~86%	6,90x	8	Blends	Float32
[loop in s274_ at lo ...]	1 Possible inefficient memory a ...	AVX2	~79%	6,29x	8	Blends; FMA; M...	Float32
[loop in SET2D at m ...]		AVX	~73%	5,81x	8		Float32
[loop in std::Fill<fl ...]		AVX	~73%	5,81x	8		Float32
[loop in SET2D at m ...]	1 Data type conversions present	AVX2	~66%	5,31x	8	Divisions; Type ...	Float32



- **Efficiency** – my performance thermometer
- **Recommendations** – get tip on how to improve performance
 - (also apply to scalar loops)

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

Issue: Assumed dependency present

Issue: Ineffective peeled/remainder loop(s) present

All or some [source loop](#) iterations are not executing in the [loop body](#). Improve performance by moving sou

Recommendation: Add data padding

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

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

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

Note: These compiler options apply only to Intel® Many Integrated Core Architecture (Intel® MIC Archi

When you use one of these compiler options, the compiler does not add any padding for static and aut application. To satisfy this assumption, you must increase the size of static and automatic objects in y

Optional: Specify the trip count, if it is not constant, using a [directive](#): `#pragma loop_count`

Read More:

- [qopt-assume-safe-padding](#), [Qopt-assume-safe-padding](#); [loop_count](#)

Optimization Notice

Vector Efficiency: my “performance thermometer”

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

~91%

~91%: Achieved Vectorization Efficiency

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

Estimated Gain = 3.65x

Vector Length = 4

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

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

(25%): Reference Efficiency for original scalar loop

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

(100%): Theoretical Maximum Vectorization Efficiency

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

Theoretical Maximum Gain = Currently selected Vector Length = 4

Intel microprocessors. Performance tests, such as SYSmark and MobileMark, change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to <http://www.intel.com/performance>. See FTC disclaimer.

Optimization Notice

ENABLE VECTORIZATION OF YOUR SCALAR CODE

- Compiler vectorization “Low-hanging fruits”
- Advisor Dependencies: check if Dependencies are real

Enable vectorization of your scalar code: Survey

- “Compiler diagnostics” to explain root cause

Some root causes are easy to overcome

- Loop boundary conditions
- Change compilation flags (fp model, -O2 -> -O3, etc)

Function Call Sites and Loops	Self Time	Why No Vectorization?▼
[loop in CS2D at mains.F:685]	0,100s	precise FP model implied by the command line o...
[loop in CSTD at mains.F:668]	0,030s	precise FP model implied by the command line o...
[loop in s118_ at loopstl.cpp:569]	0,030s	outer loop was not auto-vectorized: consider usin...
[loop in s114_ \$omp\$parallel_for@389 ...]	0,010s	outer loop was not auto-vectorized: consider usin...
[loop in s114_ \$omp\$parallel_for@389 ...]	0,000s	outer loop was not auto-vectorized: consider usin...
[loop in s481_ at loopstl.cpp:7189]	0,344s	loop with multiple exits cannot be vectorized un...
[loop in s482_ at loopstl.cpp:7251]	0,300s	loop with multiple exits cannot be vectorized un...
[loop in s332_ at loopstl.cpp:5625]	0,140s	loop with multiple exits cannot be vectorized un...
[loop in s141_ \$omp\$parallel_for@1 ...]	0,000s	loop control variable was not identified. Explici...

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

for (int i = 0; i < dim.x; i++) {

Recommendations

- For **Example 1**, where the loop iteration count is not available before the loop execution:
 - Move the calculation outside the loop using an additional variable.
 - Rewrite the loop to avoid `goto` statements or other early exits from the loop.
 - Identify the loop iterations lower bound using a constant.

For example, introduce the new `limit` variable:

```
void foo(float *A) {
    int i;
    int OuterCount = 90;
    int limit = bar(int(A[0]));
    while (OuterCount > 0) {
        for (i = 1; i < limit; i++) {
            A[i] = i + 4;
        }
        OuterCount--;
    }
}
```

- Do not use global variables or indirect accesses as loop boundaries unless you also:
 - Directive to ignore vector dependencies

Target	ICL/ICC/ICPC Directive
Source loop	#pragma ivdep

- `restrict` keyword.

Read More:

- C++ information at <https://software.intel.com/en-us/articles/cdiag15523>

Data Dependencies – Tough Problem #1

Is it safe to force the compiler to vectorize?

or

```
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.

① Enable vectorization

Potential performance gain: Information not available until Beta Update release

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

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

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

Read More:

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

Enable vectorization of your scalar code: Assumed Dependency

- More data is needed to confirm if the loop can be vectorized
- Select the given loop and run dependency analysis to see if it can be vectorized using an OpenMP* 4.0 simd pragma.

Loops		Vector Issues	Self Time▼	Total Time	Loop Type	Why No Vectorization?
[loop in fCalcInteraction_ShanChen at lbpFOR...	<input type="checkbox"/>	1 Ineffective peeled/remainder loop(..	0.894s	0.894s	Peeled/Remain...	
[loop in fGetSpeedSite at lbpGET.cpp:321]	<input type="checkbox"/>	2 Ineffective peeled/remainder loop(..	0.796s	0.796s	Vectorized (Bo...	
[loop in fPropagationSwap at lbpSUB.cpp:13 ...	<input type="checkbox"/>	2 Assumed dependency present	0.673s	1.988s 0	Scalar	vector dependence preve...

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.

Recommendation: Confirm dependency is real

Confidence: Need More Data

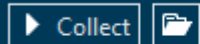
There is no confirmation that a real dependency is present in the loop. To confirm: Run a Dependencies analysis.

Check if it is safe to vectorize : Advisor Dependencies

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 Dependencies

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



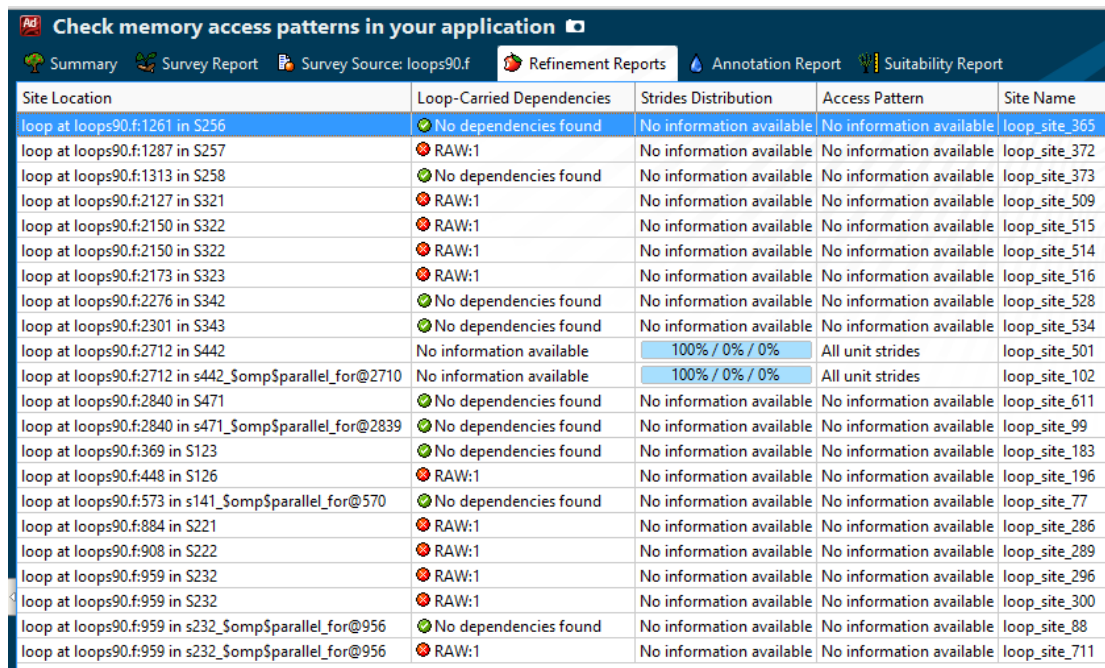
Command Line

Select loop for
Dependency
Analysis and
press play!

Vector Dependence
prevents
Vectorization!

Almost half loops checked did not have actual dependencies

We should investigate using `pragma simd` to force vectorization



The screenshot shows a software interface titled "Check memory access patterns in your application". It has several tabs: Summary, Survey Report, Survey Source: loops90.f, Refinement Reports, Annotation Report, and Suitability Report. Below the tabs is a table with the following columns: Site Location, Loop-Carried Dependencies, Strides Distribution, Access Pattern, and Site Name. The table lists 25 loop sites. The first 12 sites have "No dependencies found" or "RAW:1" dependencies and "No information available" for strides and access patterns. The 13th and 14th sites have "100% / 0% / 0%" for strides and "All unit strides" for access patterns. The remaining 10 sites have "No dependencies found" or "RAW:1" dependencies and "No information available" for strides and access patterns.

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_\$omp\$parallel_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_\$omp\$parallel_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_\$omp\$parallel_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_\$omp\$parallel_for@956	✔ No dependencies found	No information available	No information available	loop_site_88
loop at loops90.f:959 in s232_\$omp\$parallel_for@956	✘ RAW:1	No information available	No information available	loop_site_711

Optimization Notice

Data Dependencies – Tough Problem #1

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

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

Static Assumption:

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

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

Static Assumption:



```
i> [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 will ***know*** if indices overlap.

```
1) fSwapPairM ( lbf[il*lbsitlength + 1*lsy.nq + m + half],  
               lbf[ilnext*lbsitlength + 1*lsy.nq + m]);
```

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

```
2) fSwapPairM ( lbf[il*lbsitlength + 1*lsy.nq + m + half],  
               lbf[il*lbsitlength + 1*lsy.nq + m]);
```

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

 Read after write dependency

Dependencies Analysis: confirm wdependencies are **REAL**

SPEED-UP YOUR VECTOR CODE:

- Vectorization Efficiency metrics
- Root cause origins of slow-down
- Performance “Low-hanging fruits”

SPEED-UP YOUR VECTOR CODE:

- Vectorization Efficiency metrics
- Root cause origins of slow-down
- Performance “Low-hanging fruits”

Vector Efficiency: my “performance thermometer”

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

~91%

~91%: Achieved Vectorization Efficiency

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

Estimated Gain = 3.65x

Vector Length = 4

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

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

(25%): Reference Efficiency for original scalar loop

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

(100%): Theoretical Maximum Vectorization Efficiency

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

Theoretical Maximum Gain = Currently selected Vector Length = 4

Intel microprocessors. Performance tests, such as SYSmark and MobileMark, change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to <http://www.intel.com/performance>. See FTC disclaimer.

Optimization Notice

Vectorization Efficiency Metrics

- Ultimate Metric is “Speedup” or “Improvement to Productivity”
$$S = \text{Time (Scalar Exec)} / \text{Time (Vector Exec)}$$
$$= \text{WorkDone(Vector Exec)} / \text{WorkDone(Scalar Exec) [in iso time]}$$

If measurable, great. Could be non-trivial, especially, wall-clock time in multithreaded code.




- Proxy Metrics
 - Estimated Vectorization Gain/Efficiency (static + “simulation” + dynamic in Advisor)
 - Path Reduction (static or dynamic #inst count)
 - FLOPs increase
 - VPU HW Utilization (dynamic PMU approximate)
 - Mask register utilization (AVX-512 specific)

Estimated Vectorization Gain/Efficiency

























- **G** = Scalar Exec Cost / Vector Exec Cost
 - Ideally, we'd like G (estimate) to match S (actual) for VPU bound code. Very difficult.
 - **Vectorizer's View of Vector Code Quality**
- **E** = G / Max_G
 - Max_G is, naively speaking, vector width: e.g., "CPU has a 4-way vector and thus I want 4x speedup!!"
- **G_static** - available in Intel Compiler . **G_static_dynamic** available in Advisor.
 - Vector Advisor 2016 **adjusts** Compiler reported "gain" to account for :
 - **Dynamic** info on trip count.
 - **Dynamic** Peel/remainder execution. Special information on compiler overhead.
 - More **dynamic** adjustments (MASK utilization, Access Pattern) are planned in 2017 release

Gain/Efficiency: Advisor XE 2016

(support for Intel Xeon, Intel Core and 2nd generation Intel Xeon Phi)

Elapsed time: 5,46s  Vectorized  Not Vectorized 

FILTER: All Modules All Sources

Loops	Loop Type	Self Time	Vectorized Loops		
			Vecto...	Efficiency ▾	Estimated Gain
  [loop in fCollisionBGK at lbpBGK.cpp:840]	Vectorized: Expand	0,020s	AVX	 100%	2,05
  [loop in fGetFracSite at lbpGET.cpp:152]	Vectorized: Expand	0,030s	AVX	 36%	2,90
  [loop in fGetOneMassSite at lbpGET.cpp: ...]	Vectorized: Expand	0,089s 0	AVX	 36%	2,86
  [loop in fGetEquilibriumF at lbpSUB.cpp:729]	Vectorized: Collapse	0,579s 	AVX	 25%	2,00
  [loop in fGetEquilibriumF at lbpSUB.cpp: ...]	Vectorized (Body)	0,431s 	AVX		
  [loop in fGetEquilibriumF at lbpSUB.cpp: ...]	Vectorized (Remainder)	0,087s 0	AVX		
  [loop in fGetEquilibriumF at lbpSUB.cpp: ...]	Remainder	0,061s 0			
  [loop in fPropagationSwap at lbpSUB.cpp:1 ...]	Vectorized (Body)	1,259s 	AVX	 ~-14%	0,54

Advisor : Survey Analysis Type

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Gain/Efficiency Need to Know

- Vectorizer is in the middle of the compiler
 - Sometimes Hard to predict what happens downstream. May be a delta between what vectorizer sees and the ASM generated.
- Compiler - Static performance model, not dynamic measurement
 - Mask values, trip count and other things are unknown
 - ✓ *Addressed by Vector Advisor*
- High Gain doesn't mean better Speedup if not VPU bound
 - Latency- and bandwidth- bound codes
- In current version reported only for Compiler-vectorized code (i.e., not for vector intrin and inline ASM)

SPEED-UP YOUR VECTOR CODE:

- Vectorization Efficiency metrics
- Identifying root cause of slow-down
- Performance “Low-hanging fruit” fix example

Vector Efficiency: my “performance thermometer”

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

~91%

~91%: Achieved Vectorization Efficiency

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

Estimated Gain = 3.65x

Vector Length = 4

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

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

(25%): Reference Efficiency for original scalar loop

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

(100%): Theoretical Maximum Vectorization Efficiency

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

Theoretical Maximum Gain = Currently selected Vector Length = 4

How to find out
why Efficiency is <100%?

Intel microprocessors. Performance tests, such as SYSmark and MobileMark, change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to <http://www.intel.com/performance>. See FTC disclaimer.

Factors that slow-down your Vectorized code

1.A. Indirect memory access

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

1.B Memory sub-system Latency / Throughput

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

2. Serialized or “sub-optimal” function calls

```
for (i = 1; i < nx; i++) {  
    sumx = sumx +  
        serialized_func_call(x, y,  
    xp);  
}
```



3. Small trip counts not multiple of VL

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

4. Branchy codes, outer vs. inner loops

```
for(i = 0; i <= MAX; i++) {  
    if ( D[i] < N) do_this(D);  
    else if (D[i] > M) do_that();  
    //...  
}
```

5. MANY others: spill/fill, fp accuracy trade-offs, FMA, DIV/SQRT, Unrolling

Where did I
lose other
45%?

Factors that slow-down your Vectorized code



1.A. Indirect memory access

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

1.B Memory sub-system Latency / Throughput

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

2. Serialized or “sub-optimal” function calls

```
for (i = 1; i < nx; i++) {  
    sumx = sumx +  
        serialized_func_call(x, y,  
    xp);  
}
```

3. Small trip counts not multiple of VL

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

4. Branchy codes, outer vs. inner loops

```
for(i = 0; i <= MAX; i++) {  
    if ( D[i] < N)  
        do_this(D);  
    else if (D[i] > M)  
        do_that();  
    //..  
}
```

5. MANY others: spill/fill, fp accuracy trade-offs, FMA, DIV/SQRT, Unrolling

2.2 Check Memory Access Patterns
Command Line

Issue: Ineffective peeled/remainder loop(s) present
All or some source loop iterations are not executing in the loop body. Improve performance

Recommendation: Add data padding
The trip count is not a multiple of vector length. To fix, do one of the following:

Recommendations

- Optional: Specify the trip count, if it is not constant, using a directive: #pragma loop, Read More:
- opt-assume-safe-padding, Opt-assume-safe-padding: loop_count
- Utilizing Full Vectors and Use of Option -opt-assume-safe-padding, Getting S

Recommendation: Collect trip counts data

Recommendation: Disable unrolling

Recommendation: Specify the expected loop trip count

Recommendation: Use a smaller vector length

Vector Issues	Traits	Data Types	Nu.	Se
2 Possible ineffic ... FMA; Gathers; Type Conversions		Float64; Int32	14	

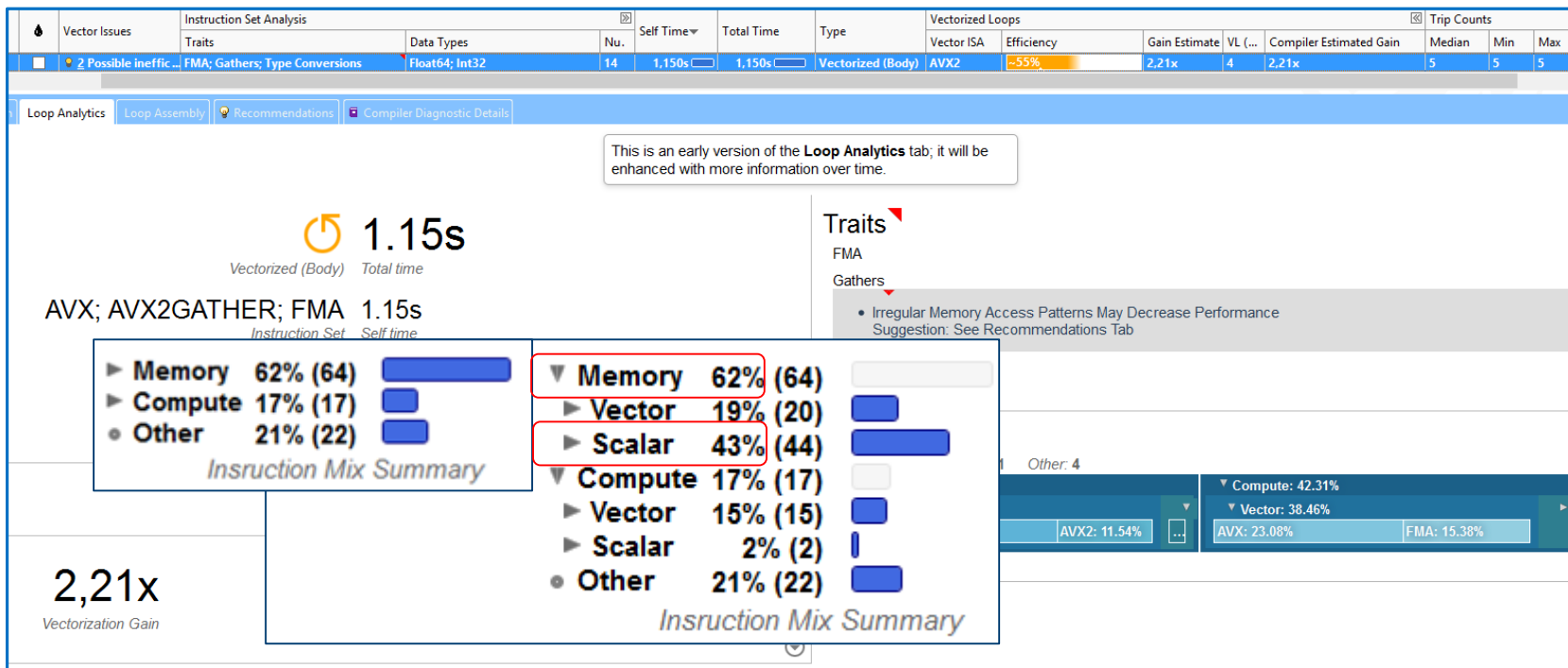
Traits
Gathers
• Irregular Memory Access Patterns May Decrease Performance

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Factors that slow-down your Vectorized code: Survey Loop Analytics



Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



Spend your time efficiently!

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

A typical vectorized loop consists of

1. Main vector body

- Fastest among the three! Maximum VL, no branches/masked computations.

Fastest!

2. (optional) peel part

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

Less
Fast

3. (optional) Remainder part

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

Larger vector register means more iterations (AVX/AVX2) or more time (AVX-512) in peel/remainder

- Align data, make the number of iterations divisible by the vector length where applicable
- Or at least **notify compiler** about alignment and **about number of iterations**

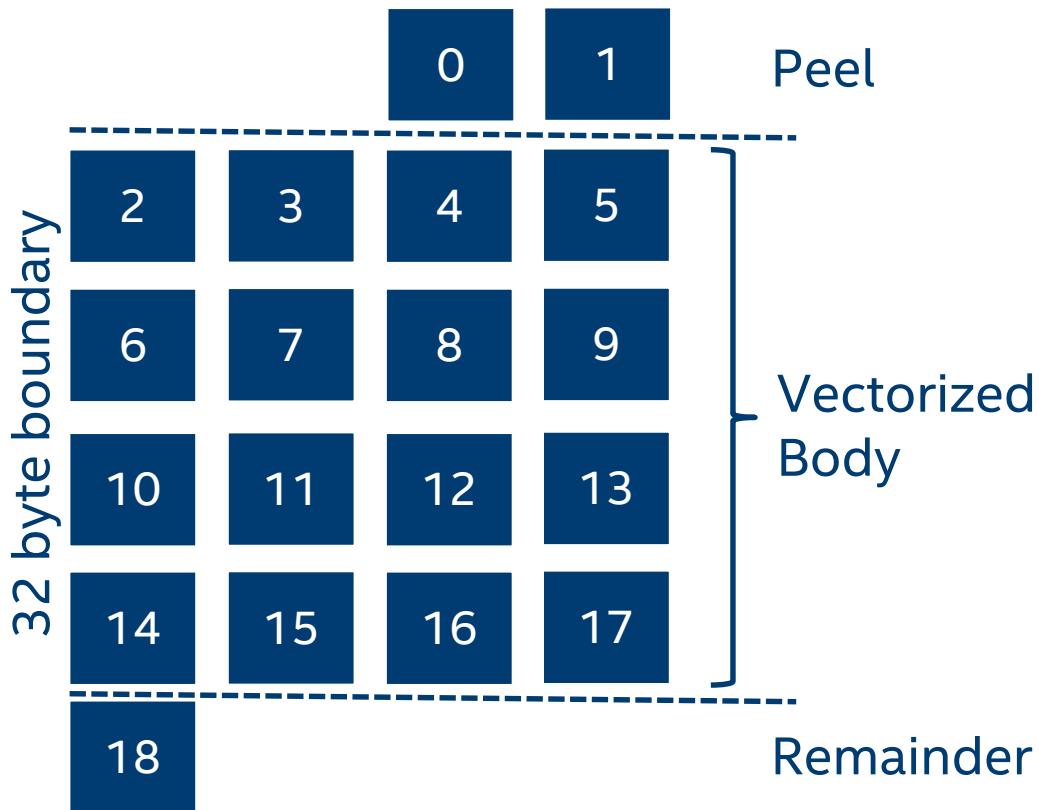
Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



What are peels and remainders?

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



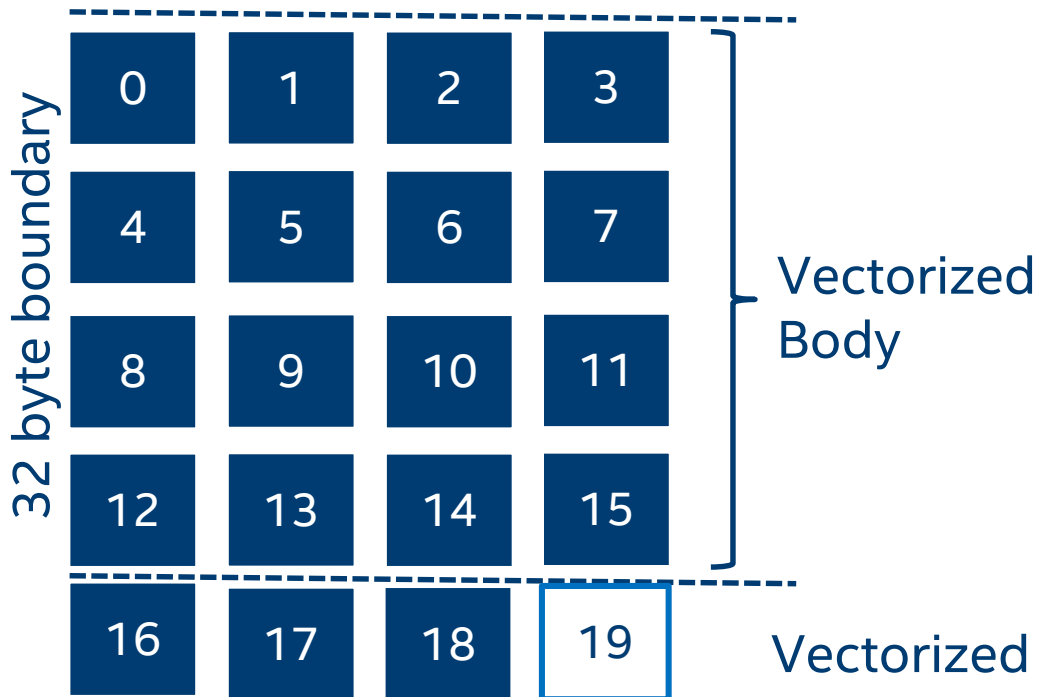
```
// xAVX
// 256 bits wide regs
// holds 4 x 64bit vals

void Func(double *pA)
{
  for (int i=0; i<19;i++)
    pA[i] = ...;
}
```

Example: fGetEquilibriumF

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

(No Peel)

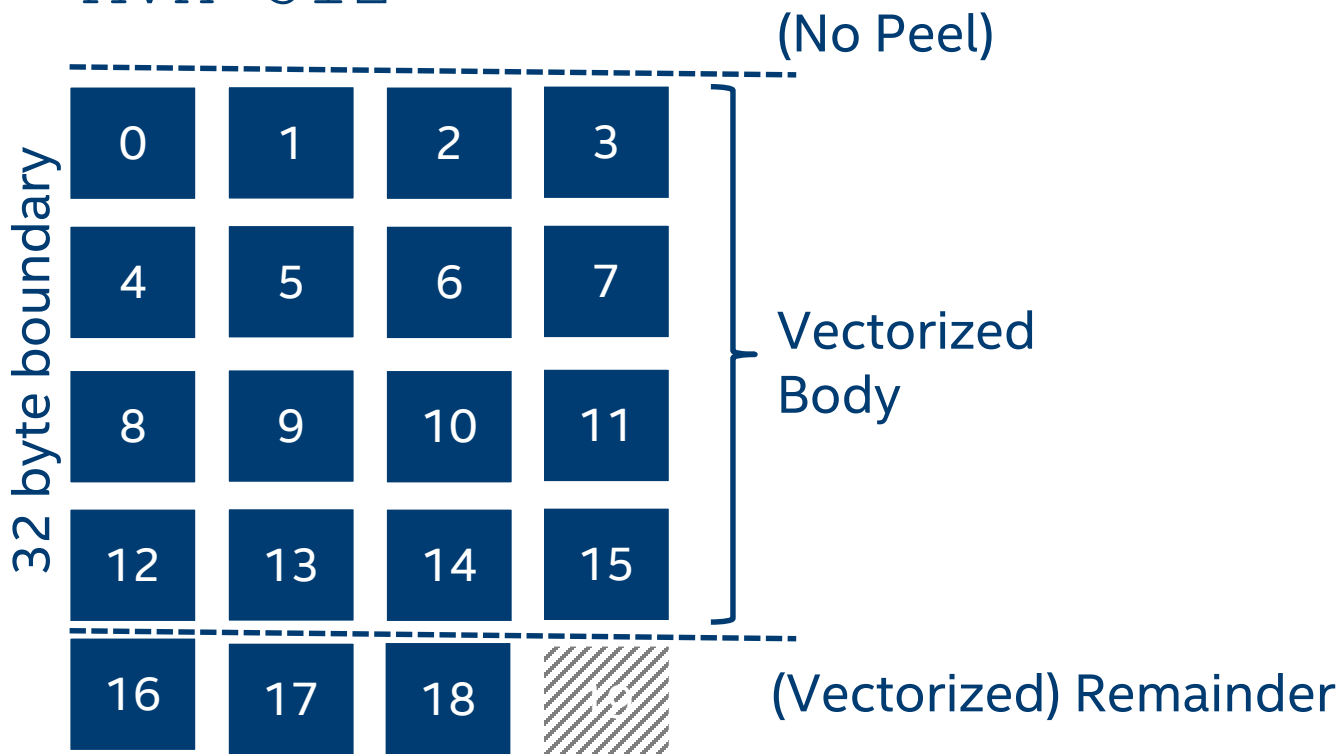


```
// xAVX
// 256 bits wide regs
// holds 4 x 64bit vals

for(i=0; i<lbsy.nq+1; i++)
{
  // . . .
}
```

Example: fGetEquilibriumF AVX-512

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



Example: fGetEquilibriumF

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

```
int fGetEquilibriumF(double *feq, double *v, double rho)
{
  double modv = v[0]*v[0] + v[1]*v[1] + v[2]*v[2];
  double uv;

  for(int i=0; i<lbsy.nq; i++)
  {
    uv = lbv[i*3] * v[0]
        + lbv[i*3+1] * v[1]
        + lbv[i*3+2] * v[2];

    feq[i] = rho * lbw[i]
             * (1 + 3.0 * uv + 4.5 * uv * uv - 1.5 * modv);
  }
  return 0;
}
```

- Low loop count
- Not multiple of Vector Length

19

Example: fGetEquilibriumF

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

```
int fGetEquilibriumF(double *feq, double *v, double rho)
{
  double modv = v[0]*v[0] + v[1]*v[1] + v[2]*v[2];
  double uv;

  #pragma loop_count (19)

  for(int i=0; i<lbsy.nq; i++)
  {
    uv = lbv[i*3] * v[0]
        + lbv[i*3+1] * v[1]
        + lbv[i*3+2] * v[2];

    feq[i] = rho * lbw[i]
              * (1 + 3.0 * uv + 4.5 * uv * uv - 1.5 * modv);
  }
  return 0;
}
```

- Loop #pragma loop_count (cheapest fix)
- Padding (medium cost fix)

Survey: Body-Peel-Remainder break-down

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

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

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	

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Get Specific Advice For Improving Vectorization

Intel® Advisor XE – Vectorization Advisor

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

Click to see recommendation

⊕	⚠	[loop in fGetOneMassSite at l...	<input type="checkbox"/>	⚠ 1 Ineffective peeled/remainder loop(s) present	AVX	~33%	1,31x	4	1,31x	0,267s	8,3%
⊕	⚠	[loop in fGetSpeedSite at lbr...	<input type="checkbox"/>	⚠ 1 Data type conversions present	AVX2	~100%	5,14x	4	5,14x	0,240s	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++)
```

Advisor XE shows hints how to decrease vectorization overhead

Read More:

- [loop_count](#)
- [Getting Started with Intel Compiler Pragmas and Directives and Other Resources for Intel® Advisor Users](#)

- 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

Optimization Notice



SPEED-UP YOUR VECTOR CODE:

Though #1 problem: memory access pattern

- Advisor MAP (stride) Analysis
- AoS and SoA
- Loopnest: choose right level for vector parallelism

Non-Contiguous Memory Access

Potential to vectorize but may be inefficient

Unit-Stride access

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



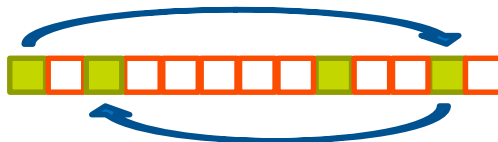
Constant stride access

```
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]
```



Non-Contiguous Memory Access

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]
```

- To make SIMD efficient – need to load/store data into vector register using single contiguous vector load/store instruction

Survey Recommendations: Vectorized loops whose performance was *possibly* limited by **data layout**

Click to see recommendation

Loops	Vector Issues	Total Time	Loop Type	Why No Vectorizat...	Vectorized Loops	
					Vect...	Efficiency
[loop in fCalcInteraction_ShanChen at lbpFOR...]	1 Ineffective peeled/remainder loop(s) pr...	0.944s	Peeled/Remain...			
[loop in fGetEquilibriumF at lbpSUB.cpp:826]	2 Possible inefficient memory access p...	0.921s	Vectorized (B...		AVX	~59%
[loop in fCalcInteraction_ShanChen at lbpFOR...]	1 Data type conversions present	0.877s	Scalar	inner l...		

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.

Advisor Memory Access Pattern (MAP): know your access pattern

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]
```

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

blue color: fraction of unit stride accesses
 yellow: "fixed" stride accesses ratio
 red color: fraction of irregular (variable stride) accesses

ID	Stride	Type	Source	Site Name	Variable
P1	3	16% / 84% / 0%	Mixed strides		
<pre> 1246 #endif 1247 for (int m=1; m<=half; m++) { 1248 nextx = fCppMod(i + lbv[3*m] 1249 nexty = fCppMod(j + lbv[3*m+ 1250 nextz = fCppMod(k + lbv[3*m+ </pre>					
P11	0; 1				
P12	-289559; -274359; -14477; -13717; -13679; 723; 302519;				
<pre> 1251 ilnext = (nextx * Ymax + nex 1252 #ifndef SWAP_OVERLAP 1253 f\$swapPair (lbf[i]*lbsitelength + 1*lbsy. </pre>					

- 16%: percentage of memory instructions with unit stride or stride 0 accesses
 Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration
 Stride 0 = Instruction accesses the same memory from iteration to iteration
- 84%: percentage of memory instructions with fixed or constant non-unit stride accesses
 Constant stride (stride N) = Instruction accesses memory by N elements from iteration to iteration
 Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4*sizeof(double)) with each iteration
- 0%: percentage of memory instructions with irregular (variable or random) stride accesses
 Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration
 Typically observed for indirect indexed array accesses, for example, a[index[i]]
- gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture

Non-Contiguous Memory Access

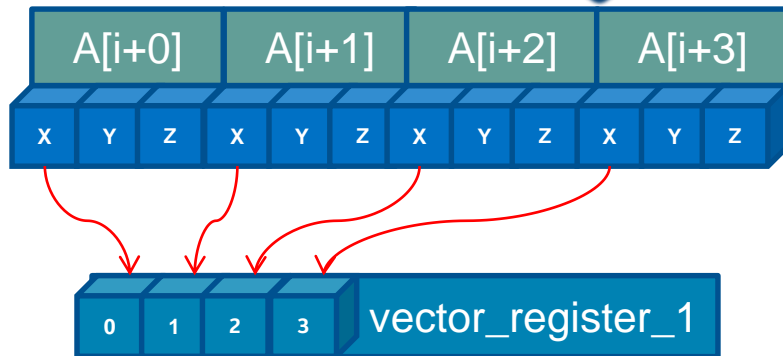
Potential to vectorize but may be inefficient

Courtesy Alex Wells

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

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

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

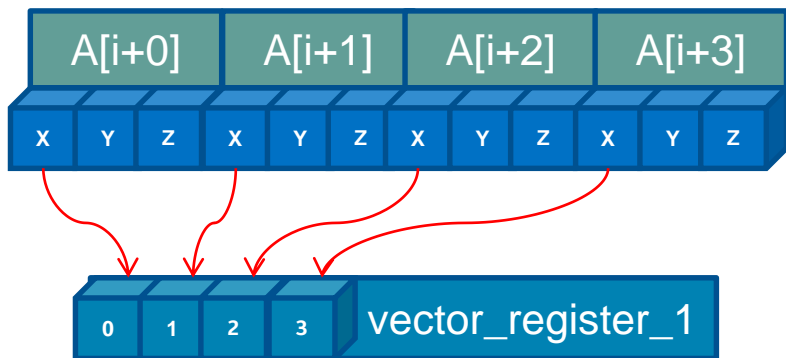


- ~~To make SIMD efficient – need to load/store data into vector register using single contiguous vector load/store instruction~~

Array of Structures (AoS) vs. Structure of Arrays (SoA)

```
struct Triangle  
{Float x,y,z;}  
Triangle A[100];
```

AoS in Memory Layout: **“Constant stride”**

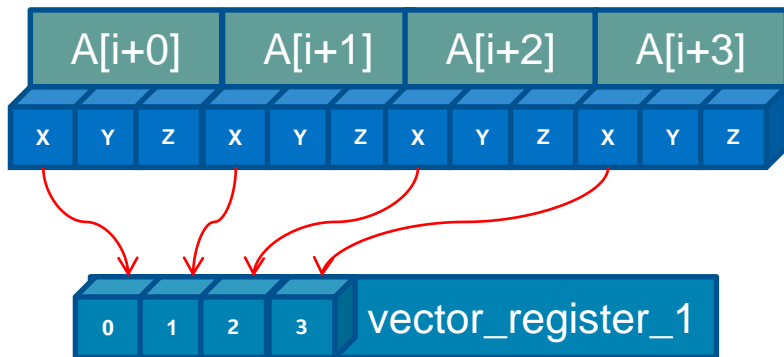


Object-oriented programming motivates AoS

Array of Structures (AoS) vs. Structure of Arrays (SoA)

```
struct Triangle  
{Float x,y,z;}  
Triangle A[100];
```

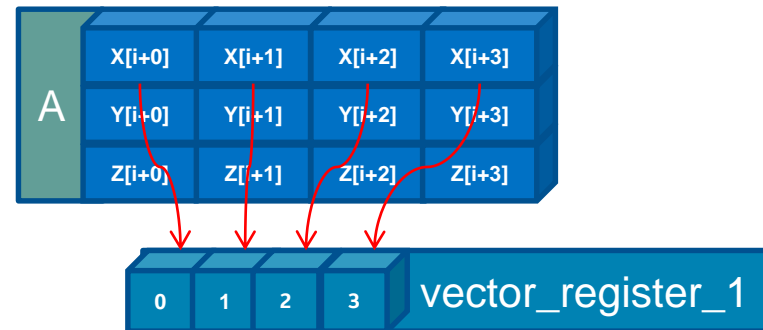
AoS in Memory Layout



```
struct Triangles  
{Float x[100],y[100],z[100];}  
Triangles A;
```

SOA in Memory Layout:

Unit Stride!



Object-oriented programming motivates AoS

MAP for Outer Loop Vectorization

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

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

x = 0

z = z * c

z = z * c

iters = 2

x = 1

z = z * c

z = z * c

....

iters = 23

x = 2

z = z * c

z = z * c

.....

iters = 255

x = 3

z = z * c

z = z * c

.....

iters = 37

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

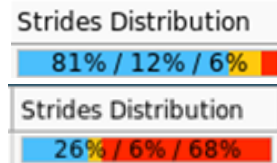
```
for(int i=0; i<Xmax; i++)           ← Here?  
  for(int j=0; j<Ymax; j++)  
    for(int k=0; k<Zmax; k++) {     ← Here????  
      //do some work  
      for (int l=0; l<qdim; l++) {   ← Here???  
        for (int m=1; m<=half; m++) { ← Here??  
          //...  
          fSwapPairM (l, m, k, j,..);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

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

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 (l, m, k, j,...);  
        }  
      }  
    }  
  }  
}
```



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

100% VECTOR EFFICIENCY! AM I DONE?

ADVISOR 2017 BETA: MEMORY BOUND CODES ANALYSIS

[Register for Intel Advisor
2017 "Beta"!](#)

Send request to
vector_advisor@intel.com

Vectorized loops with high efficiency

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

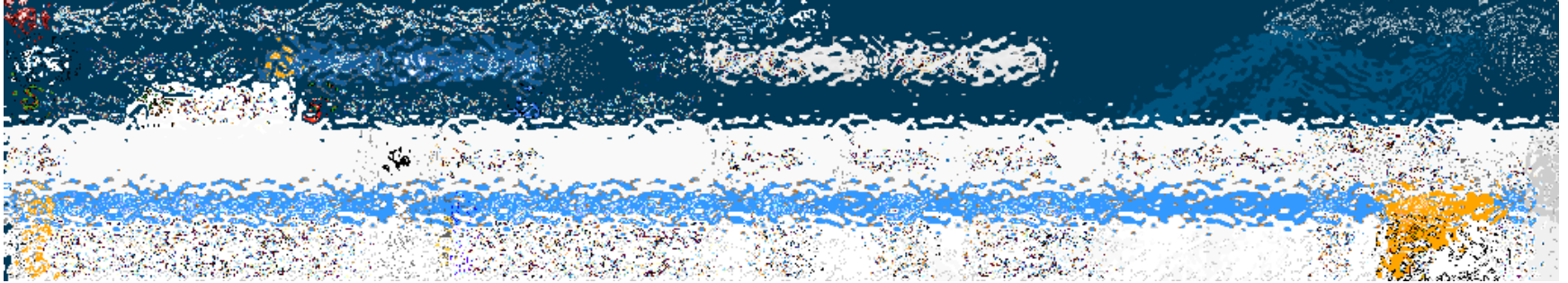
Elapsed time: 12.61s Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey Report Refinement Reports Annotation Report

Loops	🔥	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops		
							Vect...	Efficiency	Gai
🔍 [loop in fGetEquilibriumF at lbpSUB.c...	☐	💡 1 Data type conversions present	0.355s I	0.355s I	Vectorized (Bo...	AVX	100%	5.03	
⊕ [loop in fCollisionBGK at lbpBGK.cpp:...	☐	💡 1 Ineffective peeled/remainder ...	0.047s I	0.047s I	Vectorized (Re...	AVX	67%	1.34	
⊕ [loop in fGetFracSite at lbpGET.cpp:19...	☐	💡 1 Possible inefficient memory a...	0.020s I	0.020s I	Vectorized Vers...	AVX	19%	1.56	

Are we done??..

Vectorized loops with high efficiency



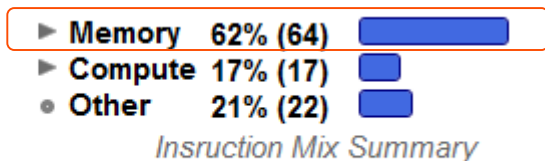
...It depends.

If code is not SIMD bound,
then $\text{Speedup} \leq \text{Vectorization Gain}$

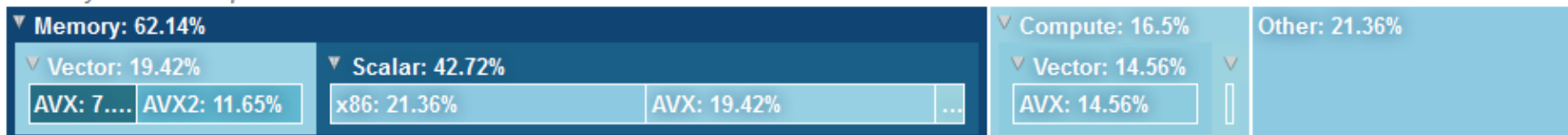
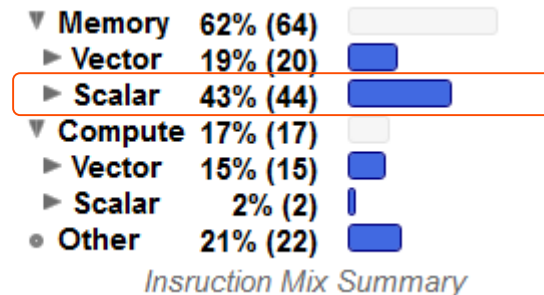
In addition (instead of) VPU-bound
code could be Memory Bound

Am I bound by VPU/CPU or by Memory?

Quick and Dirty check with Survey Loop Analytics.



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



Am I bound by VPU/CPU or by Memory?: Advisor Memory Access Pattern and Footprint

Footprint	Small enough	Big enough
Access Pattern		
Unit Stride	Effective SIMD No Latency and BW bottlenecks	Effective SIMD Bandwidth bottleneck
Const stride	Medium SIMD Latency bottleneck possible	Medium SIMD Latency and Bandwidth bottleneck possible
Irregular Access, Gather/Scatter	Bad SIMD Latency bottleneck possible	Bad SIMD Latency bottleneck

Source	Stride	Operand Type	Operand Size ...	Aggregated footprint
m=1; m<=half; m++) { = fCppMod(i + lbv[3*m], Xmax); = fCppMod(j + lbv[3*m+1], Ymax); = fCppMod(k + lbv[3*m+2], Zmax); = (nextx * Ymax + nexty) * Zmax + nextz;	[0] [0] [3] [0] [3] [0] [3]	int int int int	32 32:64 32 32	4B 88B 88B 88B
lbsitlength + 1*lbsy.nq + m + half], lbf[ilnext*lbsitele	[0] [1] [-4..	float64:int	32:64	9MB
lbsitlength + 1*lbsy.nq + m + 1], lbf[il*lbsitlength + 1				

Assembly	Physical Stride	Operand Info	Address range	Memory access
x3				
f298 1254 add r14d, r12d				
f2cb 1256 mov r12, qword ptr [r9+rsi*8]	[-43775, 118377...	int*1, int*1, i...	0x27561058 - 0x27e6cf20	9MB
f2cf 1256 vmovsd xmm0, qword ptr [r8+rbx*8]	[1]	float64*1	0x27561098 - 0x275610d0	64B
f2d5 1256 mov qword ptr [r8+rbx*8], r12	[1]	int*1	0x27561098 - 0x275610d0	64B
f2d9 1256 mov r13d, dword ptr [rip+0x1565bc]	[0]	int*1	0x18589c - 0x18589c	4B

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.




Detect possible latency or bandwidth issues

Enhanced Memory Analysis

Is your
memory
range small
enough to fit
in cache?

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

 Gather (irregular) access

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

▼ **Gather details**

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

Optimization Notice

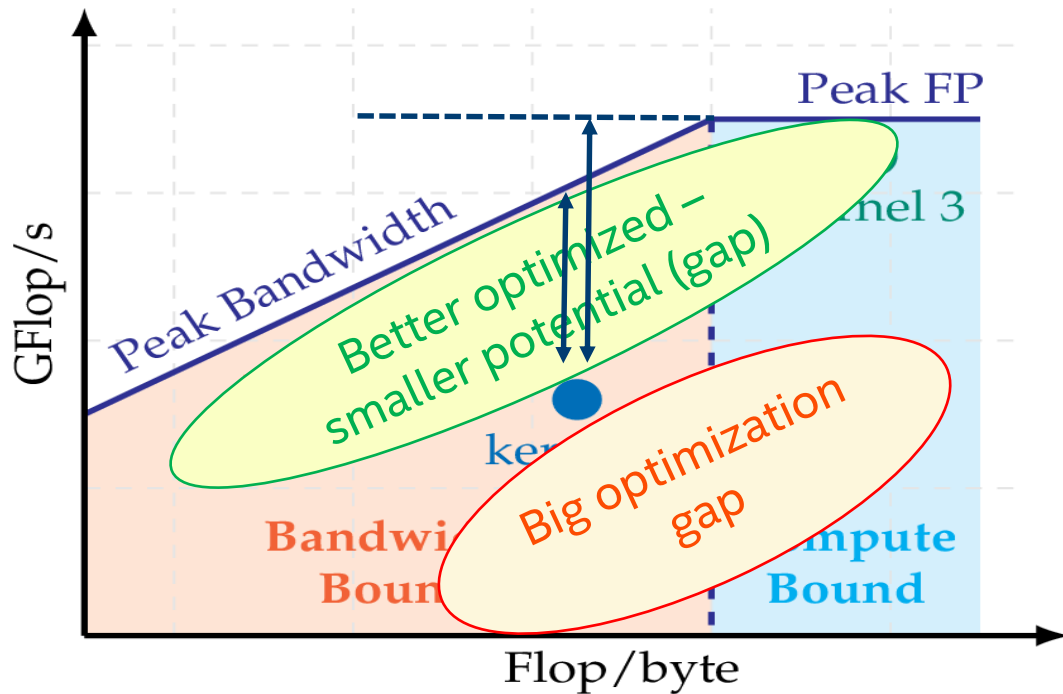
Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Am I bound by VPU/CPU or by Memory?: SIMD vs. Memory BandWidth vs. Latency in MAP

Footprint	Small enough	Big enough
Access Pattern		
Unit Stride	Effective SIMD No Latency and BW bottlenecks	Effective SIMD BW bottleneck
“Strided” access: Const stride (Gather*)	Medium SIMD Latency bottleneck possible	Medium SIMD Latency and BW bottleneck possible
Irregular Access, Gather	Bad SIMD Latency bottleneck possible	Bad SIMD Latency bottleneck

Am I bound by VPU/CPU or by Memory?

ROOFLINE ANALYSIS



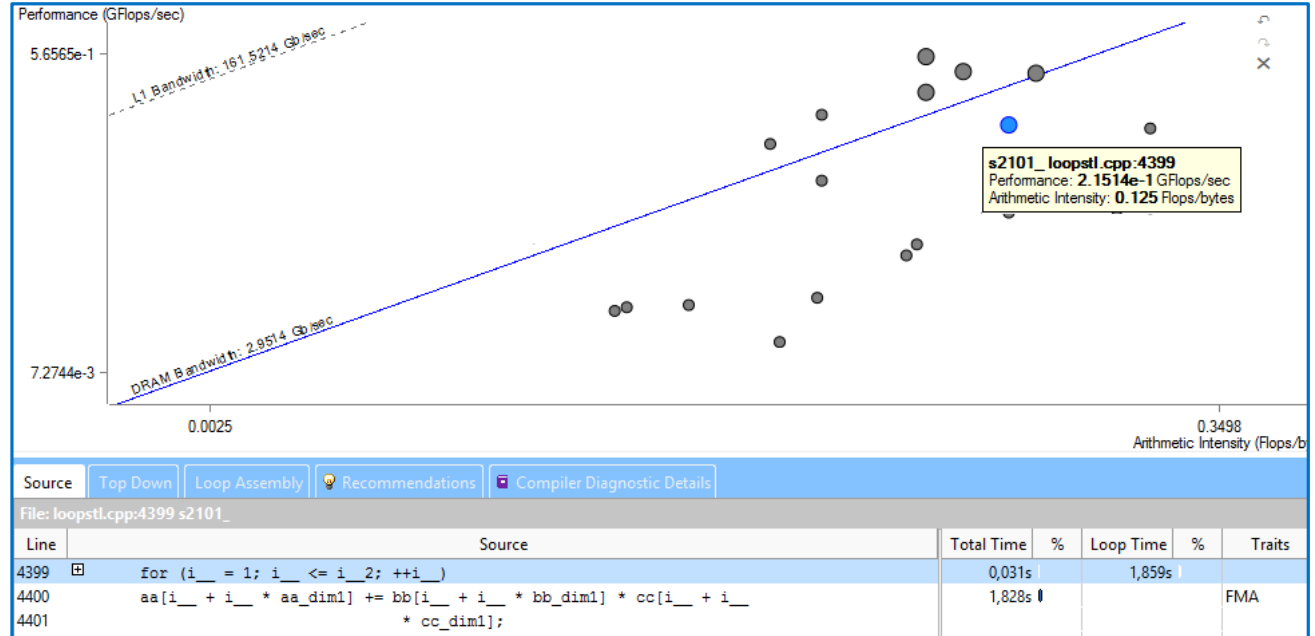
Am I bound by VPU/CPU or by Memory?

Advisor ROOFLINE ANALYSIS

Advisor Roofline
Planned:

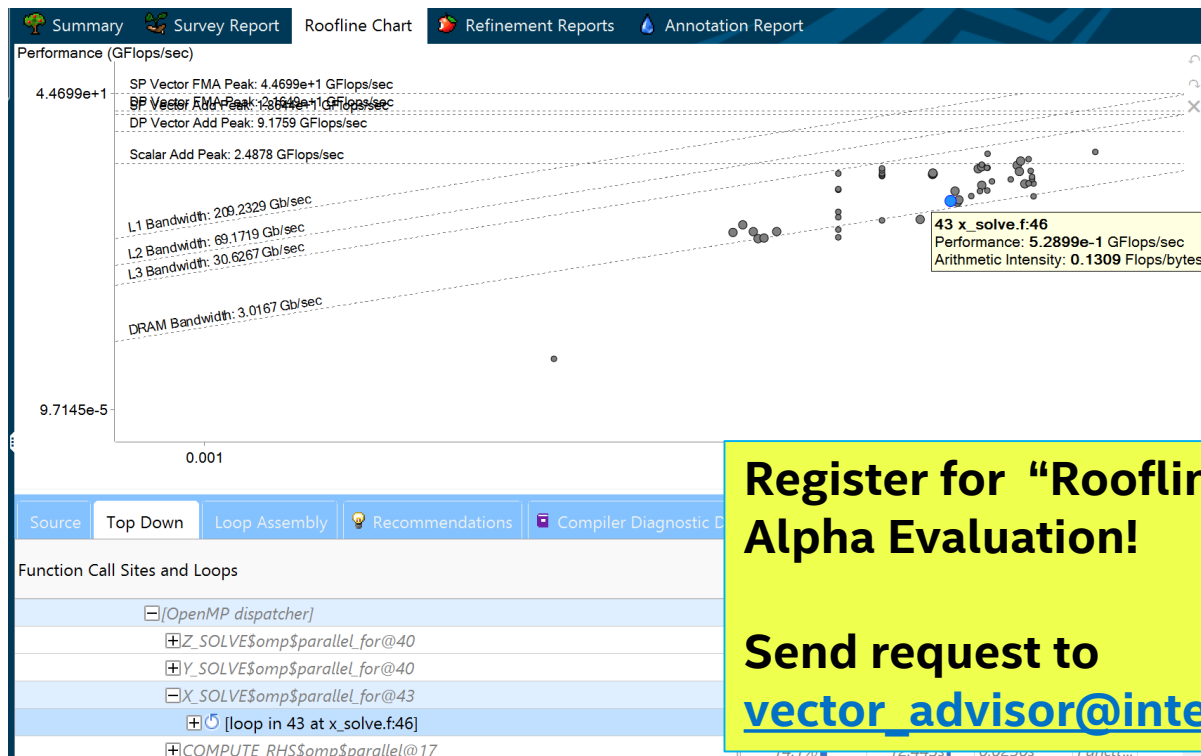
- CARM
- Classic
- Mask-aware

Interactively
mapped to other
data sources



Am I bound by VPU/CPU or by Memory?: Advisor ROOFLINE ANALYSIS

Future
Advisor 2017
Releases



Register for “Roofline”
Alpha Evaluation!

Send request to
vector_advisor@intel.com

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Conclusion/Summary

Modernizing your Code

To get the most out of your hardware, you need to modernize your code for vectorization , memory and threading.

Taking a methodical approach and taking advantage of the powerful tools in Intel® Parallel Studio XE,

can make the modernization task dramatically easier.

ADVISOR 2017 BETA: FLOPS AND MASK ANALYSIS

[Register for Intel Advisor 2017 “Beta”!](#)

Send request to
vector_advisor@intel.com

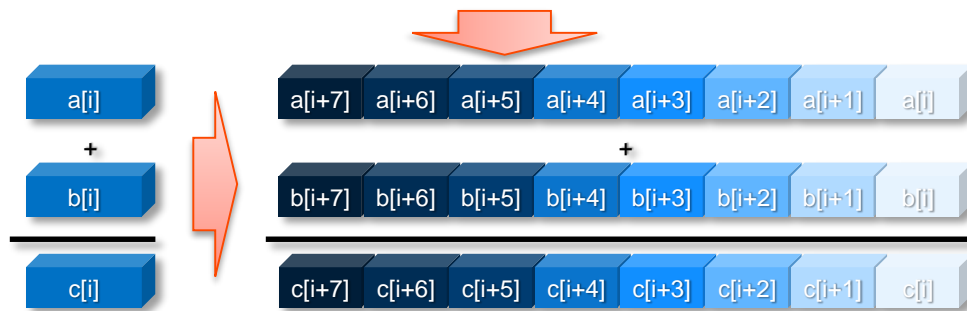
Mask Utilization and FLOPS profiler

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

Why Mask Utilization Important?

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

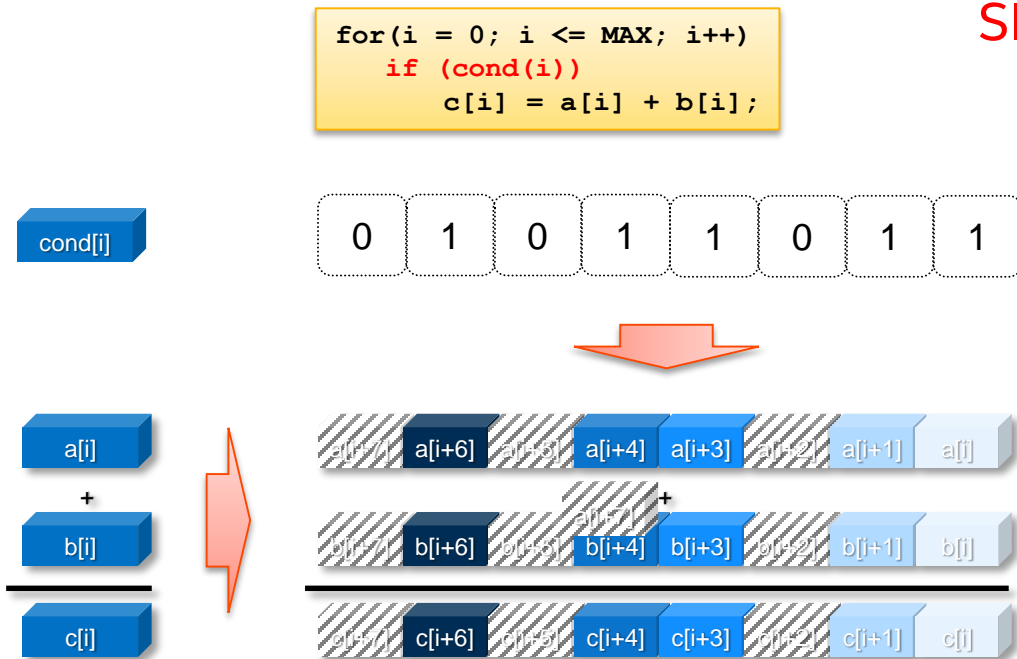
100%



Why Mask Utilization is Important?

3 elements suppressed

SIMD Utilization = 5/8
(62.5%)



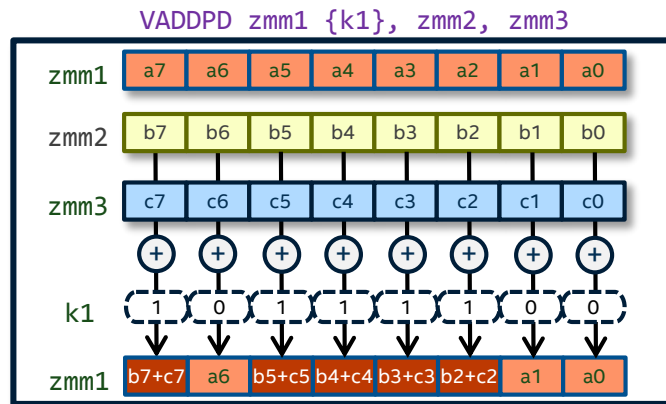
AVX-512 Mask Registers

8 Mask registers of size 64-bits

- k1-k7 can be used for predication
 - k0 can be used as a destination or source for mask manipulation operations

4 different mask granularities.
For instance, at 512b:

- Packed Integer Byte use mask bits [63:0]
 - VPADDB `zmm1 {k1}, zmm2, zmm3`
- Packed Integer Word use mask bits [31:0]
 - VPADDW `zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP32 and Integer Dword use mask bits [15:0]
 - VADDPS `zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP64 and Integer Qword use mask bits [7:0]
 - VADDPD `zmm1 {k1}, zmm2, zmm3`

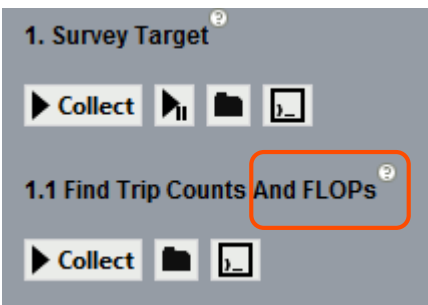


		Vector Length		
		128	256	512
element size	Byte	16	32	64
	Word	8	16	32
	Dword/SP	4	8	16
	Qword/DP	2	4	8

FLOP, FLOP/s, Mask Utilization Profiler

set ADVIXE_EXPERIMENTAL=FLOPS

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



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

FLOP/s metric

- Integrated with Trip Counts
- Additionally provides L1 bytes traffic and (AVX-512) Mask Register Utilization
- Currently mapped to loops/functions/workload
- **GFLOPs/s**
 - Analogy to **IPC**
 - **Mask-aware, focus on “useful computation”**
 - vs. **Vector Efficiency**

FLOPs, Masks, Trip Counts		Vectorized Loops		
Median	GFLOPs/s ▼	Arithmetic...	Vect...	Efficiency
12	8.36101	0.15	AVX2	~96%
6	8.34294	0.125	AVX	~100%
	8.33731	0.166667	AVX2	~47%
	8.30612	0.125	AVX2	~100%
	8.04988	0.125	AVX	~100%
5; 7	7.52399	0.125	AVX2	~96%
1; 64; 2	7.49084	0.1875	AVX	~100%
5; 6	6.89345	0.1875	AVX	~100%
6	6.2507	0.1	AVX2	~99%
6	6.24983	0.25	AVX	~45%
	6.2464	0.125	AVX2	~97%
5; 7	5.69063	0.117647	AVX	~100%
3; 3	5.4257	0.170606	AVX2	~28%
12	4.56186	0.1	AVX2	~79%
5; 7	4.5496	0.142857	AVX2	~98%
12; 3	4.17379	0.0833333	AVX	~43%
6	4.16433	0.125	AVX2	~90%
	4.15745	0.166667	AVX2	~42%
12; 4	3.93297	0.070028	AVX	~99%
	3.66404	0.1	AVX2	~52%
49; 49; 49 ...	3.38575	0.0833333	AVX	~65%
	3.12171	0.125	AVX2	~51%
6	3.11463	0.125	AVX2	~100%
5	2.97544	0.25	AVX2	~54%
	2.84187	0.125	AVX2	~42%
12	2.78592	0.125	AVX2	~100%
	2.28076	0.25	AVX	~29%
6	2.25889	0.125	AVX2	~100%
	2.08228	0.1	AVX2	~19%
6	1.77923	0.125	AVX2	~100%
12	1.74944	0.153846	AVX2	~30%

VECTORIZATION ANALYSIS FOR AVX-512 PLATFORMS

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

AVX-512 ERI – specific to Intel® Xeon Phi

Loops	Vector Issues	Self Time	Loop Type	Vectorized Loops				Instruction Set Analysis					
				Vector ISA	Efficiency	Gain Esti...	VL (V...	Traits	Data Types	Vector ...	Instruction Sets		
[loop]	3 Possible in...	35.226s	5.4%	Vectorized+Threaded (Body; Peeled; Re...	AVX512	-26%	2.21x	8	Divisions; FMA; Gathers	Float32; ...	256/512	AVX; AVX2; AVX512; ...	Masked L
[loc]	2 Possible in...	26.025s	4.0%	Vectorized (Body)+Threaded (OpenMP)	AVX512			8	Divisions; Gathers; FMA	Float32; ...	256/512	AVX; AVX512ER; AVX512F...	
[loc]	1 High vecto...	5.876s		Vectorized (Peeled)+Threaded (OpenMP)	AVX512			8	Divisions; Gathers; FMA	Float32; ...	256/512	AVX2; AVX512ER; AVX512...	Masked Lc
[loc]	1 High vecto...	3.324s		Vectorized (Remainder)+Threaded (Open...	AVX512			8	Divisions; Gathers; FMA	Float32; ...	256/512	AVX2; AVX512ER; AVX512...	Masked Lc
[loop]		34.599s	5.3%	Vectorized (Body; Remainder)	AVX512	-70%	5.64x	8	Divisions; FMA; Square Roots	Float32; ...	256/51...	AVX2; AVX512ER; AVX512...	Masked Lc
[loop]	1 Possible in...	33.849s	5.2%	Vectorized (Body; Peeled; Remainder)	AVX512	-28%	2.24x	8	Divisions; FMA; Gathers	Float32; ...	256/512	AVX; AVX2; AVX512ER; AV...	Masked Lc
[loop]		19.839s	3.1%	Vectorized (Body; Remainder)	AVX512	-72%	11.46x	16.8					

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

Issue: Possible inefficient memory access patterns present
 Inefficient memory access patterns may result in significant vector code execution slowdown or block automatic vectorization by the compiler. Improve performance by investigating.
 Recommendation: Confirm inefficient memory access patterns
 There is no confirmation inefficient memory access patterns are present. To confirm: Run a [Memory Access Patterns analysis](#).

Confidence: Need More Data

Issue: Ineffective peeled/remainder loop(s) present
 All or some [source loop](#) iterations are not executing in the [loop body](#). Improve performance by moving source loop iterations from [peeled/remainder](#) loops to the loop body.
 Recommendation: Collect trip counts data
 The Survey Report lacks [trip counts](#) data that might generate more precise recommendations. To fix: Run a [Trip Counts analysis](#).
 Recommendation: Align data
 Recommendation: Add data padding
 The [trip count](#) is not a multiple of [vector length](#). To fix: Do one of the following:
 • Increase the size of objects and add iterations so the trip count is a multiple of vector length.
 • Increase the size of static and automatic objects, and use a compiler option to add data padding.

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

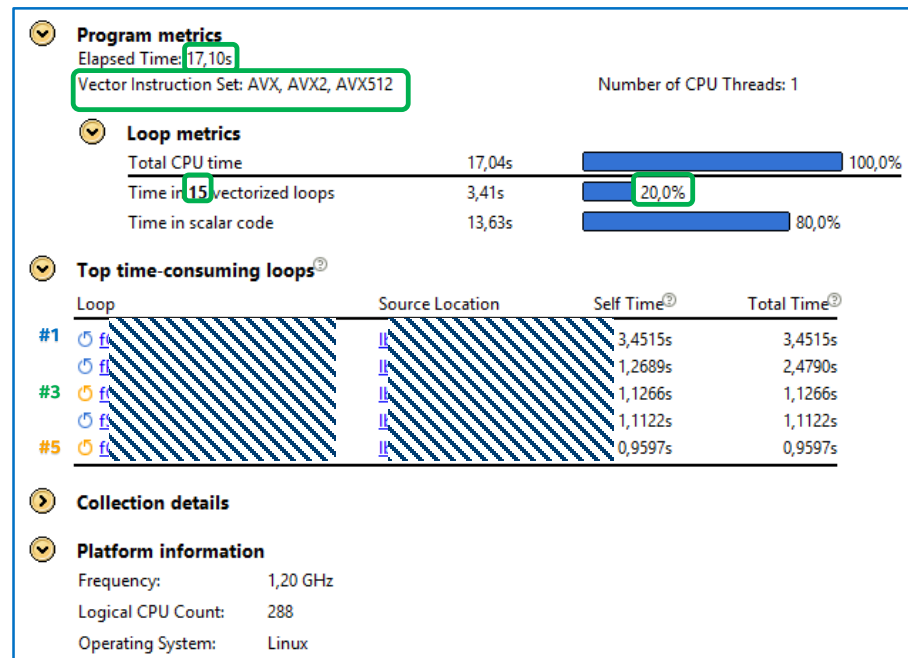
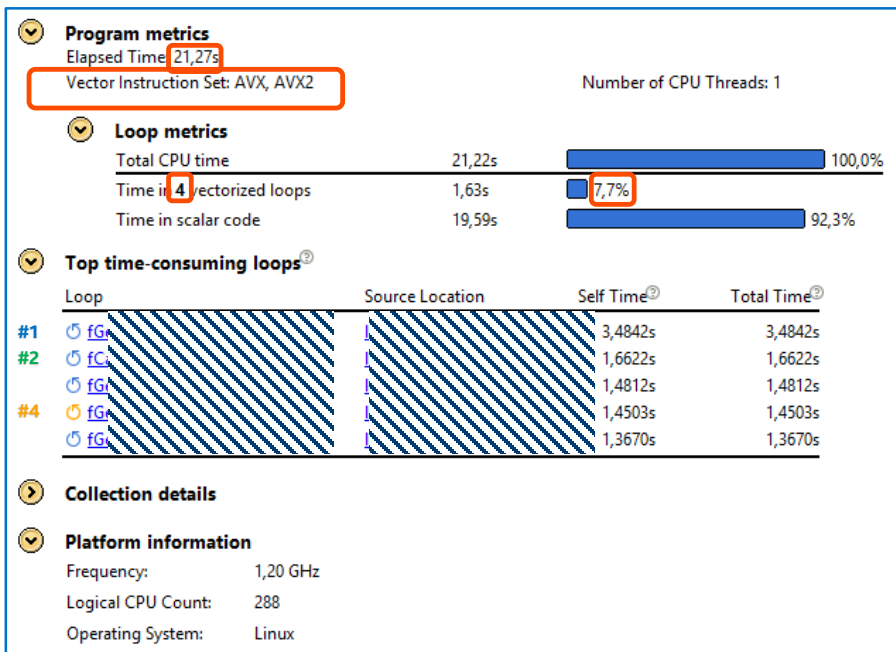
Total CPU time	454.08s	100.0%
Time in 88 vectorized loops	41.86s	9.2%

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
 *Other names and brands may be claimed as the property of others.



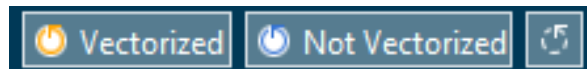
Characterize and compare AVX-512 against AVX2 versions (on Xeon Phi)



AVX-512-specific performance trade-offs

Advisor AVX-512 Recommendations

Increasing Vector Register Size ->



Increase fraction of time spent in Remainders

Function Call Sites and Loops	🔥	Vector Issues	Self Time	Total Time	Type	Vectoriz
[-] 🔥 [loop in fCollisionBGKShanChenSom ...]	<input type="checkbox"/>	🔦 1 Ineffective peeled/remainder loop(s ...)	0,110s ↓	0,110s ↓	Vectorized (Remainder; [Body])	AVX512
[-] 🔥 [loop in fCollisionBGKShanChenSo ...]	<input type="checkbox"/>		0,110s ↓	0,110s ↓	Vectorized (Remainder)	AVX512
[-] 🔥 [loop in fCollisionBGKShanChenSo ...]	<input type="checkbox"/>		n/a	n/a	Vectorized (Body) [Not Executed]	AVX512
[-] 🔥 [loop in fGetFracSite at lbpGET.cpp:19 ...]	<input type="checkbox"/>	🔦 1 Ineffective peeled/remainder loop(s ...)	0,060s ↓	0,060s ↓	Vectorized (Peeled; Remainder; [Body])	AVX512
[-] 🔥 [loop in fGetFracSite at lbpGET.cpp ...]	<input type="checkbox"/>		0,040s ↓	0,040s ↓	Vectorized (Peeled)	AVX512
[-] 🔥 [loop in fGetFracSite at lbpGET.cpp ...]	<input type="checkbox"/>		0,020s ↓	0,020s ↓	Vectorized (Remainder)	AVX512
[-] 🔥 [loop in fGetFracSite at lbpGET.cpp ...]	<input type="checkbox"/>		n/a	n/a	Vectorized (Body) [Not Executed]	AVX512
[-] 🔥 [loop in fCalcInteraction_ShanChen a ...]	<input type="checkbox"/>	🔦 1 Ineffective peeled/remainder loop(s ...)	0,060s ↓	0,060s ↓	Vectorized (Remainder; [Body])	AVX512
[-] 🔥 [loop in fCalcInteraction_ShanChe ...]	<input type="checkbox"/>		0,060s ↓	0,060s ↓	Vectorized (Remainder)	AVX512
[-] 🔥 [loop in fCalcInteraction_ShanChe ...]	<input type="checkbox"/>		n/a	n/a	Vectorized (Body) [Not Executed]	AVX512
[+] 🔥 [loop in fGetOneMassSite at lbpGET.c ...]	<input type="checkbox"/>	🔦 1 Ineffective peeled/remainder loop(s ...)	0,050s ↓	0,050s ↓	Vectorized (Remainder; [Body])	AVX512
[+] 🔥 [loop in fGetTotMomentSite at lbp ...]	<input type="checkbox"/>	🔦 1 Ineffective peeled/remainder loo ...	0,040s ↓	0,040s ↓	Vectorized (Remainder)	AVX512
[+] 🔥 [loop in fGetOneDirecSpeedSite at lbp ...]	<input type="checkbox"/>	🔦 1 Ineffective peeled/remainder loop(s ...)	0,030s ↓	0,030s ↓	Vectorized (Remainder)	AVX512
[+] 🔥 [loop in fGetOneMassSite at lbpGET.c ...]	<input type="checkbox"/>	🔦 1 Ineffective peeled/remainder loop(s ...)	0,030s ↓	0,030s ↓	Vectorized (Remainder)	AVX512
[+] 🔥 [loop in fGetOneDirecSpeedSite at lbp ...]	<input type="checkbox"/>	🔦 1 Ineffective peeled/remainder loop(s ...)	0,020s ↓	0,020s ↓	Vectorized (Remainder)	AVX512

Optimization Notice

AVX-512 Also Benefit Scalar Code a lot...

Survey Static Analysis - AVX-512 “Traits”

Summary Survey Report Survey Source: lbpIO.cpp X Refinement Reports Annotation Report

Source Assembly Stack

File: lbpIO.cpp:206 flnputParameters

Line	Source	Traits
272	lbrigt = fStringToNumber <double> (value);	
273		
274	else if(issue.compare(0,14,"relax_mobility")==0)	
275	lbtmob = 1.0 / fStringToNumber <double> (value);	Appr. Reciprocals(AVX-512ER); Exponent extractions; FMA; Mantissa
276	else if(issue.compare(0,19,"relax_freq_mobility")==0)	
277	lbtmob = fStringToNumber <double> (value);	
Selected (Total Time):		

Module: sibe_o2_avx512_loopcount_28.exe!0x40e8a0

Address	Line	Assembly	Traits
0x414e92		Block 908:	
0x414e92	275	vmovsdq 0x3cdd1e(%rip), %xmm4	
0x414e9a	275	vgetmantsd \$0x0, %xmm1, %xmm1, %k0, %xmm5	Mantissa extractions
0x414ea1	275	vgetexpsdq 0x3cdd0d(%rip), %xmm2, %k0, %xmm2	Exponent extractions
0x414eab	275	vgetexpsd %xmm1, %xmm1, %k0, %xmm3	Exponent extractions
0x414eb1	275	vrcp28sd %xmm5, %xmm5, %k0, %xmm7	Appr. Reciprocals(AVX-512ER)
0x414eb7	275	vsubsd %xmm3, %xmm2, %xmm9	
0x414ebb	275	vmulsd %xmm6, %xmm7, %k0, %xmm8{rne-sae}	
0x414ec1	275	vfnmadd231sd %xmm5, %xmm7, %k0, %xmm4{rne-sae}	FMA

Gather/Scatter Analysis

Motivation

AVX-512 Gather/Scatter-based vectorization.

Much wider usage than before :

- Makes much more codes (profitably) vectorizable
- Gives good average performance, but often far from optimal.
- Demand for explicit profiling and “tweaked” vectorization

Gather/Scatter Analysis

Advisor MAP detects gather “offset patterns”.

Stride	Operand Type
[0]	int;ubyte
[0]	float64;int

Details View

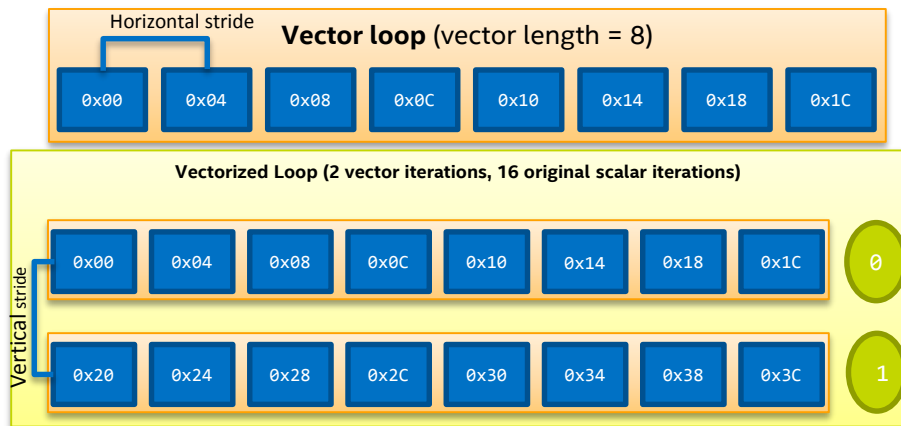
Gather (irregular) access

Operand Size (bits): 64
Operand Type: int*1
Instruction Width: 1
Memory access footprint: 8B

Gather details

Pattern #1: "Invariant"
Instruction gathers values from the same memory throughout the loop
Horizontal stride: 8
Vertical stride: N/A

Mask is constant
Mask: [00000101]
Mask is filled to 25.0%

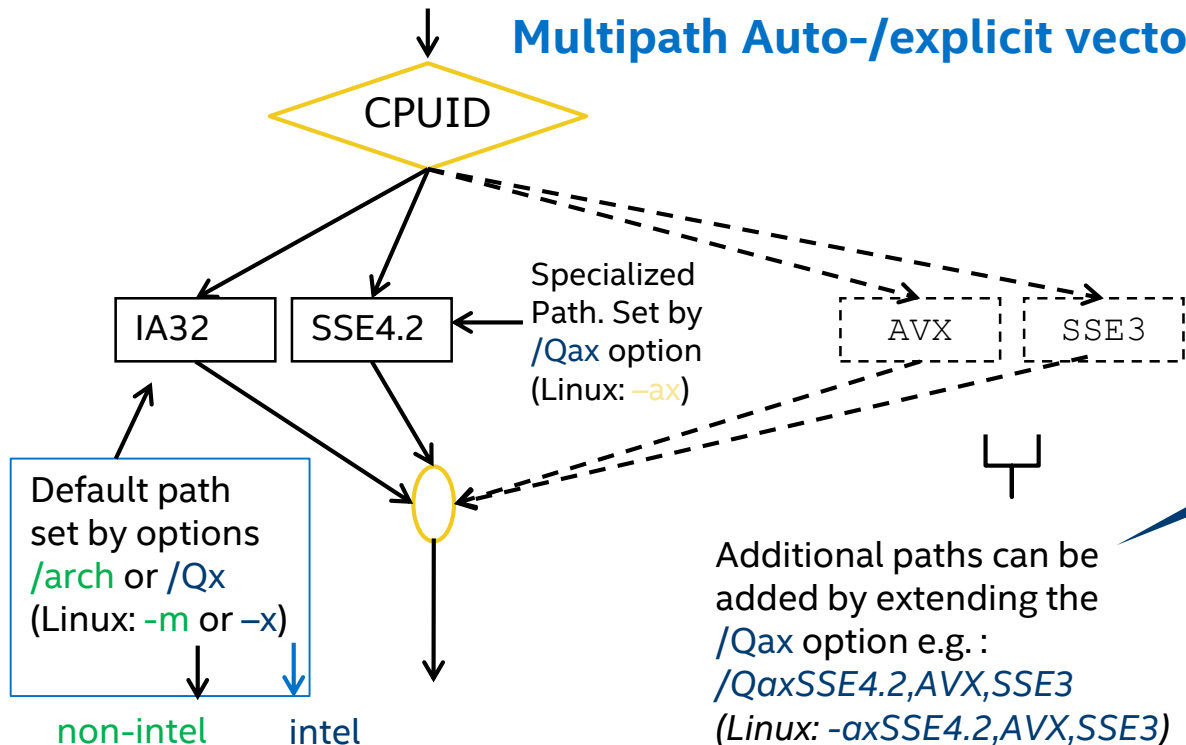


Pattern #	Pattern Name	Horizontal Stride Value	Vertical Stride Value	Example of Corresponding <i>Fix(es)</i>
1	Invariant	0	0	OpenMP uniform clause, simd pragma/directive , refactoring
2	Uniform (horizontal invariant)	0	Arbitrary	OpenMP uniform clause, simd pragma/directive
3	Vertical Invariant	Constant	0	OpenMP private clause, simd pragma/directive
4	Unit	1 or -1	$ \text{Vertical Stride} = \text{Vector Length}$	OpenMP linear clause, simd pragma/directive
5	Constant	Constant = X	Constant = $X * \text{VectorLength}$	Subject for AoS -> SoA transformation

Start Tuning for AVX-512 without AVX-512 hardware

Intel® Advisor - Vectorization Advisor “axcode feature”

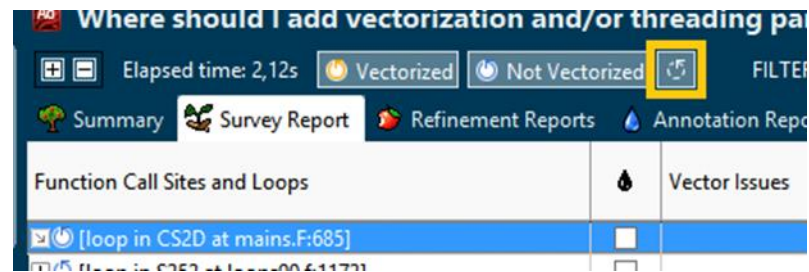
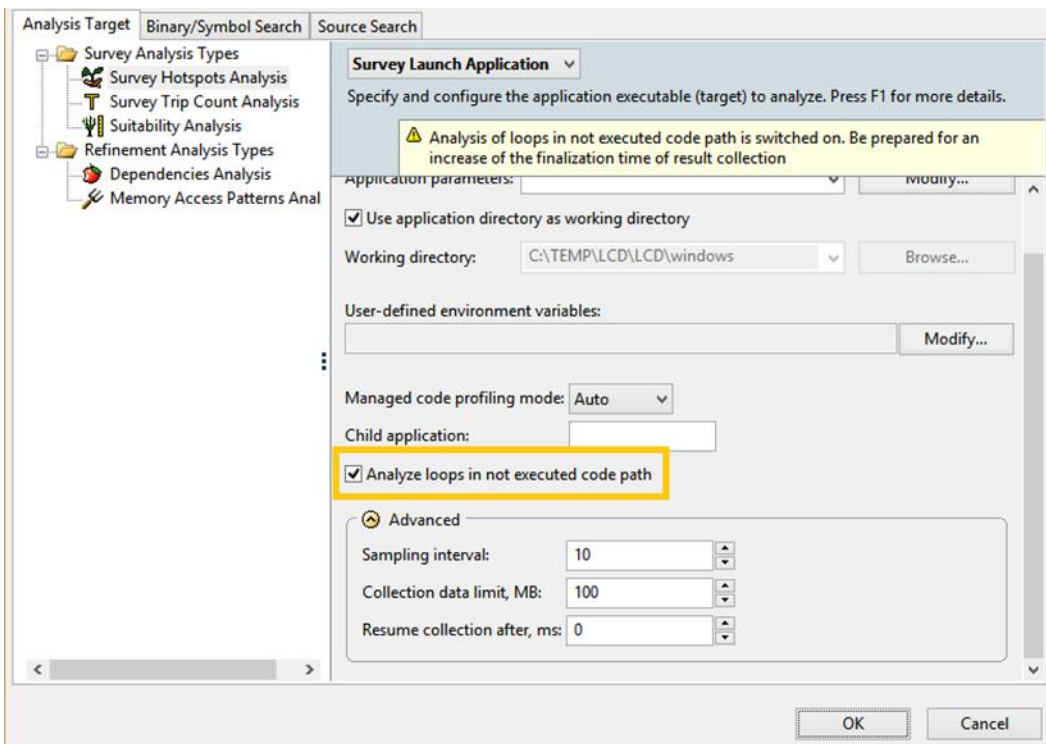
Multipath Auto-/explicit vectorisation



Use `-ax` option when compiling to create multiple paths through code

Additional paths can be added by extending the `/Qax` option e.g. :
`/QaxSSE4.2,AVX,SSE3`
(Linux: `-axSSE4.2,AVX,SSE3`)

Viewing non-executed paths



Optimization Notice

Start Tuning for AVX-512 without AVX-512 hardware

Intel® Advisor - Vectorization Advisor “axcode feature”

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

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

Compare AVX and AVX-512 code characteristics with Intel Advisor

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

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

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

Optimization Notice

Download Today!

Intel® Parallel Studio XE 2016



Vectorization is a tough problem

It is decomposable into tractable steps

Get help at each step:

- Find the best opportunities
- Improve vectorization effectiveness
- Assure correctness

Download Today

Google:

"Intel Advisor 2016"

Web-site:

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

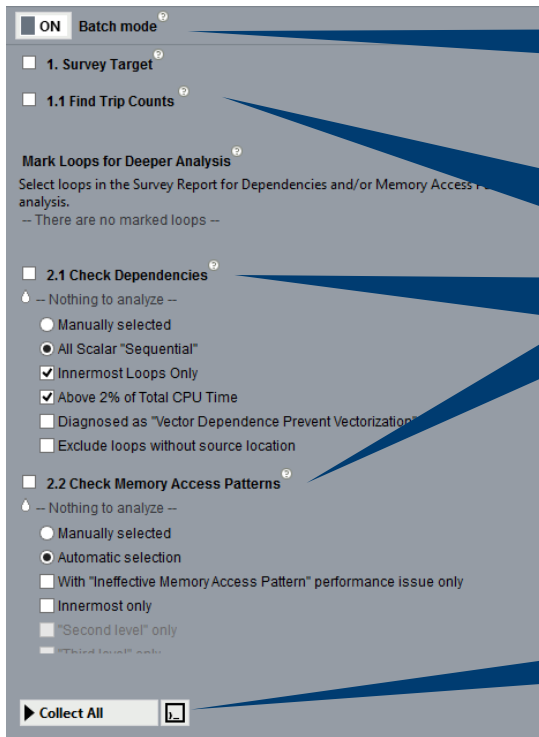
Beta 2017 Registration & Download:

<http://bit.ly/PSXE2017-Beta>

BACKUP

Batch Mode Workflow Saves Time

Intel® Advisor - Vectorization Advisor



Turn On
Batch Mode

Select
analyses to
run

Click
Collect all

Run several analyses in batch
as a single run

Contains pre-selected criteria
for advanced analyses

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

What is SIMD Data Layer Templates (SDLT)?



Courtesy Alex Wells

- A C++11 template library providing concepts of Containers, accessors, and Indexes to abstract out different aspects of creating an efficient data parallel SIMD program.
- Containers encapsulate the in memory data layout of an Array of “Plain Old Data” objects.
- SIMD loops use accessors with an array subscript operator (just like C++ arrays) to read from or write to the objects in the Containers.
- Offsets can be embedded in accessors or applied to a Index passed to the accessors array subscript operator.
- Since these concepts are abstracted out, multiple concrete versions can exist and can encapsulate best known methods, thus avoiding common pitfalls in generating efficient SIMD code.

Many C++ Programs use Array of Structures (AOS)

- It is very typical for programmers to use “Plain Old Data” objects and store them in array based containers. IE:

User's Plain Old Data object

```
struct YourStruct
{
    float x;
    float y;
    float z;
};
```

- Heap based arrays

```
YourStruct * input = new YourStruct[count];
YourStruct * result = new YourStruct[count];
```

- Stack based Variable Length Arrays

```
YourStruct input[count];
YourStruct result[count];
```

- `std::vector<>`

```
typedef std::vector<YourStruct> Container;
Container input(count);
Container result(count);
```

SOA data layout of object using SDLT

```
typedef sdlt::soa1d_container<YourStruct> Container;  
Container inputContainer(count);  
Container resultContainer(count);
```

- Intent is data be kept in an SOA or ASA Container the entire time instead of converting from AOS.
- SDLT's container will internally store the members of YourStruct in a **one dimensional "Structure of Arrays"** (SOA) layout.
 - Places aligned arrays inside a single allocated buffer vs. a separate allocation per array
- Just like `std::vector` the Containers own the array data and its scope controls the life of that data.

SDLT Primitives

- How do the Containers discover the data members of your struct?
- C++ lacks compile time reflection, so the user must **provide SDLT** with some information on **the layout of YourStruct**.
- This is easily done with the **SDLT_PRIMITIVE helper macro** that accepts a struct type followed by a list of its data members.
- A struct must be declared as a primitive before it is used as template parameter to a Container.

```
SDLT_PRIMITIVE(YourStruct, x, y, z)
```

```
typedef sdtl::soa1d_container<YourStruct> Container;  
Container inputContainer(count);  
Container resultContainer(count);
```

- If just using a built-in type like float, double, int, etc. instead of a struct
 - no need to provide a **SDLT_PRIMITIVE**

```
typedef sdtl::soa1d_container<float> Container;
```

Optimization Notice

INTEL[®] ADVISOR XE

VECTORIZATION OPTIMIZATION AND **THREAD PROTOTYPING**



BACKUP

Faster Code Faster Using Intel® Advisor

Vectorization

"Intel® Advisor's 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
Senior Software Architect
Irish Centre for High-End Computing

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

Dr. Luigi Iapichino
Scientific Computing Expert
Leibniz Supercomputing Centre

Threading

"Intel® Advisor 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 and we can use Advisor to find potential thread safety issues to help avoid problems later on."

Carlos Boneti
HPC software engineer,
Schlumberger

"Intel® Advisor 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

[More Case Studies](#)

Get Faster Code Faster! Intel® Advisor Thread Prototyping

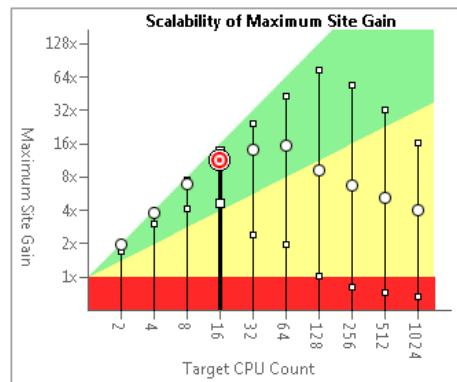
Have you:

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

Data Driven Threading Design:

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

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



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

Simon Hammond
Senior Technical Staff
Sandia National Laboratories

Design Then Implement

Intel® Advisor Thread Prototyping

Design Parallelism

- No disruption to regular development
- All test cases continue to work
- Tune and debug the design before you implement it

Implement Parallelism

1) Analyze it.

2) Design it.
(Compiler ignores these annotations.)

3) Tune it.

4) Check it.

5) Do it!

THREADING WORKFLOW

- 1. Survey Target**
Explore where to add efficient vectorization and/or threading.
▶ Collect [i] [f]
Command Line
- 1.1 Find Trip Counts**
Find how many iterations are executed.
▶ Collect [f]
Command Line
- 2. Annotate Sources**
Add Intel Advisor XE annotations to identify possible parallel tasks and their enclosing parallel sites.
▣ Steps to annotate
- 3. Check Suitability**
Analyze the annotated program to check its predicted parallel performance.
▶ Collect [i] [f]
Command Line
- 4. Check Dependencies**
Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.
▶ Collect [f]
Command Line

Less Effort, Less Risk, More Impact

Survey

Where should I add parallelism?

Summary of predicted parallel behavior

Maximum program gain[®]: 5.20x (8 CPUs, Intel TBB Threading Model)

These annotated parallel sites were detected:

Parallel Site	Maximum Site Gain [®]	Correctness Problems
solve (nqueens_annotated.cpp:113)	6.51x	2 0

Consider adding parallel site and task annotations around these time-consuming loops found during Survey an

Loop	Source Location	CPU Total Time [®]
setQueen	nqueens_annotated.cpp:96	1.8252s
solve	nqueens_annotated.cpp:117	1.8252s
setQueen	nqueens_annotated.cpp:69	0.1976s

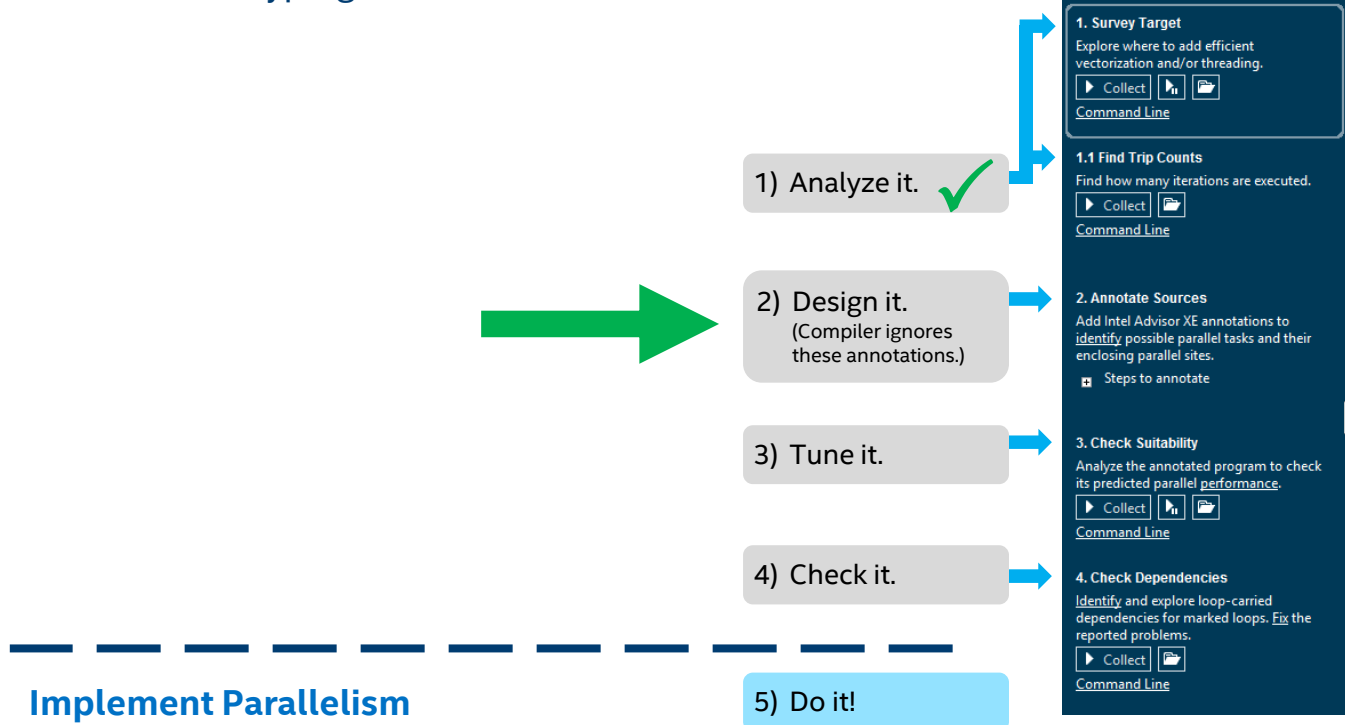
Locate where parallelism will have high impact for your application

Where should I add parallelism?

Function Call Sites and Loops	Total Time %	Total Time	Self Time	Hot Loops	Source Location
[ntdll.dll]	100.0%	2.8704s	0s		
_tmainCRTStartup	100.0%	2.8704s	0s		crtexe.c:378
solve	63.6%	1.8252s	0s		nqueens_anno
[loop at nqueens_annotated.cpp:113]	63.6%	1.8252s	0s		nqueens_anno
setQueen	63.6%	1.8252s	0s		nqueens_anno
[loop at nqueens_annotated.cpp:69]	63.6%	1.8252s	0s		nqueens_anno
setQueen	63.6%	1.8252s	0s		nqueens_anno
[loop at nqueens_annotated.cpp:96]	63.6%	1.8252s	0s		nqueens_anno

Design Then Implement

Intel® Advisor Thread Prototyping



Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Serial Modeling Has Multiple Benefits

Intel® Advisor

- 1) Your application can't fail due to bugs caused by incorrect parallel execution.
(It's running serially.)
- 2) You can easily experiment with several different proposals before committing to the expense of implementation.
 - a) Measure performance - focus on where it will pay off.
 - b) Predict scalability, load balancing and overheads.
 - c) Predict (and avoid) data races
- 3) All of your test suites should still pass.
Validate the correctness of your transformations.
- 4) You can use Advisor on partially or completely parallelized code.

Design, measure and test before implementation

Annotate Sources

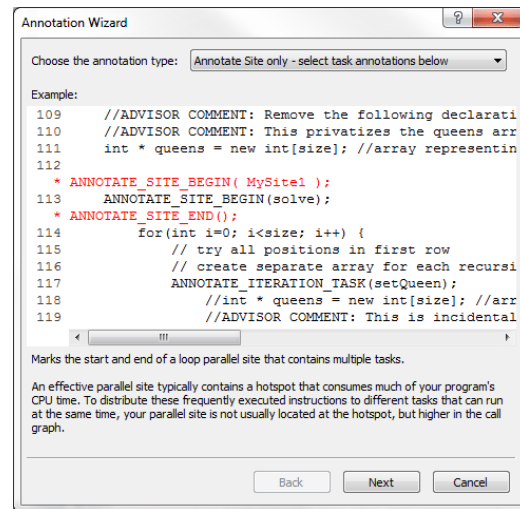
Design The Parallelism

What are annotations?

- Macros (function calls) that can be easily disabled*
- Inform analysis tool about intended parallelization

Example Intel Advisor annotations:

```
#define ANNOTATE_SITE_BEGIN(NAME)      call annotate_site_begin(NAME)
#define ANNOTATE_SITE_END(NAME)       call annotate_site_end()
#define ANNOTATE_ITERATION_TASK(NAME) call annotate_iteration_task(NAME)
#define ANNOTATE_TASK_BEGIN(NAME)     call annotate_task_begin(NAME)
#define ANNOTATE_TASK_END(NAME)       call annotate_task_end()
#define ANNOTATE_LOCK_ACQUIRE(ADDRESS) call annotate_lock_acquire(ADDRESS)
#define ANNOTATE_LOCK_RELEASE(ADDRESS) call annotate_lock_release(ADDRESS)
```



Design & Experiment Without Disrupting Development

Advisor Annotation Concepts
















Advisor uses 3 primary concepts to create a model

- **SITE** - A region of code in your application you want to transform into parallel code
- **TASK** - The region of code in a SITE you want to execute in parallel with the rest of the code in the SITE
- **LOCK** - Mark regions of code in a TASK which must be serialized

NOTE

- All of these regions may be nested
- You may create more than one SITE
- Just macros, so work with any C/C++ compiler

Annotate Sources using Survey Data

Function Call Sites and Loops	Total Time %	Total Time
▢ rt_renderscene	96.2% 	19.9664s
▢ renderscene	96.2% 	19.9662s
▢ trace_region	96.2% 	19.9662s
▢ trace_shm	96.2% 	19.9662s
▢ thread_trace	96.2% 	19.9662s
▢  parallel_thread [loop]	96.2% 	19.9662s
▢  parallel_thread [loop]	96.1% 	19.9462s
▢ parallel_thread	96.0% 	19.9262s
▢ render_one_pixel	93.1% 	19.3208s
▢ trace	79.2% 	16.4369s
▢ shader	43.6% 	9.0521s
▢  shader [loop]	35.4% 	7.3348s

**Pick out expensive loop
and add annotations**

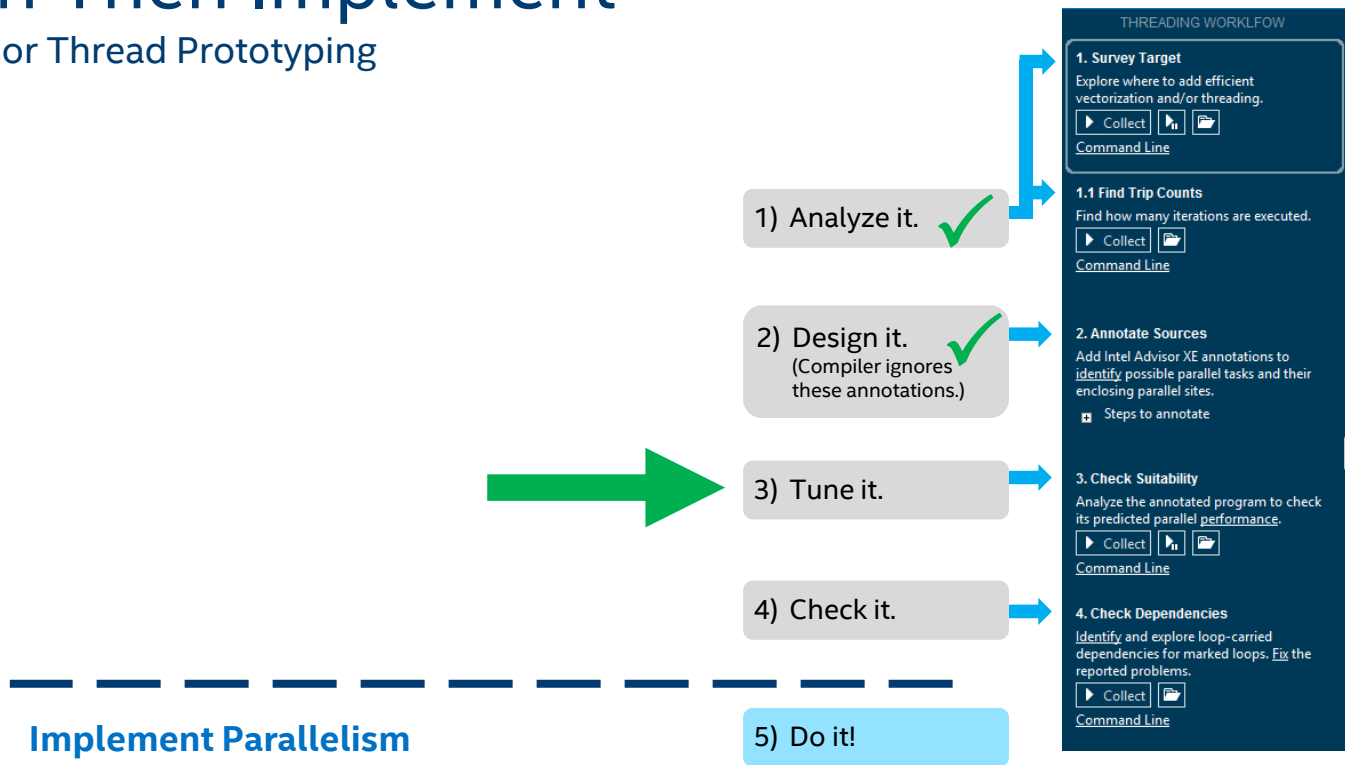
```
ANNOTATE_SITE_BEGIN(allRows);
for (int y = starty; y < stopy; y++)
{
    ANNOTATE_TASK_BEGIN(eachRow);
    m_storage.serial = 1;
    m_storage.mboxsize = sizeof(unsig
    m_storage.local_mbox = (unsigned
    memset(m_storage.local_mbox,0,m

    drawing_area drawing(startx, to
    for (int x = startx; x < stopx;
        color_t c = render_one_pixe
        drawing.put_pixel(c);
    }

    if(!video->next_frame())
    {
        free(m_storage.local_mbox);
        return;
    }
    free(m_storage.local_mbox);
    ANNOTATE_TASK_END(eachRow);
}
ANNOTATE_SITE_END(allRows);
```

Design Then Implement

Intel® Advisor Thread Prototyping



Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



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

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

What are the performance implications of the annotated sites? Intel Advisor XE 2015

Summary Survey Report Annotation Report **Suitability Report** Correctness Report

Maximum Program Gain For All Sites: 5.21x

Serial time: 1.8593s
Predicted Parallel time: 0.3570s

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

Site Performance Scalability | Site Details

Scalability of Maximum Site Gain

Maximum Site Gain vs CPU Count graph showing gain increasing from 1x at 2 CPUs to 5.21x at 8 CPUs.

Tasks Modeling

Avg. Number of Tasks: 13
Avg. Task Duration: 0.1365s

0.008x, 0.040x, 0.200x, 1x (13), 5x, 25x, 125x

0.008x, 0.040x, 0.200x, 1x (0.1365s), 5x, 25x, 125x

Apply

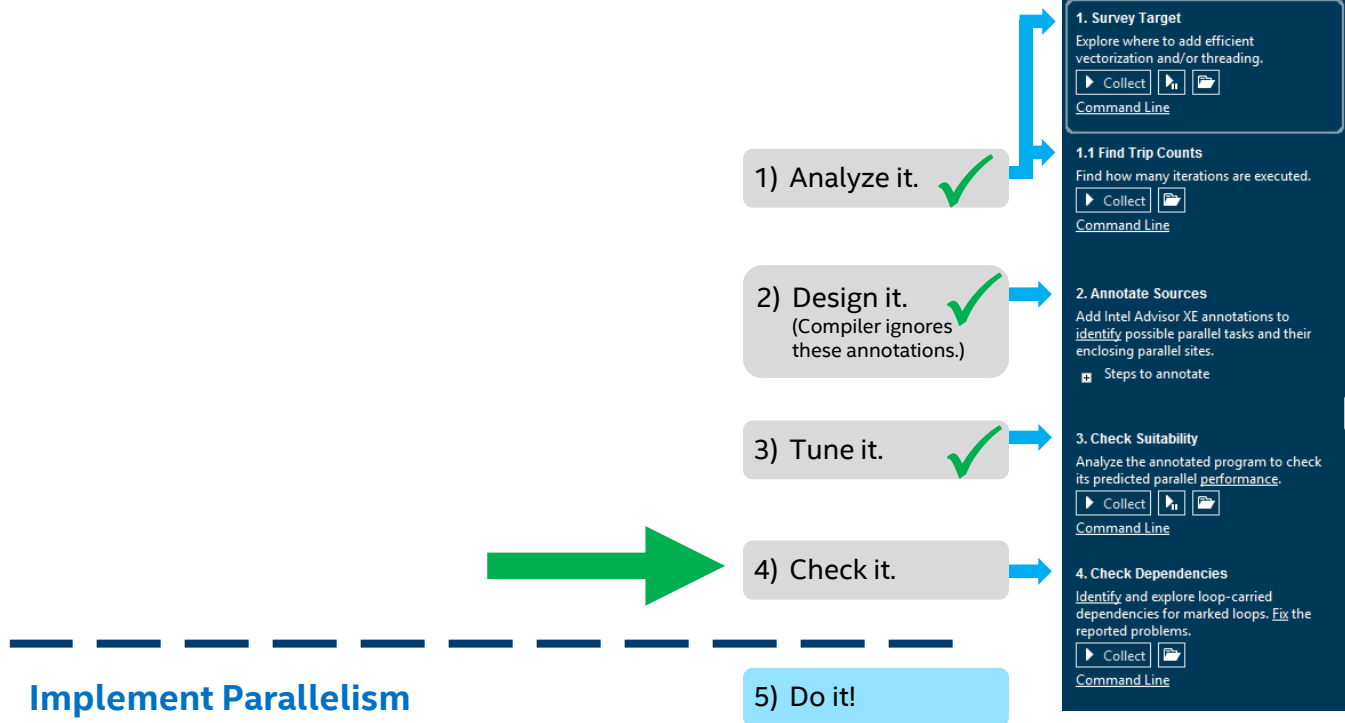
Runtime Modeling

Type of Change | Gain Benefit if Checked

- Reduce Site Overhead
- Reduce Task Overhead
- Reduce Lock Overhead
- Reduce Lock Contention
- Enable Task Chunking

Design Then Implement

Intel® Advisor Thread Prototyping



Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Check Correctness

Did the annotated tasks expose data sharing problems? Intel Advisor XE 2015

Summary Survey Report Annotation Report Suitability Report **Correctness Report**

Problems and Messages

ID	Type	Site Name	Sources	State
P1	Parallel site information	solve	nqueens_annotated.cpp	✓ Not a problem
P2	Data communication	solve	nqueens_annotated.cpp	✓ Confirmed
P3	Memory reuse	solve	nqueens_annotated.cpp	🚩 New

Data communication: Code Locations

ID	Description	Source	Function	Module	State
X2	Parallel site	nqueens_annotated...	solve	2_nqueens_annotated...	✓ Confirmed
X3	Read	nqueens_annotated...	setQueen	2_nqueens_annotated...	✓ Confirmed
X4	Write	nqueens_annotated...	setQueen	2_nqueens_annotated...	✓ Confirmed

```
111 int * queens = new int[size]; //array representing queens pla
112
113 ANNOTATE_SITE_BEGIN(solve);
114 for(int i=0; i<size; i++) {
115     // try all positions in first row
88 //ANNOTATE_LOCK_ACQUIRE(0);
89 //ADVISOR COMMENT: This is a race condition because multi
90 nrOfSolutions++; //Placed final queen, found a solution!
91 //ANNOTATE_LOCK_RELEASE(0);
92 }
```

Filter

Severity

- Error 2 items
- Remark 1 item

Type

- Parallel site informati... 1 item
- Data communication 1 item
- Memory reuse 1 item

Site Name

- solve 3 items

Source

- nqueens_annotated... 3 items

Module

- 2_nqueens_annotate... 3 items

State

- Confirmed 1 item

Sort By Item Name

Any data sharing issues?

Are they easy to fix?

Will adding locks wipe out the performance gain?

Quickly Evaluate Design Alternatives

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Repeat...

You don't have to be perfect the first time

Iterative refinement will either:

- **Create a suitable and correct design**
- **Conclude no viable design is possible**

Find out efficiently before investing in implementation

Implement Parallelism



1) Analyze it. ✓

2) Design it. ✓
(Compiler ignores these annotations.)

3) Tune it. ✓

4) Check it. ✓

5) Do it!

THREADING WORKFLOW

1. Survey Target
Explore where to add efficient vectorization and/or threading.
▶ Collect ▶ Run ▶ Save
Command Line

1.1 Find Trip Counts
Find how many iterations are executed.
▶ Collect ▶ Save
Command Line

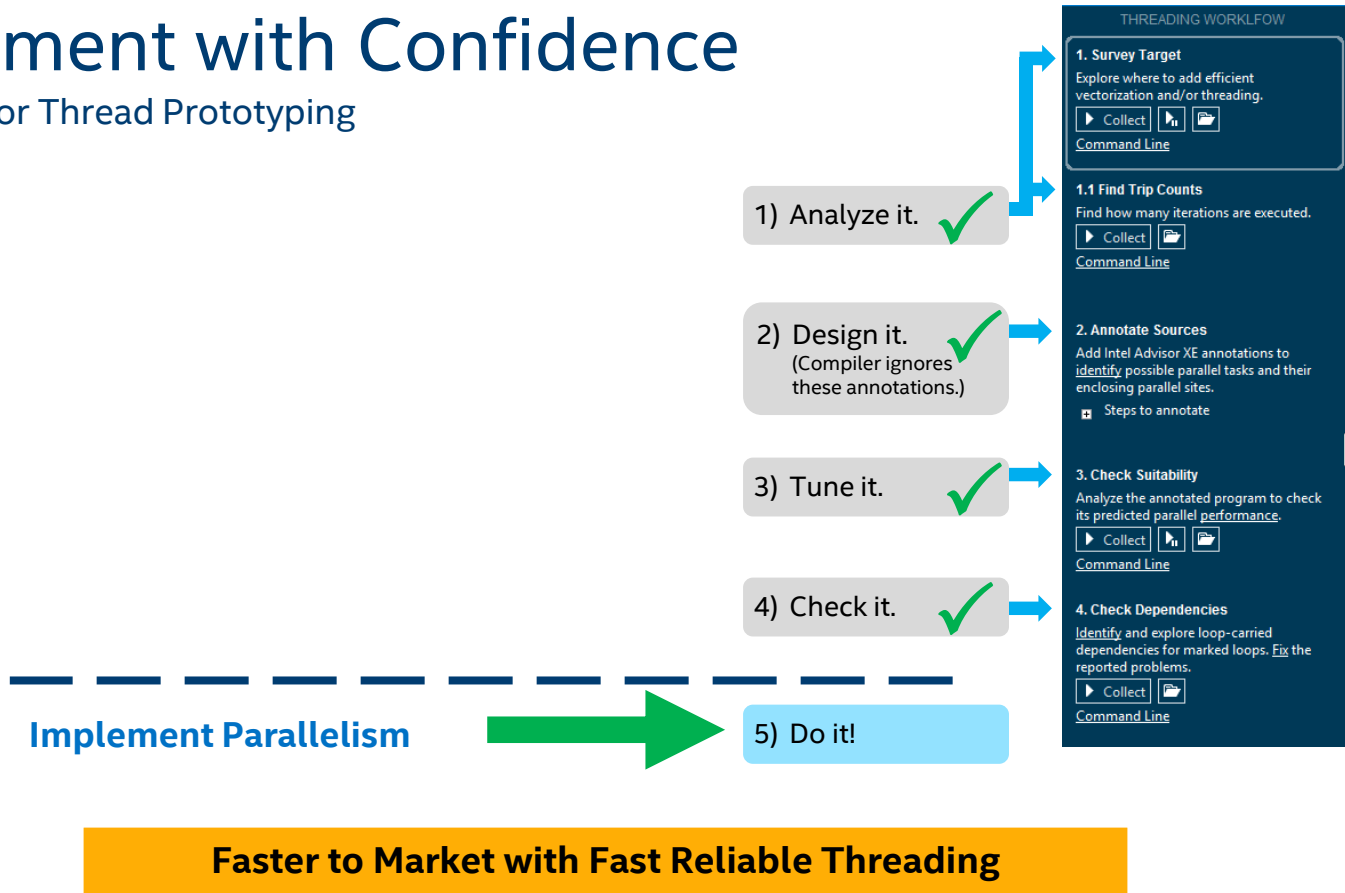
2. Annotate Sources
Add Intel Advisor XE annotations to identify possible parallel tasks and their enclosing parallel sites.
▣ Steps to annotate

3. Check Suitability
Analyze the annotated program to check its predicted parallel performance.
▶ Collect ▶ Run ▶ Save
Command Line

4. Check Dependencies
Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.
▶ Collect ▶ Save
Command Line

Implement with Confidence

Intel® Advisor Thread Prototyping



Faster to Market with Fast Reliable Threading

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Add Parallel Framework

Summary of predicted parallel behavior Intel Advisor XE 2015

Summary Survey Report Annotation Report Suitability Report Correctness Report

Maximum program gain[®]: 5.20x (8 CPUs, Intel TBB Threading Model)

These annotated parallel sites were detected:

Parallel Site	Maximum Site Gain [®]	Correctness Problems
solve (nqueens_annotated.cpp:113)	6.51x	2 0

Consider adding parallel site and task annotations around these time-consuming loops found during Survey an

Loop	Source Location	CPU Total Time [®]
setQueen	nqueens_annotated.cpp:96	1.8252s
solve	nqueens_annotated.cpp:117	1.8252s
setQueen	nqueens_annotated.cpp:69	0.1976s
setQueen	nqueens_annotated.cpp:69	0.1877s
setQueen	nqueens_annotated.cpp:69	0.1346s

Here is the list of source locations

Here are templates for popular parallel frameworks

Serial Code with Intel Advisor Annotations	Parallel Code using Intel TBB
<pre>// Locking ANNOTATE_LOCK_ACQUIRE(); Body(); ANNOTATE_LOCK_RELEASE();</pre>	<pre>// Locking can use various mutex types provided // by Intel TBB. For example: #include <tbb/tbb.h> ... tbb::mutex g_Mutex; ... { tbb::mutex::scoped_lock lock(g_Mutex); Body(); }</pre>
<pre>// Do-All Counted loops, one task ANNOTATE_SITE_BEGIN(site); For (I = 0; I < N; ++I) { ANNOTATE_ITERATION_TASK(task); statement; } ANNOTATE_SITE_END();</pre>	<pre>// Do-All Counted loops, using lambda // expressions #include <tbb/tbb.h> ... tbb::parallel_for(0, N, [&](int I) { statement; });</pre>
<pre>// Create Multiple Tasks ANNOTATE_SITE_BEGIN(site); ANNOTATE_TASK_BEGIN(task1); statement-or-task1; ANNOTATE_TASK_END(); ANNOTATE_TASK_BEGIN(task2); statement-or-task2; ANNOTATE_TASK_END(); ANNOTATE_SITE_END();</pre>	<pre>// Create Multiple tasks, using lambda // expressions #include <tbb/tbb.h> ... tbb::parallel_invoke([&]{statement-or-task1;}, [&]{statement-or-task2;});</pre>

Threading Model:

- Intel TBB
- Other
- Intel Cilk Plus
- OpenMP
- Microsoft TPL

Intel® Advisor

- Contains overhead metrics for popular parallel frameworks
- Quickly prototype and evaluate alternatives
- Detailed help pages for popular parallel frameworks

Optimization Notice

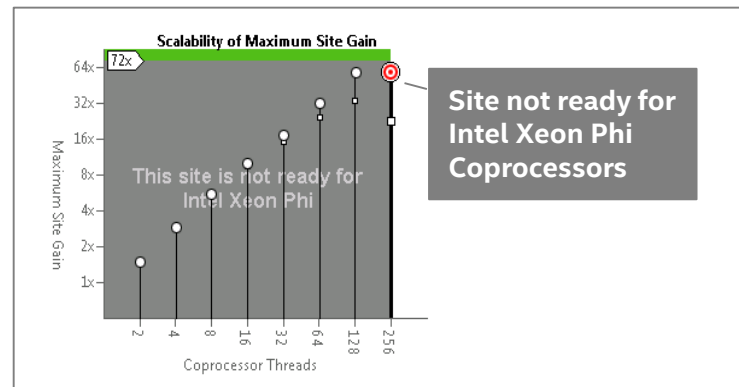
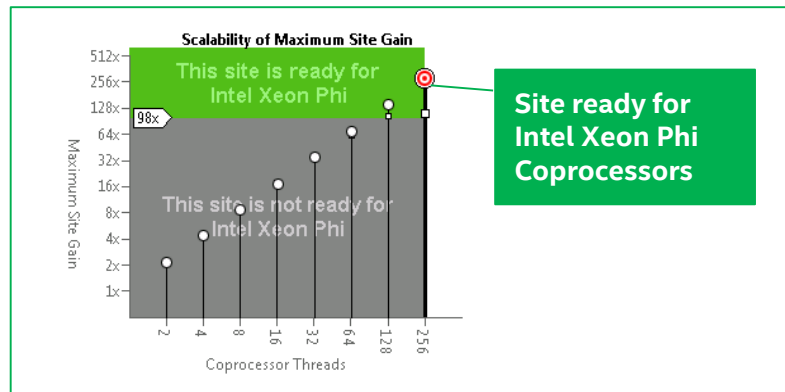
Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Target Intel® Xeon Phi™ coprocessors

Threading design and prototyping for software architects

- New Target Platforms option – See modeling based on:
- Intel® Xeon® processors or
- Intel® Xeon Phi™ coprocessors



Make better design decisions with more confidence

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Faster Code Faster with Data Driven Design

Intel® Advisor – Vectorization Optimization and Thread Prototyping

Faster Vectorization Optimization:

- Vectorize where it will pay off most
- Quickly ID what is blocking vectorization
- Tips for effective vectorization
- Safely force compiler vectorization
- Optimize memory stride

Breakthrough for Threading Design:

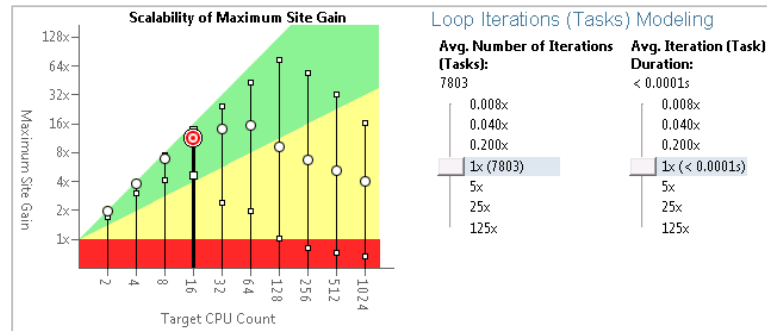
- Quickly prototype multiple options
- Project scaling on larger systems
- Find synchronization errors before implementing threading
- Design without disrupting development



Where should I add vectorization and/or threading parallelism?

Intel Advisor XE 2016

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops
[loop at stl_algo.h:4740 in std:itr...		0.170s	0.170s	1	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%
[loop at loopstl.cpp:2449 in s...		0.150s	0.150s	12	Vectorized (B		AVX
[loop at loopstl.cpp:2449 in s...		0.020s	0.020s	4	Remainder		
[loop at loopstl.cpp:7900 in vas_]		0.170s	0.170s	500	Scalar	vectorization possible but...	
[loop at loopstl.cpp:3509 in s2...	1 High vector register ...	0.160s	0.160s	12	Expand	Expand	AVX 60%



Add Parallelism with Less Effort, Less Risk and More Impact

Part of Intel® Parallel Studio
<http://intel.ly/advisor-xe>

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



Courtesy Rakesh K

Intel® Parallel Studio XE

Faster code faster!

Vectorizing Compiler

Squeeze all the performance out of the latest instruction set

Threaded Performance Libraries

Pre-vectorized, pre-threaded, pre-optimized

High Level Parallel Models

Productive solutions for thread, process & vector parallelism

Parallel Performance Profilers

Quickly discover bottlenecks and tune for high performance

Thread Debugger

Find and debug non-deterministic threading errors

Vectorization Optimization and Thread Prototyping

Data driven design tools help you vectorize & thread effectively



Download Today

Google:

“Intel Parallel Studio 2016”

Or go directly to:

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

Additional Resources

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

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® Composer XE User and Reference Guides:

https://software.intel.com/compiler_15.0_ug_c

https://software.intel.com/compiler_15.0_ug_f

Compiler User Forums: <http://software.intel.com/forums>

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

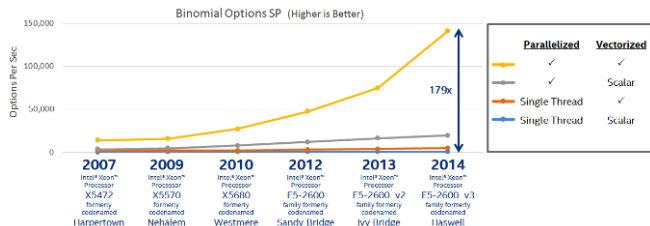
Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Configurations for Binomial Options SP



Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Performance measured in Intel Labs by Intel employees

Platform Hardware and Software Configuration

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

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



