

A series of bright blue, glowing light trails that sweep across the upper half of the slide, converging towards the right. In the background, there is a faint, semi-transparent binary code pattern (0s and 1s).

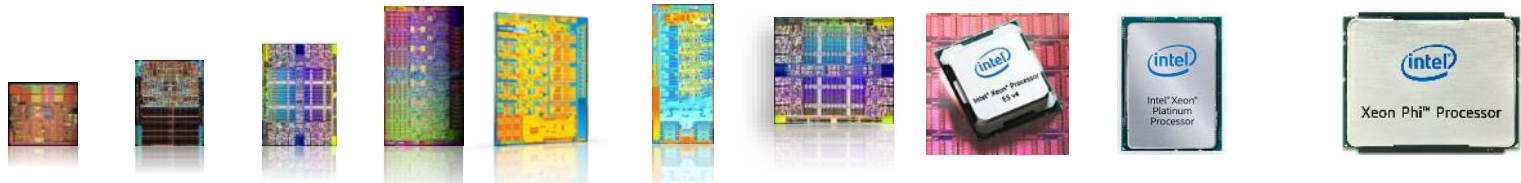
INTEL[®] ADVISOR

Vectorization Optimization

Klaus-Dieter Oertel
Intel IAGS
HLRN User Workshop, 3-6 Nov 2020

Changing Hardware Impacts Software

More Cores → More Threads → Wider Vectors



Intel® Xeon® Processor

	64-bit	5100 series	5500 series	5600 series	E5-2600	E5-2600 V2	E5-2600 V3	E5-2600 V4	Platinum 8180
Core(s)	1	2	4	6	8	12	18	22	28
Threads	2	2	8	12	16	24	36	44	56
SIMD Width	128	128	128	128	256	256	256	256	512

Intel® Xeon Phi™

processor Knights Landing
72
288
512

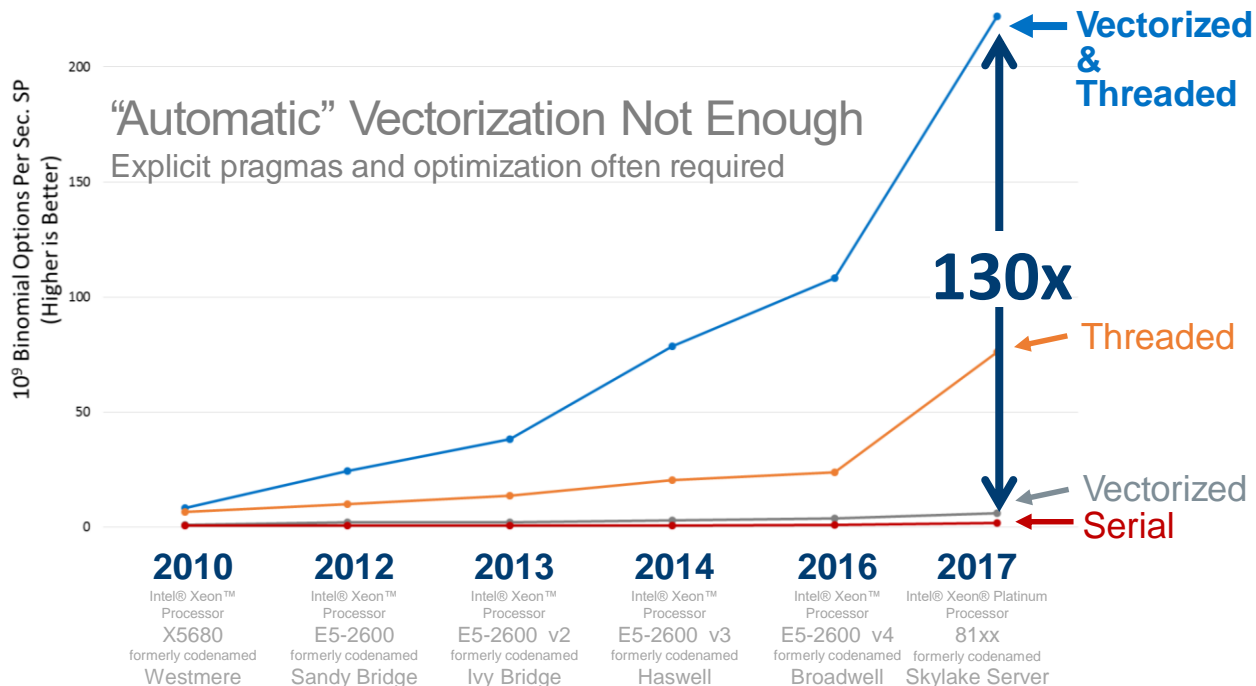
High performance software must be both

- Parallel (multi-thread, multi-process)
- Vectorized

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

Vectorize & Thread or Performance Dies

Threaded + Vectorized can be much faster than either one alone



The Difference Is Growing With Each New Generation of Hardware

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. For more information go to <http://www.intel.com/performance>

[Configurations for 2007-2016 Benchmarks](#)
at the end of this presentation

Optimization Notice

Copyright © 2020, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



ADVISOR OVERVIEW

Faster Code Faster with Data Driven Design

Intel® Advisor – Vectorization Optimization and Thread Prototyping

Faster Vectorization Optimization:

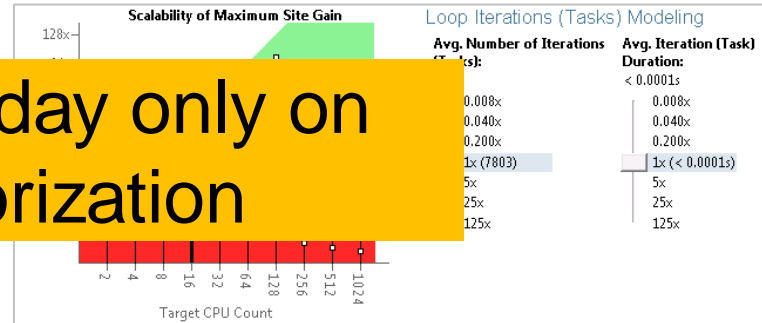
- Vectorize where it will pay off most
- Quickly ID what is blocking vectorization
- Tips for effective vectorization
- Safely force compiler vectorization
- Optimize memory stride

Function Call Sites and Loop	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops
[loop at stl_algo.h:4740 i...		0.170s	0.170s		Scalar	non-vectorizable l...	
[loop at loopstl.cpp:2449...	2 Ineffective peeled...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX 100%
[loop at loopstl.cpp:2...		0.150s	0.150s	12	Vectorized (B		AVX
[loop at loopstl.cpp:2...		0.020s	0.020s	4	Remainder		
[loop at loopstl.cpp:7902...		0.170s	0.170s	500	Scalar	vectorization possi...	
[loop at loopstl.cpp:35...	1 High vector regi...	0.160s	0.160s	12	Expand	Expand	AVX 69%

Breakthrough for Threading Design:

- Quickly prototype multiple
- Project scaling on larger
- Find synchronization errors
- Design without disrupting development

Focus today only on
vectorization



Less Effort, Less Risk and More Impact

Part of Intel® Parallel Studio for Windows* and Linux*

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

Intel® Advisor – Vectorization Advisor

Get breakthrough vectorization performance

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

The data and guidance you need:

- Compiler diagnostics + Performance Data + SIMD efficiency
- Detect problems & recommend fixes
- Loop-Carried Dependency Analysis
- Memory Access Patterns Analysis

The screenshot shows the Intel Advisor 2018 Vectorization Advisor interface. At the top, it displays 'Elapsed time: 70.29s' and buttons for 'Vectorized' and 'Not Vectorized'. Below this are filter options: 'All Modules', 'All Sources', 'Loops And Functions', and 'All Threads'. The main table is titled 'Function Call Sites and Loops' and contains the following data:

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Type	FLOPS		Why No Vectorization?	Vectorized Loops			Trip Counts	
					GFLOPS	AI		Vector...	Efficiency	Gain...		VL ..
[loop in S252 at loops90.f:1172]	1 Possible...	3.129s 7.0%	3.129s	Vectorized ...	0.1911	0.115	1 vectorizat...	AVX2	17%	1.36x	4; 8	99; 6; 1; 1
[loop in S2101 at loops90.f:1749]	2 Possible...	2.765s 6.2%	2.765s	Scalar	0.1421	0.067	vectorizatio...					12
[loop in s442_Somp\$parallel_for ...]	1 Ineffecti...	1.492s 3.4%	1.492s	Vectorized+ ...	0.5861	0.165		AVX2	14%	1.09x	8	30; 1; 3
f_svm1_sin8_i9		1.108s 2.5%	1.108s	Vector Funct...	3.9111	0.156		AVX2				
[loop in S353 at loops90.f:2381]	1 Possible...	0.989s 2.2%	0.989s	Vectorized (...)	2.0231	0.134		AVX2	27%	2.16x	8	6; 4; 1

Optimize for AVX-512 with or without access to AVX-512 hardware

Part of Intel® Parallel Studio XE

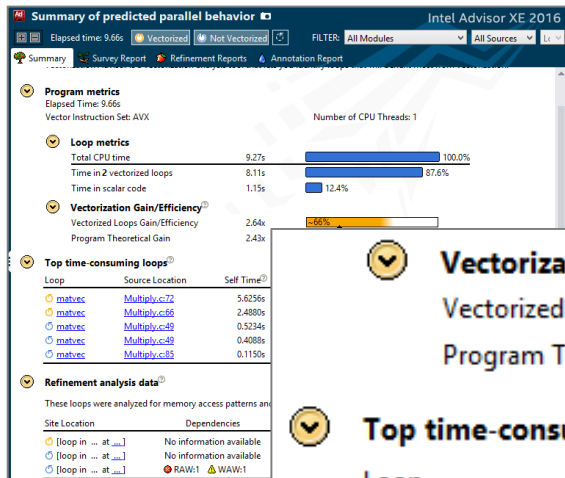
<http://intel.ly/advisor-xe>

Optimization Notice

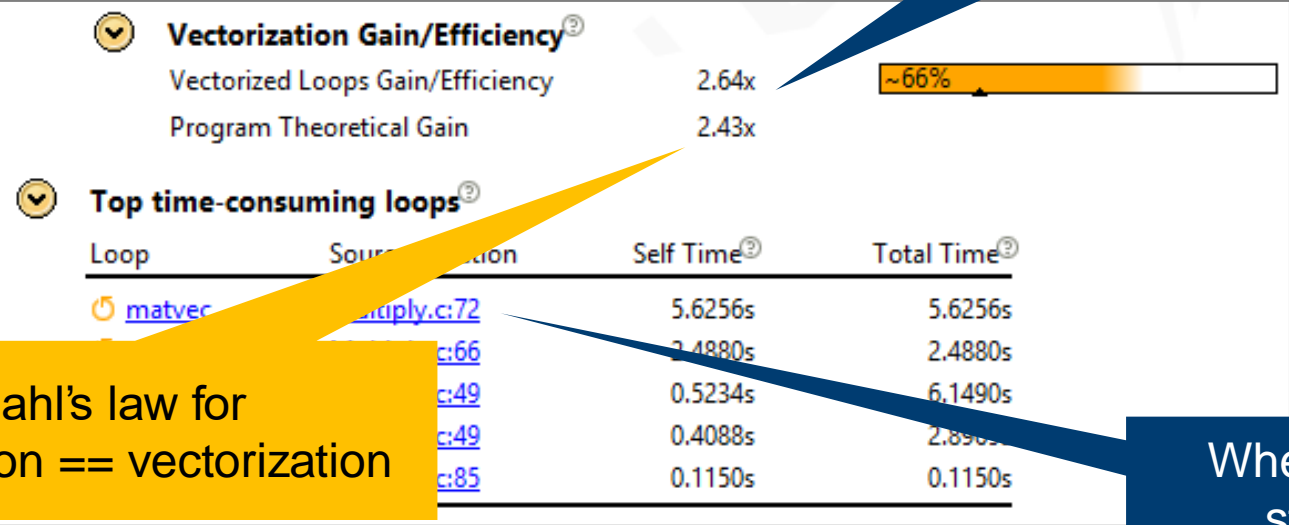
Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Summary View: Plan Your Next Steps



What can I expect to gain?



Amdahl's law for parallelization == vectorization

Where do I start?

Amdahl's law

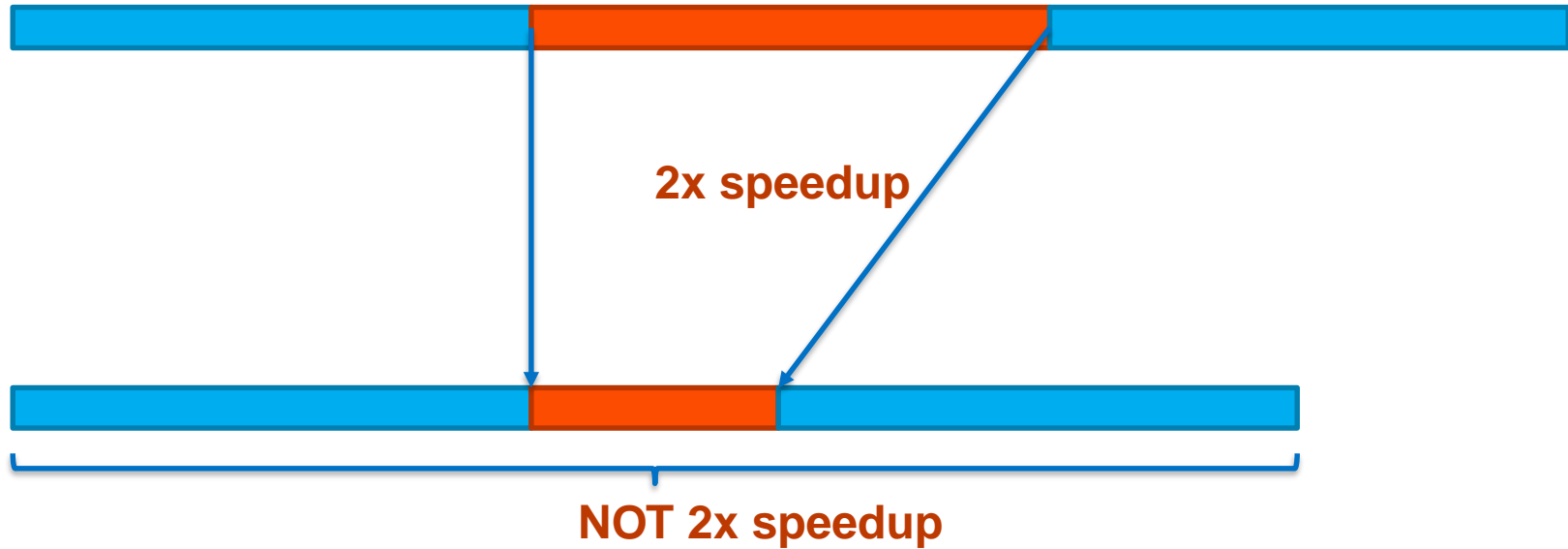
$$S_{total} = \frac{100\%}{(100\% - p) + \frac{p}{s_p}}$$

S = speedup (in parallelized part or total)

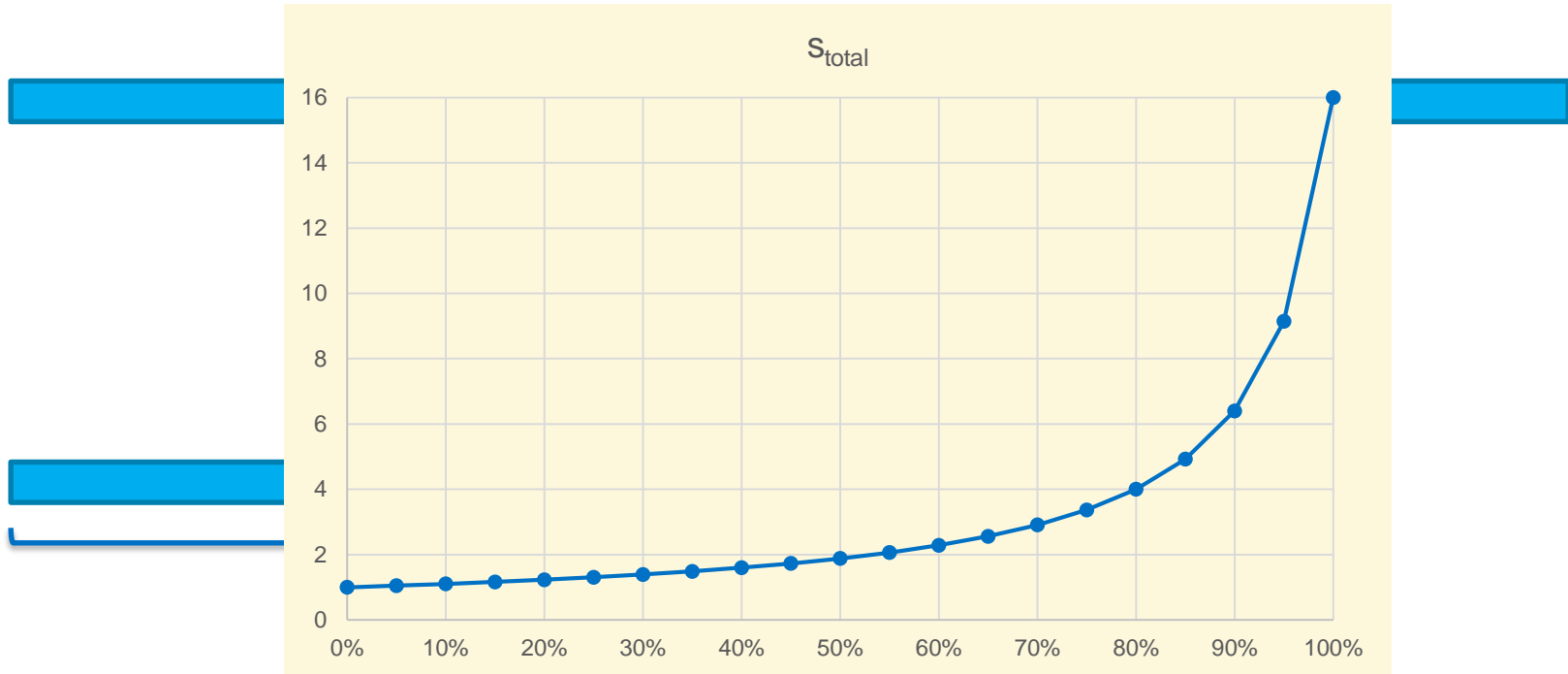
P = proportion of execution time that benefits from parallelization

Example: P=80%, $s_p=16$ [AVX-512] $\Rightarrow S_{total}=4$

Amdahl's law



Amdahl's law



The Right Data At Your Fingertips

Get all the data you need for high impact vectorization

Filter by which loops are vectorized!

Trip Counts

What prevents vectorization?

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

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

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

Focus on hot loops

What vectorization issues do I have?

Which Vector instructions are being used?

How efficient is the code?

Get Faster Code Faster!

THE ROOFLINE MODEL

WHAT IS THE ROOFLINE MODEL ?

Do you know how fast you should run ?

Comes from Berkeley

Performance is limited by equations/implementation & code generation/hardware

2 hardware limitations

- PEAK Flops
- PEAK Bandwidth

The application performance is bounded by hardware specifications

$$\mathbf{Gflop/s} = \min \left\{ \begin{array}{l} \mathbf{Platform\ PEAK} \\ \mathbf{Platform\ BW * AI} \end{array} \right.$$

Arithmetic Intensity
(Flops/Bytes)



PLATFORM PEAK FLOPS

How many floating point operations per second

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \text{Platform PEAK} \\ \text{Platform BW} * AI \end{array} \right.$$

Theoretical value can be computed by specification

Example with 2 sockets Intel® Xeon® Processor E5-2697 v2

$$\text{PEAK FLOP} = 2 \times 2.7 \times 12 \times 8 \times 2 = 1036.8 \text{ Gflop/s}$$

Number of sockets

Core Frequency

Number of cores

Number of single precision element in a SIMD register

1 port for addition, 1 for multiplication

More realistic value can be obtained by running Linpack

=~ 930 Gflop/s on a 2 sockets Intel® Xeon® Processor E5-2697 v2

PLATFORM PEAK BANDWIDTH

How many bytes can be transferred per second

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \text{Platform PEAK} \\ \text{Platform BW} * AI \end{array} \right.$$

Theoretical value can be computed by specification

Example with 2 sockets Intel® Xeon® Processor E5-2697 v2

$$\text{PEAK BW} = 2 \times 1.866 \times 8 \times 4 = 119 \text{ GB/s}$$

Number of sockets

Memory Frequency

Byte per channel

Number of mem channels

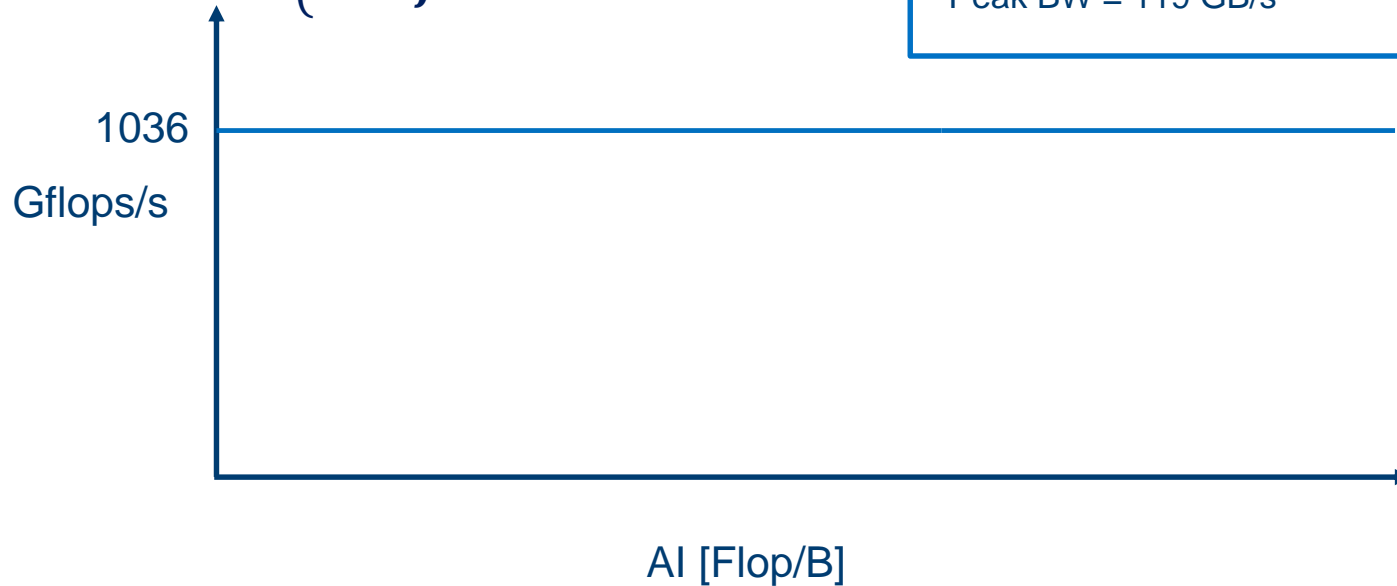
More realistic value can be obtained by running Stream

=~ 100 GB/s on a 2 sockets Intel® Xeon® Processor E5-2697 v2

DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \textit{Platform PEAK} \\ \textit{Platform BW} * \textit{AI} \end{array} \right.$$



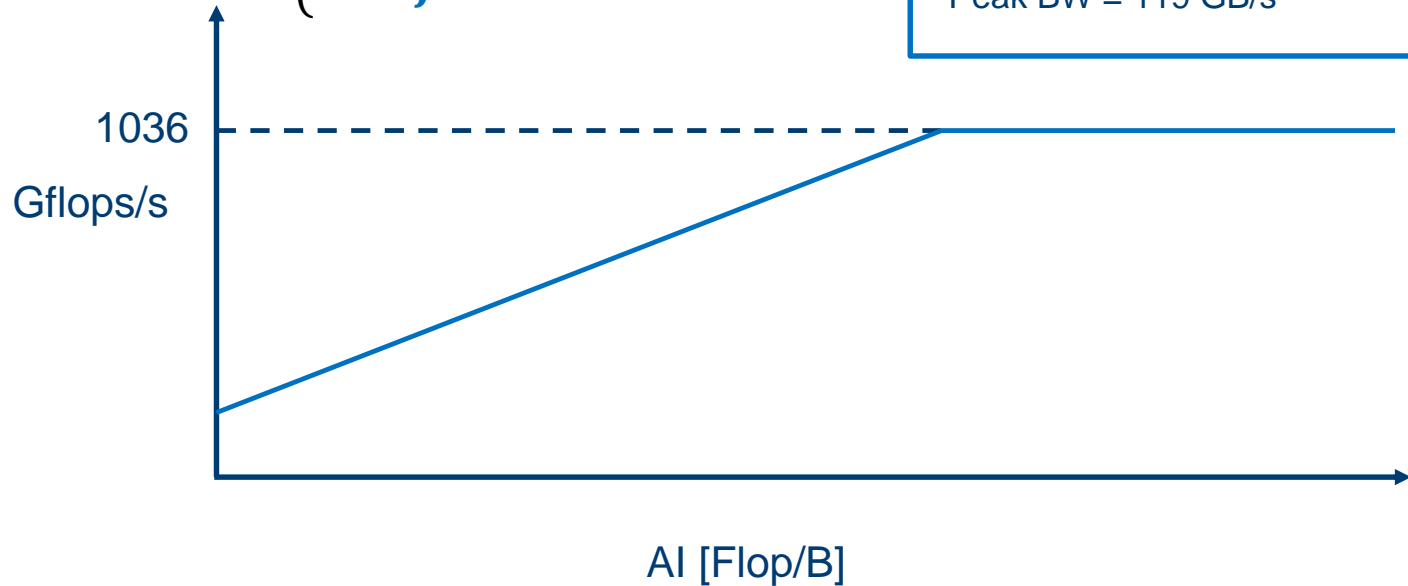
2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s

DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \text{Platform PEAK} \\ \text{Platform BW} * \text{AI} \end{array} \right.$$

2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s

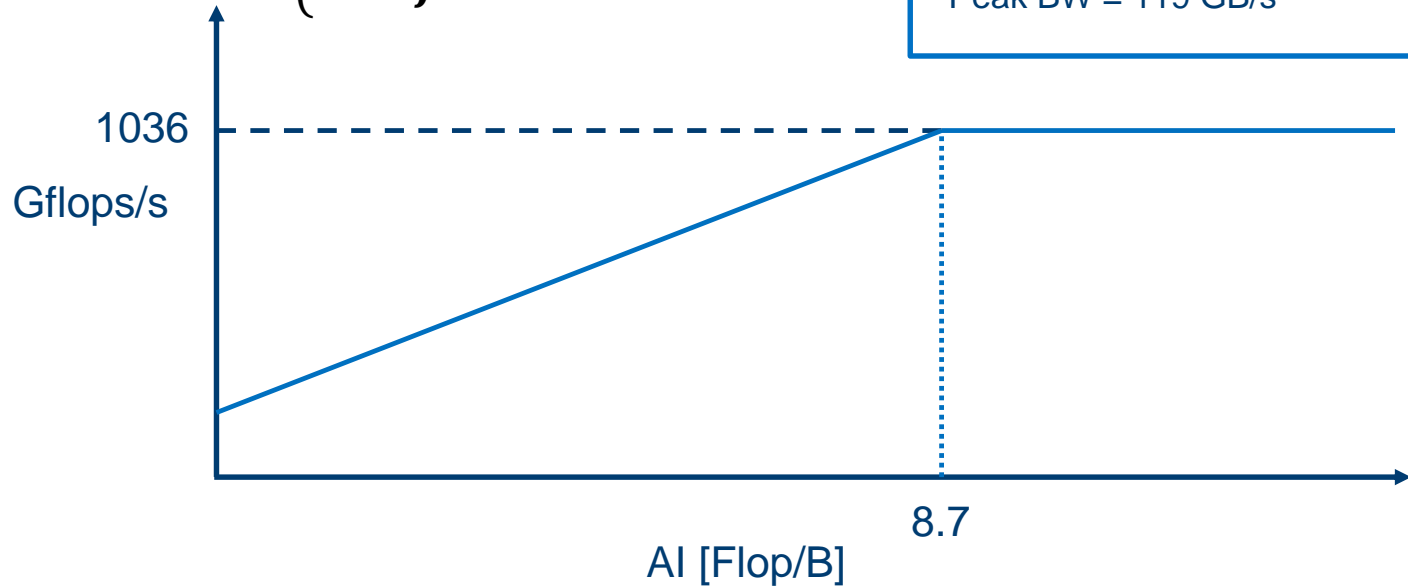


DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \text{Platform PEAK} \\ \text{Platform BW} * \text{AI} \end{array} \right.$$

2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s



WHAT IS THE PERFORMANCE BOUNDARY?

Manual way to do it

Manual counting on matrix/matrix multiplication

```
for(i=0; i<N; i++)  
  for(j=0; j<N; j++)  
    for(k=0; k<N; k++)  
      c[i][j] = c[i][j] + a[i][k] * b[k][j]
```

add = $N * N * N$

#Read = $3 * N * N * 4$ bytes

mul = $N * N * N$

#Write = $N * N * 4$ bytes

$$AI = \frac{2N^3}{16N^2} = \frac{1}{8}N$$

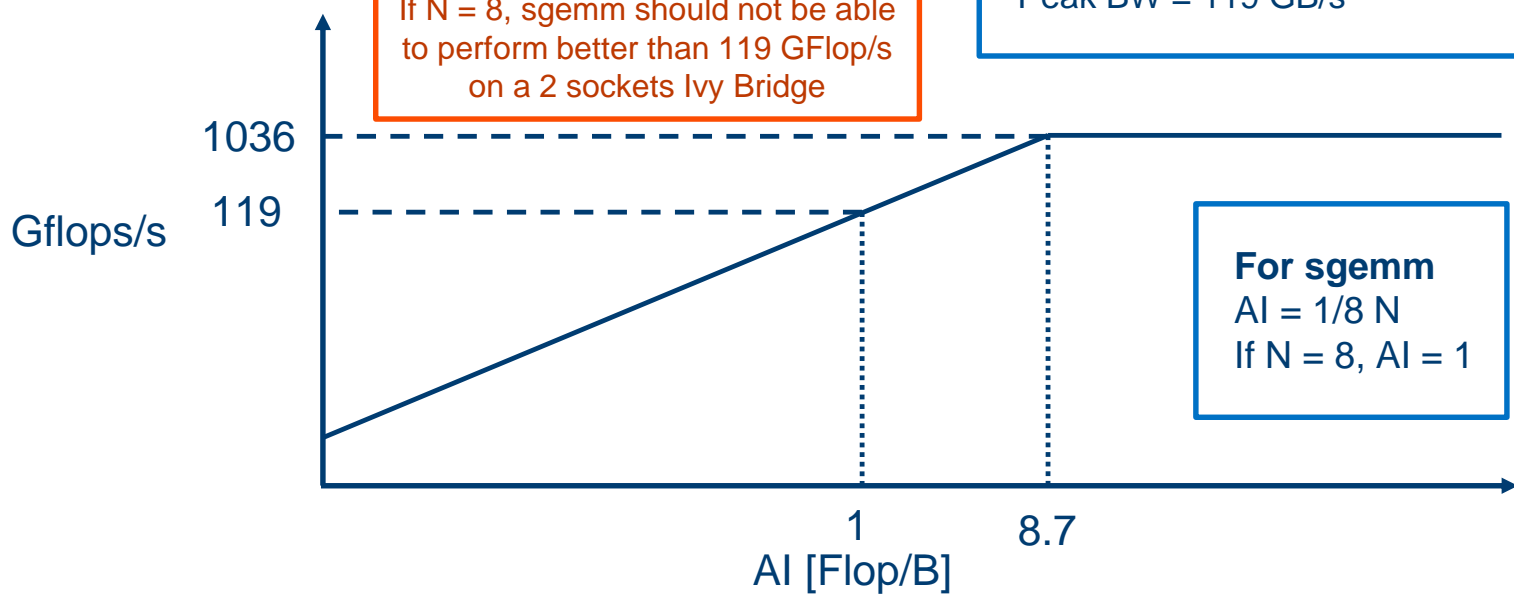
COMPUTE THE MAXIMUM PERFORMANCE

BW * Arithmetic Intensity

$$\text{Gflop/s} = \min \left\{ \begin{array}{l} \text{Platform PEAK} \\ \text{Platform BW} * \text{AI} \end{array} \right.$$

If N = 8, sgemm should not be able to perform better than 119 GFlop/s on a 2 sockets Ivy Bridge

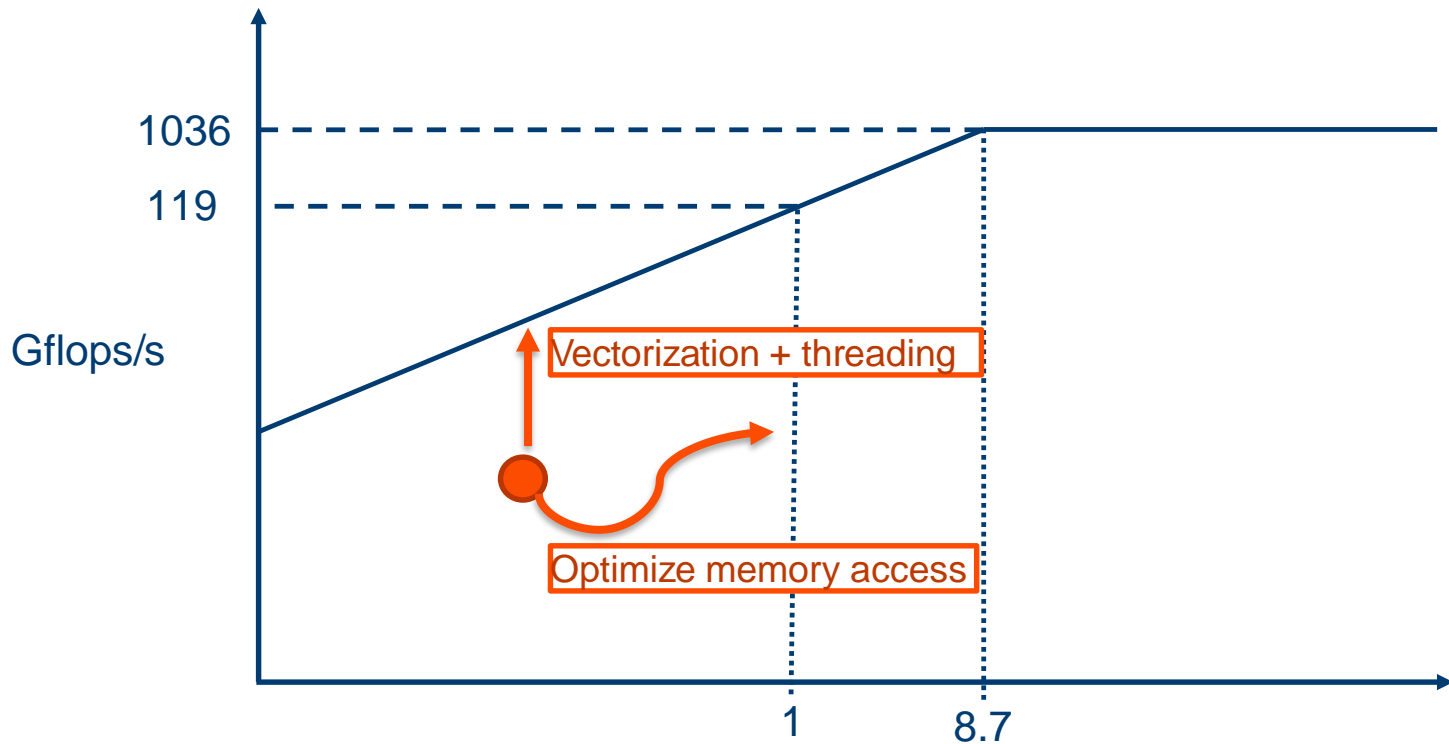
2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s



For sgemm
AI = 1/8 N
If N = 8, AI = 1

AND NOW?

How to get better performance?



Optimization Notice

Copyright © 2020, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

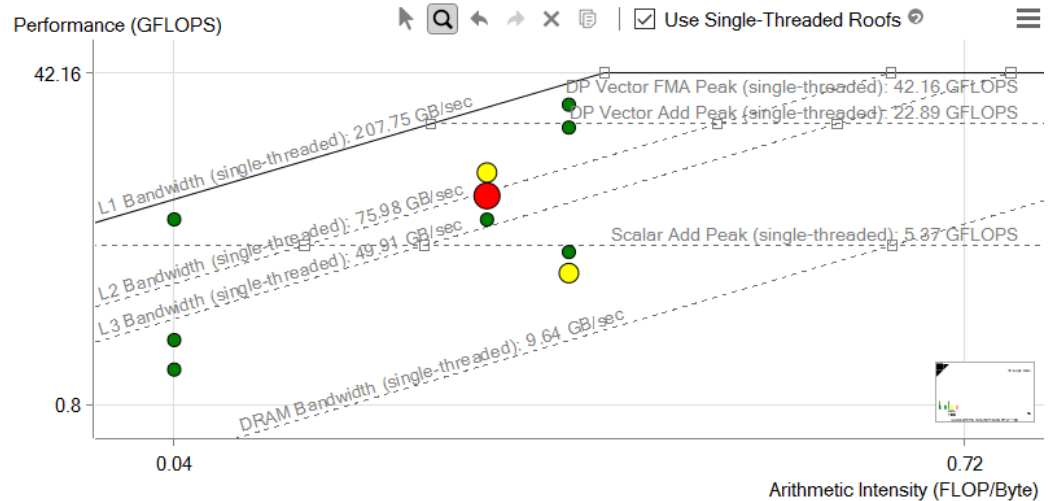


ROOFLINE IN INTEL[®] ADVISOR

What is a Roofline Chart?

A Roofline Chart plots application performance against hardware limitations.

- Where are the bottlenecks?
- How much performance is being left on the table?
- Which bottlenecks can be addressed, and which *should* be addressed?
- What's the most likely cause?
- What are the next steps?



Roofline first proposed by University of California at Berkeley:
[Roofline: An Insightful Visual Performance Model for Multicore Architectures](#), 2009
Cache-aware variant proposed by University of Lisbon:
[Cache-Aware Roofline Model: Upgrading the Loft](#), 2013

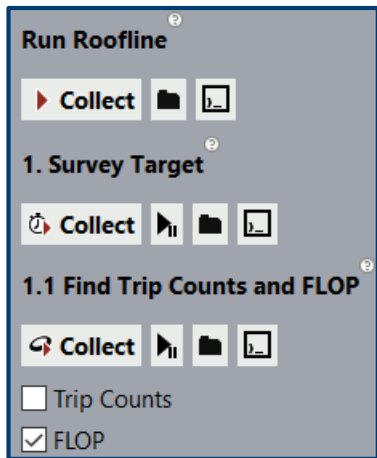
Roofline Metrics

Roofline is based on FLOPS and Arithmetic Intensity (AI).

- **FLOPS:** Floating-Point Operations / Second
- **Arithmetic Intensity:** FLOP / Byte Accessed



Collecting this information in Intel® Advisor requires two analyses.

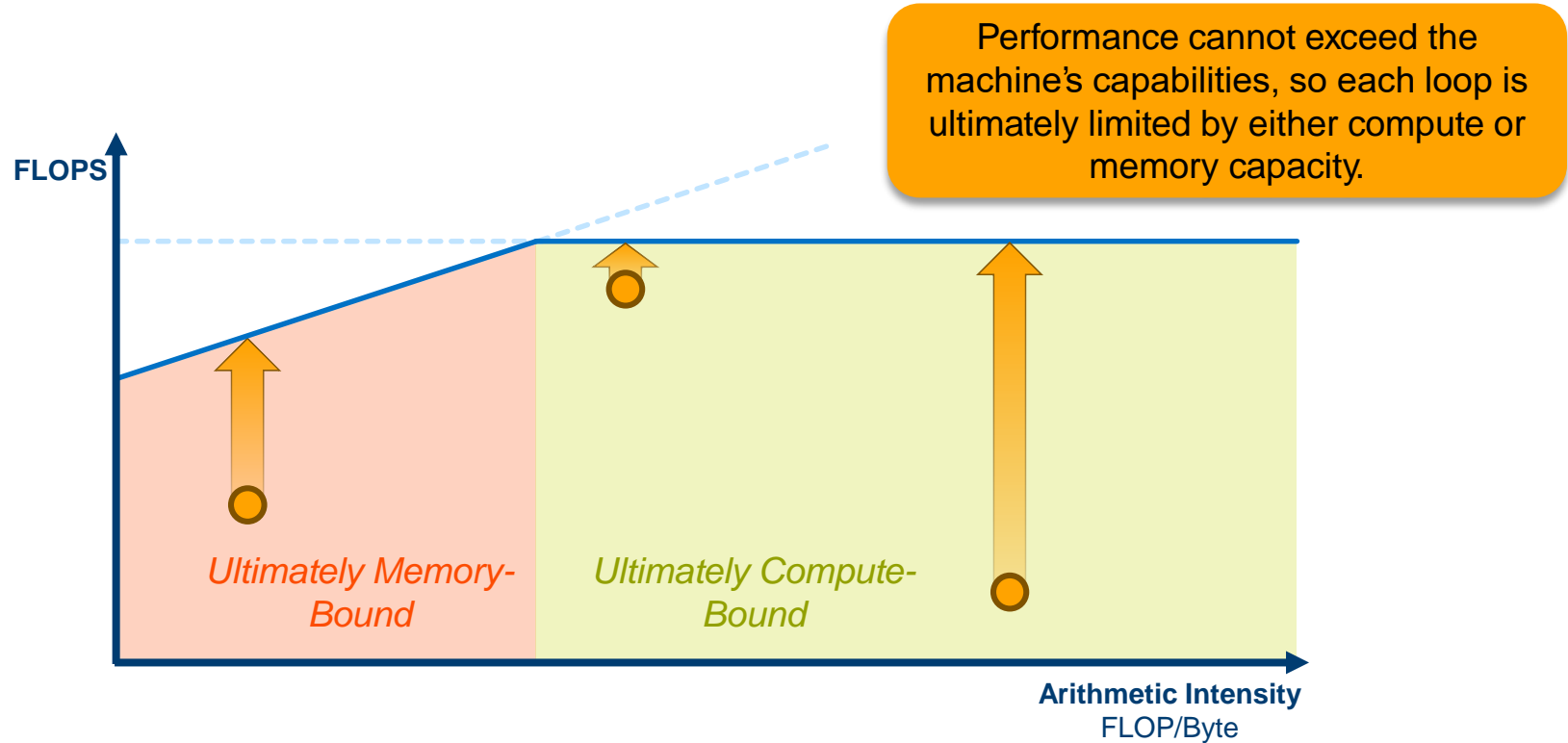


Shortcut to run Survey followed by Trip Counts + FLOPs

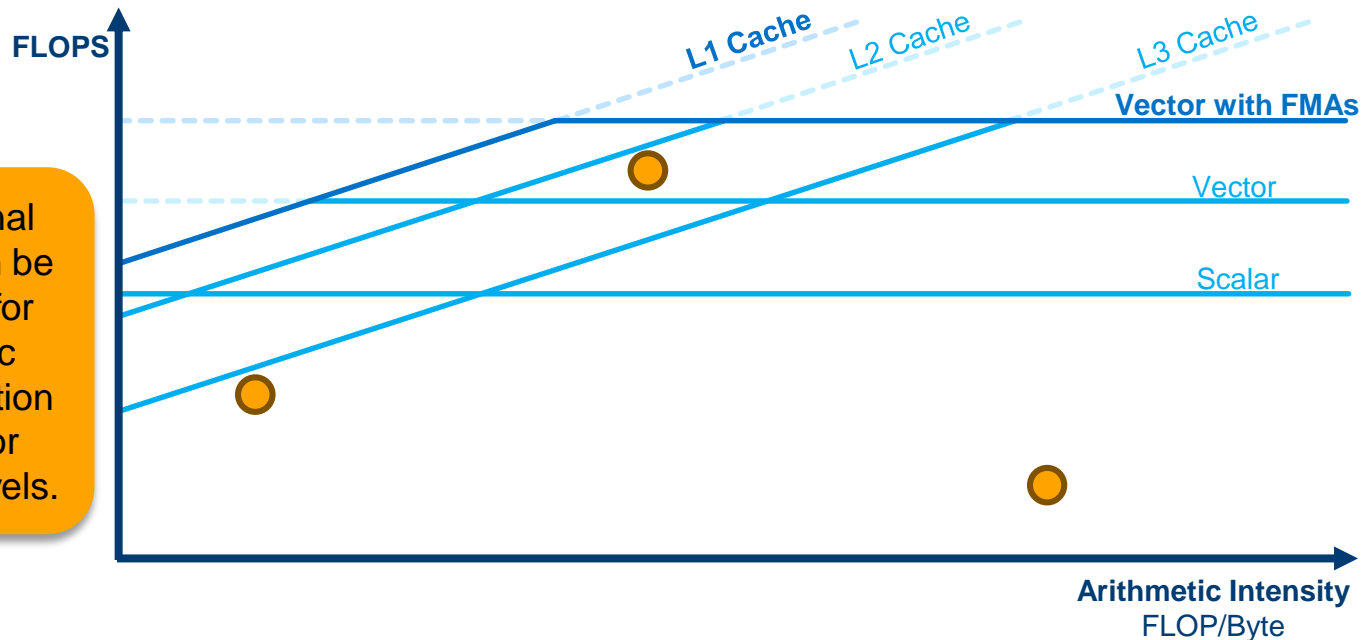
Runs system benchmarks and collects timing data.

Collects memory traffic and FLOP data.
Must be run separately due to higher overhead that would interfere with timing measurements.

Ultimate Performance Limits



Sub-Roofs and Current Limits

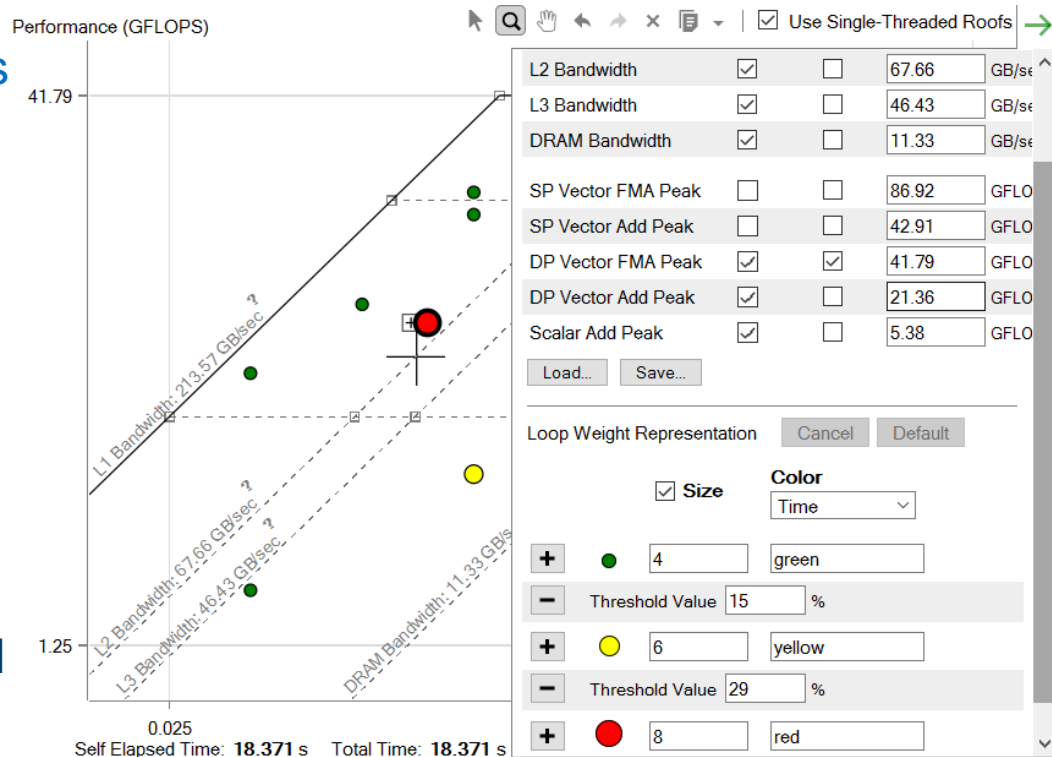


Additional roofs can be plotted for specific computation types or cache levels.

These sub-roofs can be used to help diagnose bottlenecks.

The Intel® Advisor Roofline Interface

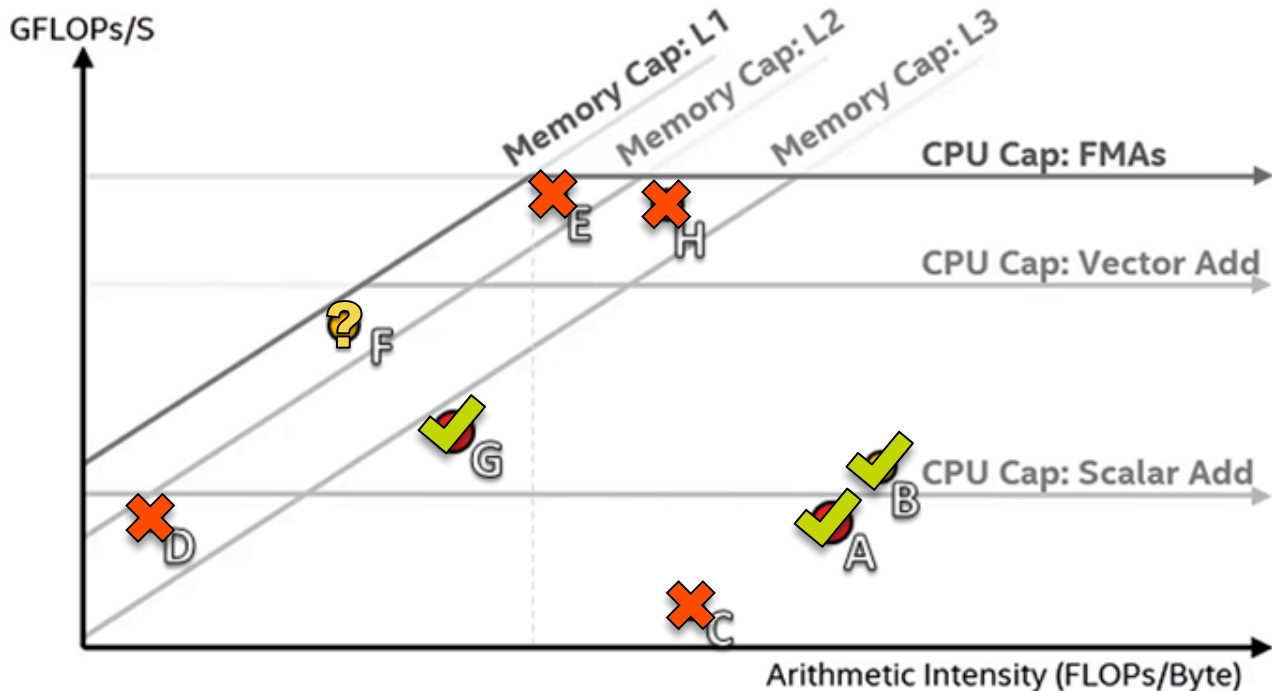
- Roofs are based on benchmarks run before the application.
- Roofs can be hidden, highlighted, or adjusted.
- Intel® Advisor has size- and color-coding for dots.
- Color code by duration or vectorization status
- Categories, cutoffs, and visual style can be modified.



Identifying Good Optimization Candidates

Focus optimization effort where it makes the most difference.

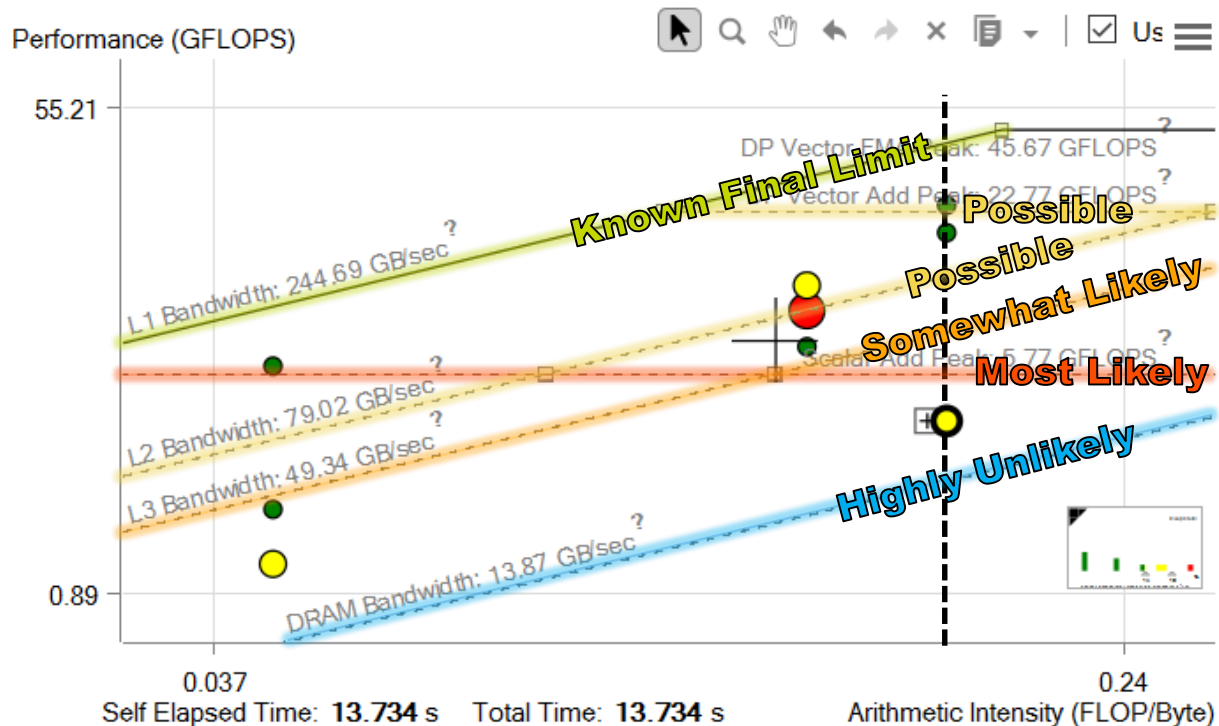
- Large, red loops have the most impact.
- Loops far from the upper roofs have more room to improve.



Identifying Potential Bottlenecks

Final roofs *do* apply;
sub-roofs *may* apply.

- Roofs above indicate *potential* bottlenecks
- Closer roofs are the most likely suspects
- Roofs below may contribute but are generally not primary bottlenecks

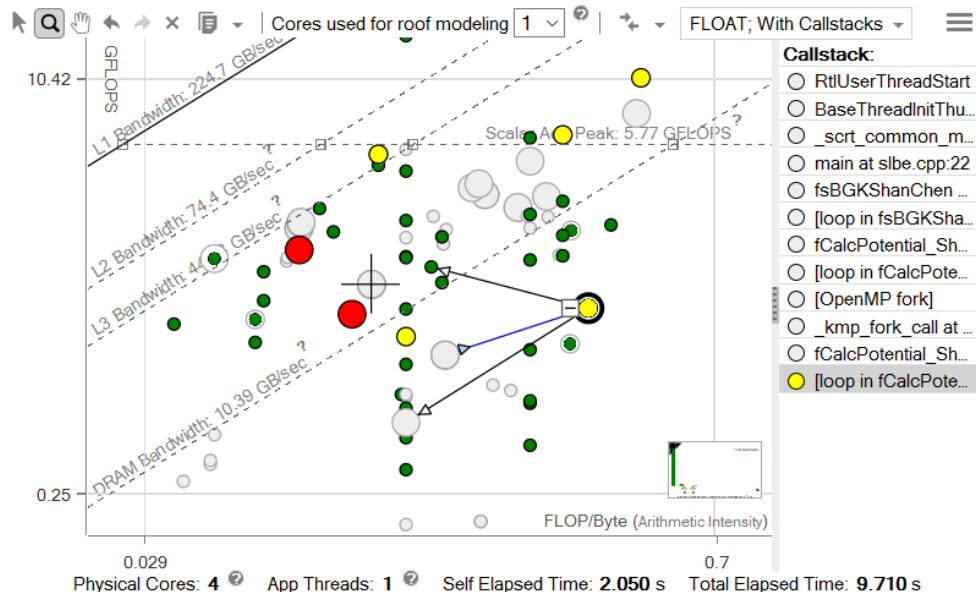


ROOFLINE WITH CALL STACKS

Roofline with Call Stacks

Advisor 2018 Update 1 added call stacks to Roofline.

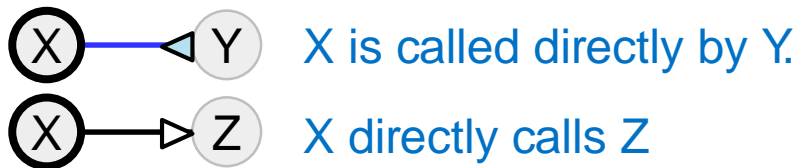
- Granularity of data can be adjusted.
- Reveals inefficiencies that originate higher in the call chain.
- More accurate view of functions or loops that behave differently under different circumstances
- Differentiates between instances with different call chains.



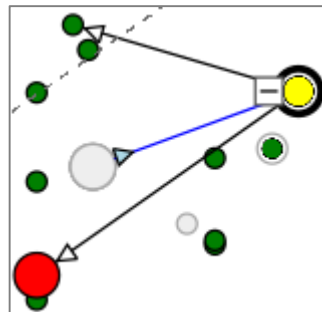
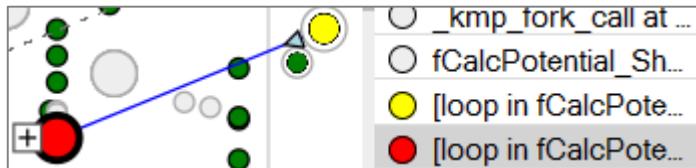
Reading the Roofline with Call Stacks

Visualizing the Call Chain

Arrows indicate relationships between dots.

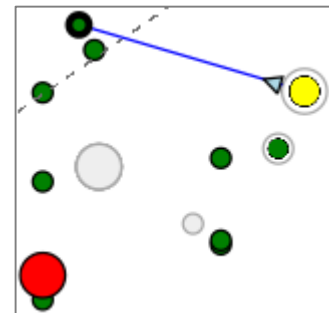


The call stack displays the call chain for the selected loop. Clicking an entry causes it to flash on the Roofline for easy identification.



The selected yellow dot was called by the gray dot, and it calls the red and green dots.

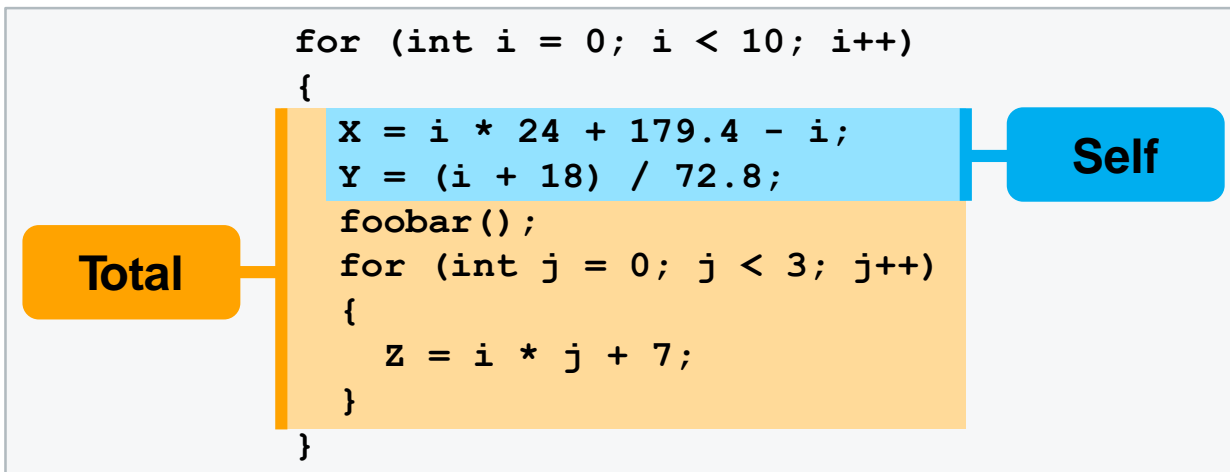
Selecting the green dot shows that it is called by the yellow dot, and doesn't call anything itself.



Self Data vs Total Data

The original Roofline used only **self data**: only work done directly is recorded.

The Roofline with call stacks uses both **self data and total data**, which includes work done in functions or loops called as well as work done directly.



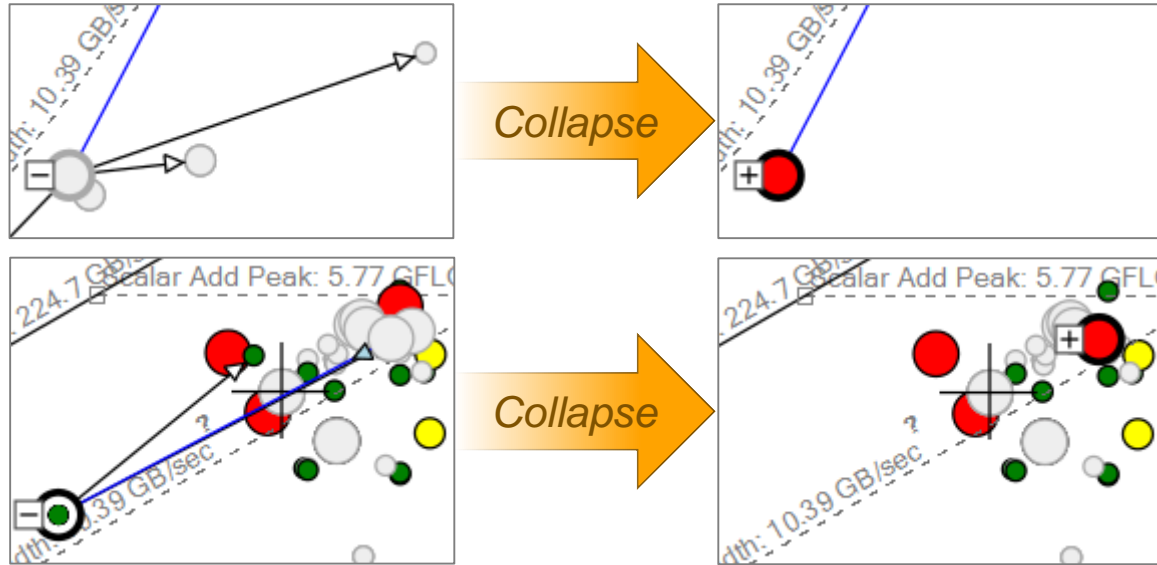
Reading the Roofline with Call Stacks

Expanding and Collapsing Outer Loops

Collapsing and expanding dots switches between self- and total-data mode.

Dots with no self data are grayed out when expanded and in color when collapsed.

Dots that have self data have the appearance and location based on it when expanded, with a halo of the size related to their total data.



When collapsed, their appearance and location changes to reflect the total data.

GUI AND COMMAND LINE

Get Roofline data using GUI

The image shows the Intel Advisor GUI with the Survey & Roofline workflow active. The 'Run Roofline' section is highlighted with a red box, showing the 'Collect' button and the 'Enable Roofline with Callstacks' checkbox. The '1. Survey Target' section is also highlighted with a red box, showing the 'Collect' button and the 'Mark Loops for Deeper Analysis' section. The '1.1 Find Trip Counts and FLOP' section is highlighted with a red box, showing the 'Collect' button and the 'Trip Counts' and 'FLOP' checkboxes. The '2.1 Check Memory Access Patterns' section is highlighted with a red box, showing the 'Collect' button. The 'Performance (GFLOPS)' graph is visible, showing a red dot representing the current configuration. The 'miniGhost - Project Properties' dialog box is open, showing the 'Analysis Target' tab with 'Survey Analysis Types' selected. The 'Launch Application' dropdown is set to 'Launch Application'. The 'Collect information about Loop Trip Counts' and 'Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage' checkboxes are checked. The 'Self FLOP Per Iteration' value is 32.

Command line created by GUI

Source	Top Level	Self FLOP	Self AI	Total AI	Total GFLOPS
Self GFLOP		0,41943	0,12500	0,12500	1,06375
Self AI		0,12500	0,12500	0,12500	0,41943
Total AI		0,12500	0,12500	0,12500	0,41943
Self GFLOP		0,41943	0,12500	0,12500	0,41943
Total GFLOP		0,41943	0,12500	0,12500	0,41943
Self FLOP Per Iteration		32			

Get roffline data using **command line**. Example:

```
> source advixe-vars.sh
```

1st method:

```
> advixe-cl -collect roffline -project-dir ./your_project -- <your-executable-with-parameters>
```

2nd method (more flexible):

```
> advixe-cl -collect survey -project-dir ./your_project -- <your-executable-with-parameters>
```

```
> advixe-cl -collect tripcounts -flop -project-dir ./your_project -- <your-executable-with-parameters>
```

```
> advixe-gui ./your_project
```

Running Intel Advisor XE on a cluster

Example: Collect from middle rank of 3x3x3 cube of processes:

```
mpirun -n 27 advixe-cl -collect survey-project-dir ./my_proj ./your_app
```

```
mpirun -n 13 ./your_app \  
: -n 1 advixe-cl -collect survey -project-dir ./my_proj ./your_app \  
: -n 13 ./your_app
```

Intel MPI-specific (adding corner rank and middle surface rank):

```
mpirun -n 27 \  
-gtool "advixe-cl -collect survey --project-dir ./my_proj :1,5,14" ./your_app
```

or: **I_MPI_GTOOL**="**advixe-cl** -collect survey --project-dir ./my_proj :1,5,14"

FLOPS AND MASK UTILIZATION PROFILER



Precise Repeatable FLOPS Metrics

Intel® Advisor – Vectorization Optimization

- FLOPS by loop and function
- All recent Intel processors (not co-processors)
- Instrumentation (count FLOPs) plus sampling (time with low overhead)
- Adjusted for masking with AVX-512 processors

Function Call Sites and Loops		FLOPS							
+	-	GFLOPS	AI	L1 GB/s	GFLOP	FLOP Per Iteration	L1 GB	L1 Bytes Per Iteration	
⌵	🔄	[loop in matvec at Multiply.c:69]	0.826 0	0.1633	5.0586	3.0720	32	18.8160	196
⌵	🔄	[loop in matvec at Multiply.c:60]	0.912 0	0.1633	5.5853	3.0720	32	18.8160	196
⌵	🔄	[loop in matvec at Multiply.c:69]	1.248 0	0.2500	4.9920	1.3440	4	5.3760	16
⌵	🔄	[loop in matvec at Multiply.c:60]	1.592 0	0.2500	6.3699	1.3440	4	5.3760	16
+	🔄	[loop in matvec at Multiply.c:69]	3.055 0	0.2500	12.2205	0.0960	16	0.3840	64
+	🔄	[loop in matvec at Multiply.c:60]	6.282 0	0.2500	25.1279	0.0960	16	0.3840	64

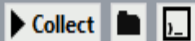
Getting FLOP/S in Advisor

FLOP/S = #FLOP/Seconds	Seconds	#FLOP - and Mask Utilization - and #Bytes
Step 1: Survey <ul style="list-style-type: none">- Not intrusive, sampling based, ~5-10% overhead- Time/Performance-representative		
Step 2: Trip counts+FLOPS <ul style="list-style-type: none">- Precise, instrumentation based- Physically count Num-Instructions- Possible to do all types of dynamic analysis including mask register tracking etc		

1. Survey Target



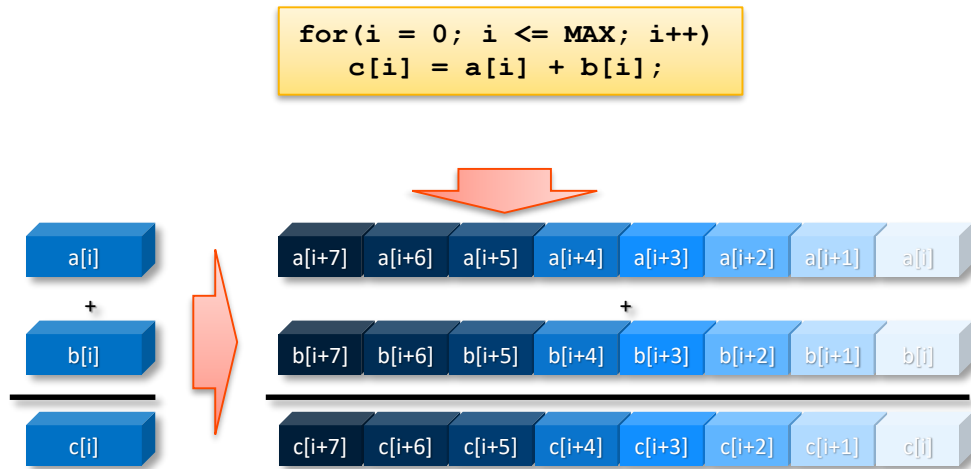
1.1 Find Trip Counts and FLOPS



Why Mask Utilization is Important?

100%

#FLOP (MAX = 8): 8



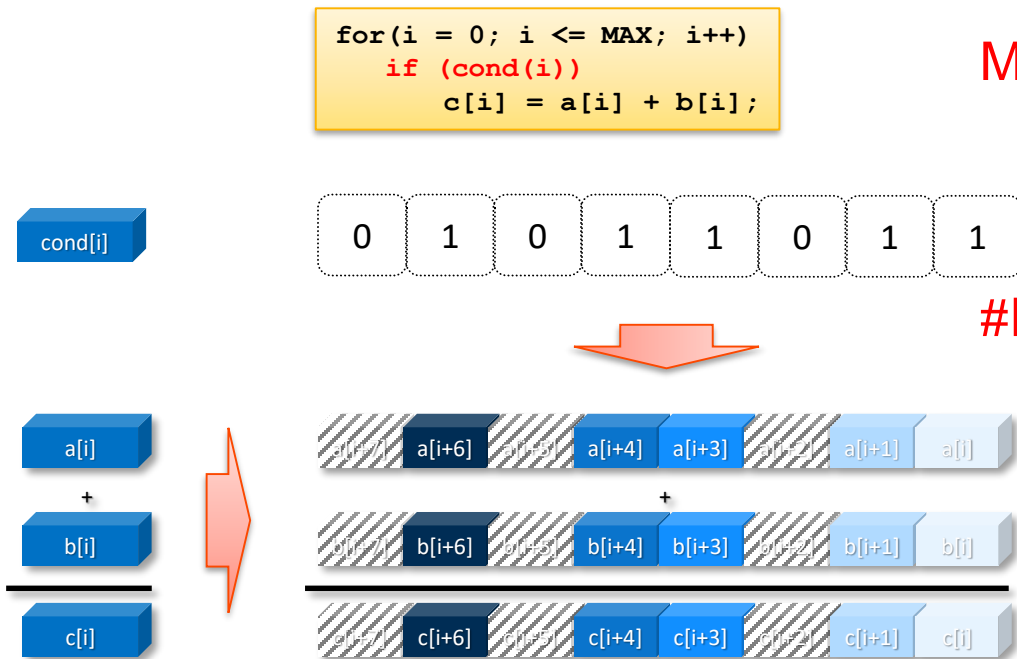
Why Mask Utilization Important?

3 elements suppressed

Mask Utilization = 5/8

62.5%

#FLOP (MAX = 8): 5



VECTORIZATION EFFICIENCY

Spend your time in the most efficient place!

A typical vectorized loop consists of...

Main vector body

- Fastest among the three!

Fastest!

Optional peel part

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

Less
Fast

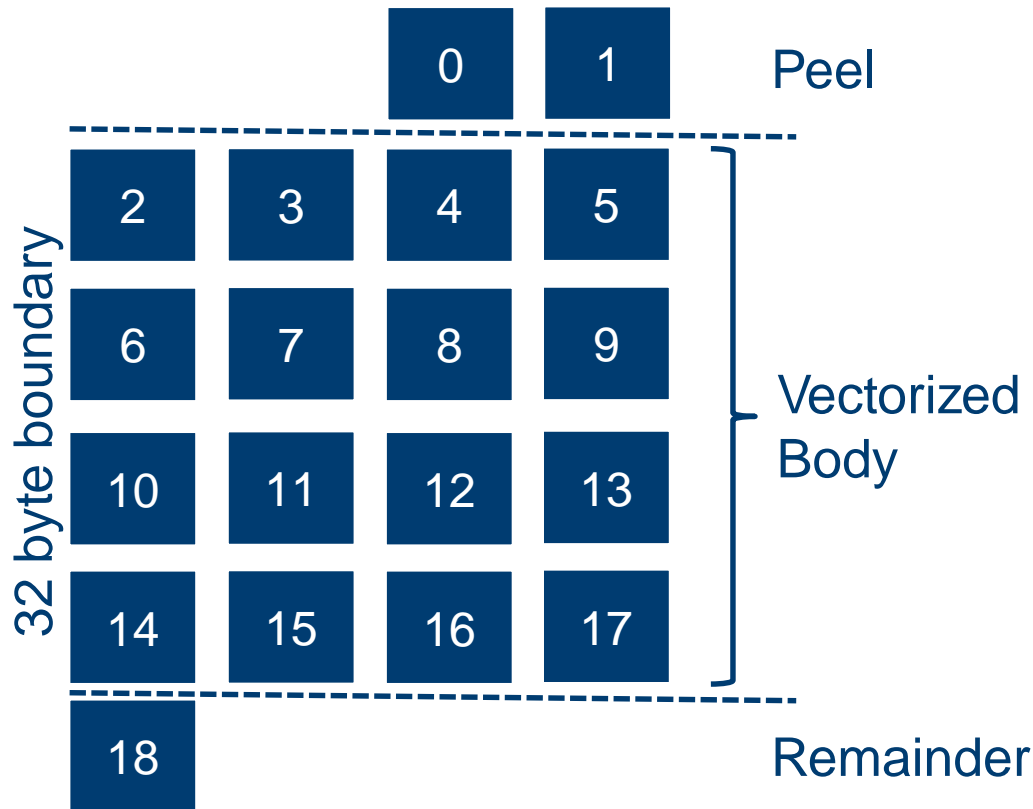
Remainder part

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

Larger vector register means more iterations in peel/remainder

- Make sure you align your data! (and you tell the compiler it is aligned!)
- Make the number of iterations divisible by the vector length!

What are peels and remainders?



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

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

Don't Just Vectorize, Vectorize Efficiently

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

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

Get Specific Advice For Improving Vectorization

Intel® Advisor – Vectorization Advisor

The screenshot shows the Intel Advisor XE 2016 interface. At the top, the title is "Where should I add vectorization and/or threading parallelism?". Below the title bar, there are navigation tabs: Summary, Survey Report, Refinement Reports, Annotation Report, and Suitability Report. A status bar shows "Elapsed time: 8,81s" and filters for "Vectorized", "Not Vectorized", "All Modules", and "All Sources".

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

A blue callout box points to the "Ineffective ..." issue in the third row, containing the text: "Click to see recommendation".

Below the table, there are tabs for "Top Down", "Source", "Loop Assembly", "Assistance", "Recommendations", and "Compiler Diagnostic Details". The "Recommendations" tab is active, showing an issue:

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.

Under the issue, there is a recommendation: "Disable unrolling". The text explains: "The [trip count](#) after loop unrolling is too small compared to [factor](#) using a [directive](#)."

ICL/ICC/ICPC Directive	IFORT Directive
#pragma nounroll	IDIRS NOUNROLL
#pragma unroll	IDIRS UNROLL

Below the table, it says "Read More:" followed by a link: "User and Reference Guide for the Intel C++ Compiler 15.0 > Compiler Reference > Pragmas > Intel-specific Pragma Reference > unroll/nounroll."

A blue callout box points to the "Disable unrolling" recommendation, containing the text: "Advisor shows hints to move iterations to vector body."

Critical Data Made Easy

Loop Trip Counts

Knowing the time spent in a loop is not enough!

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

Summary Survey Report Refinement Reports Annotation Reports Suitability Report

Program time: 12.82s Vectorized Not Vectorized Filters: All Modules All Sources

Function Call Sites and Loops	Self Time	Total Time	🔥	💡	Trip Counts				Compiler Vectorization	
					Median	Min	Max	Call Count	Loop Type	Why No Vectorization
[-] [V] [loop at Multiply.c:53 in matvec]	11.898s	11.898s		💡 1					Collapse	Collapse
[-] [V] [loop at Multiply.c:53 in matvec]	11.851s	11.851s		💡 1	101	101	101	12000000	Vectorized (Body)	vector dependence p
[-] [V] [loop at Multiply.c:53 in matvec]	0.047s	0.047s			3	3	3	1000000	Vectorized (Body)	
[-] [V] [loop at Multiply.c:53 in matvec]	0.413s	0.413s			101	101	101	2000000	Scalar	
[+] [V] [loop at Multiply.c:45 in matvec]	0.109s	12.373s		💡 1					Expand	Expand
[-] [V] [loop at Driver.c:146 in main]	0.016s	12.483s		💡 1	1000000	1000000	1000000	1	Scalar	vector dependence p

1.1 Find Trip Counts

Find how many iterations are executed.

▶ 📁

Command Line

Check actual trip counts

Loop is iterating 101 times but called > million times

Since the loop is called so many times it would be a big win if we can get it to vectorize.

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]
```

$i \neq j \rightarrow B[i] \neq B[j]$

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..

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:
- [qopt-assume-safe-padding, qopt-assume-safe-padding: loop_count](#)
- [Utilizing Full Vectors and Use of Option -qopt-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	Instruction Set Analysis	Data Types	Nu.	Se
2 Possible ineff...	FMA; Gathers; Type Conversions	Float64; Int32	14	

Traits

Gathers

- Irregular Memory Access Patterns May Decrease Performance

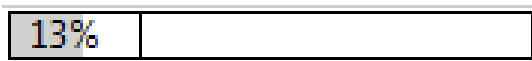


Vector Efficiency: All The Data In One Place

My “performance thermometer”

Elapsed time: 8,01s

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



Achieved Efficiency

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

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

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

Survey: Find out if your code is “under vectorized” and why

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

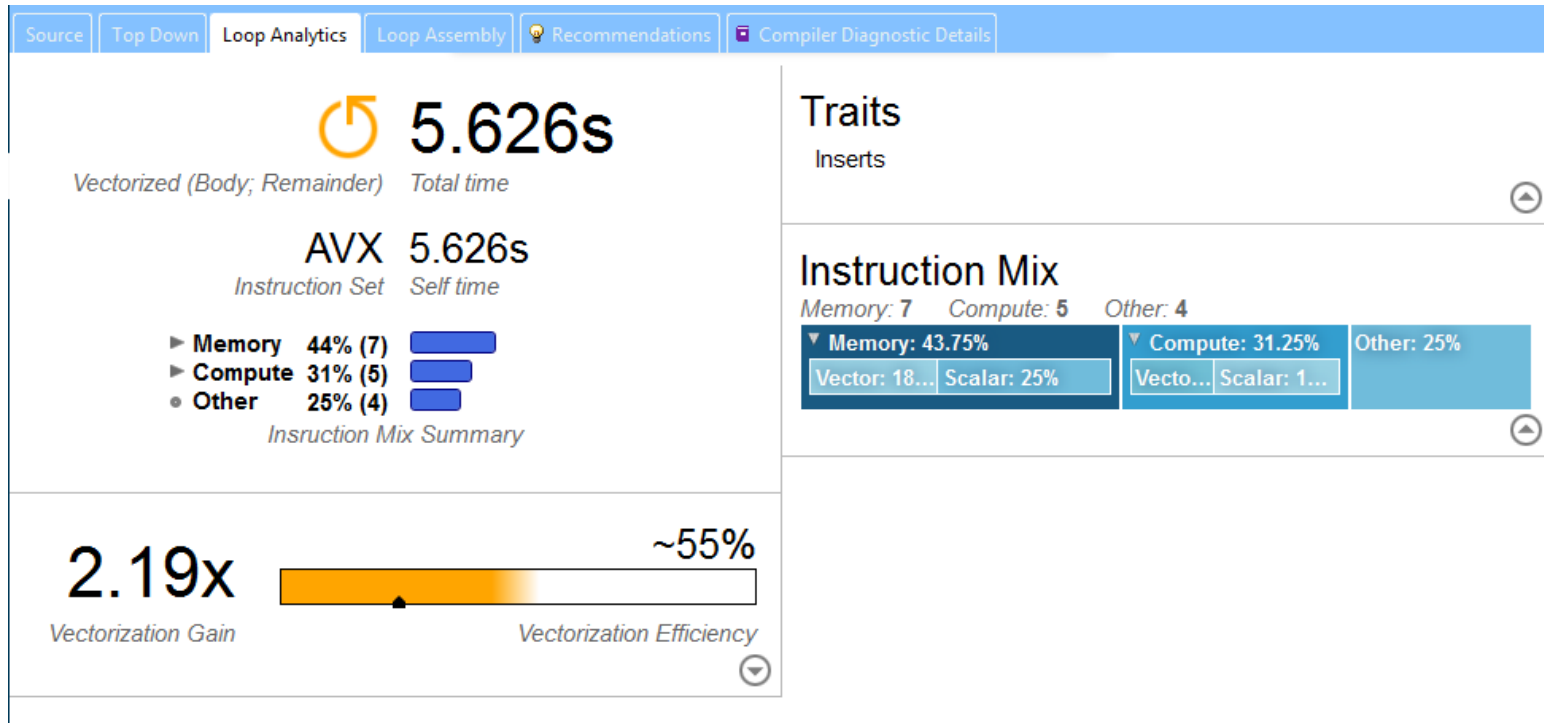
Optional: Specify the trip count, if it is not constant, using a [directive](#): `#pragma loop_count`

Read More:

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

Loop Analytics

Get detailed information about your loops



DEPENDENCY ANALYSIS

Factors that prevent Vectorizing your code

1. Loop-carried dependencies

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

`M >= SIMDlength?`

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

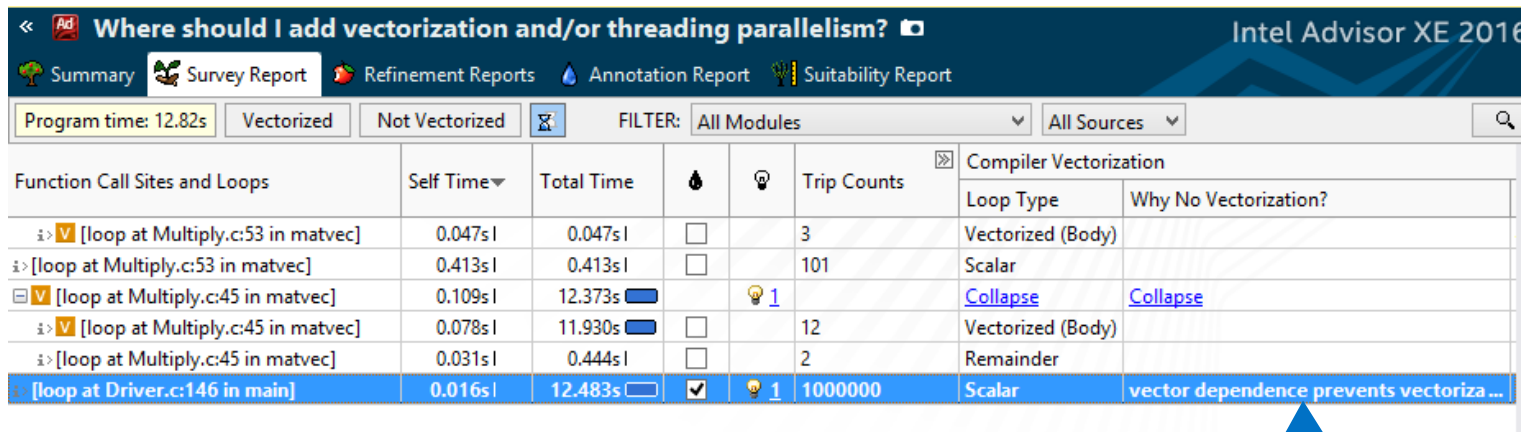
```
for(i = 0; i <= MAX; i++) {
  for(j = 0; j <= MAX; j++) {
    D[j][i] += 1;
  }
}
```

5. Cost-benefit (compiler specific..)

And others.....

Is It Safe to Vectorize?

Loop-carried dependencies analysis verifies correctness



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

Summary Survey Report Refinement Reports Annotation Report Suitability Report

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

Function Call Sites and Loops	Self Time	Total Time	🔥	💡	Trip Counts	Compiler Vectorization	
						Loop Type	Why No Vectorization?
↳ [loop at Multiply.c:53 in matvec]	0.047s	0.047s	<input type="checkbox"/>		3	Vectorized (Body)	
↳ [loop at Multiply.c:53 in matvec]	0.413s	0.413s	<input type="checkbox"/>		101	Scalar	
↳ [loop at Multiply.c:45 in matvec]	0.109s	12.373s	<input type="checkbox"/>	1		Collapse	Collapse
↳ [loop at Multiply.c:45 in matvec]	0.078s	11.930s	<input type="checkbox"/>		12	Vectorized (Body)	
↳ [loop at Multiply.c:45 in matvec]	0.031s	0.444s	<input type="checkbox"/>		2	Remainder	
↳ [loop at Driver.c:146 in main]	0.016s	12.483s	<input checked="" type="checkbox"/>	1	1000000	Scalar	vector dependence prevents vectoriza...

2.1 Check Correctness

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

▶ 📄

Command Line

Select loop for Correct Analysis and press play!

Vector Dependence prevents Vectorization!

Data Dependencies – Tough Problem #1

Is it safe to force the compiler to vectorize?

```
for (i=0;i<N;i++)           // Loop carried dependencies!  
    A[i] = A[i-M]*C[i];    // Need to check if it is safe to force  
                           // the compiler to vectorize!
```

Issue: Assumed dependency present

The compiler assumed there is an anti-dependency (Write after read – WAR) or true dependency (Read after write – RAW) in the loop. Improve performance by investigating the assumption and handling accordingly.

Enable vectorization

Potential performance gain: Information not available until Beta Update release

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

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

ICL/ICC/ICPC Directive	IFORT Directive	Outcome
<code>#pragma simd</code> or <code>#pragma omp simd</code>	<code>!DIR\$ SIMD</code> or <code>!\$OMP SIMD</code>	Ignores all dependencies in the loop
<code>#pragma ivdep</code>	<code>IDIR\$ 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`

Correctness – Is It Safe to Vectorize?

Loop-carried dependencies analysis

Check for loop-carried dependencies in your application

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Site Name Site Function Site Info Loop-Carried Dependencies Strides Distribution Access Pattern

loop_site_6 main main.cpp:13 RAW:1 WAR:1 WAW:1 91% / 0% / 9% Mixed strides

Detected dependencies

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

ID	Description	Source	Function	Module	State
X17	Read	main.cpp:22	main	test_1.exe	New
20		k += a[9];			
21		k -= a[8];			
22		k -= a[7];			
23		k += a[6];			
24		k -= a[5];			
X18	Read	main.cpp:23	main	test_1.exe	New
21		k -= a[8];			
22		k -= a[7];			
23		k += a[6];			

Source lines with Read and Write accesses detected

Received recommendations to force vectorization of a loop:

1. Mark-up loop and check for REAL dependencies
2. Explore dependencies with code snippets

In this example 3 dependencies were detected:

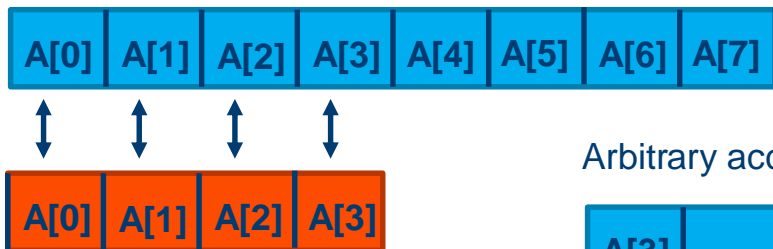
- RAW – Read After Write
- WAR – Write After Read
- WAW – Write After Write

This is NOT a good candidate to force vectorization!

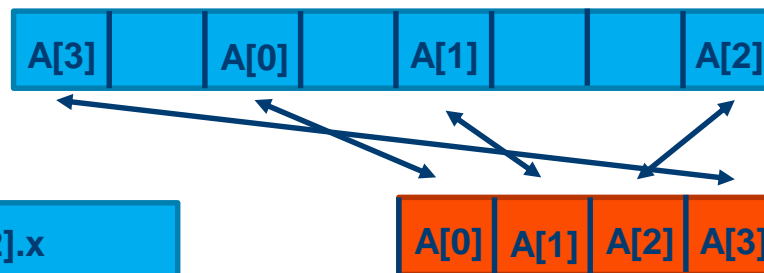
MEMORY ACCESS PATTERN ANALYSIS

Memory access patterns

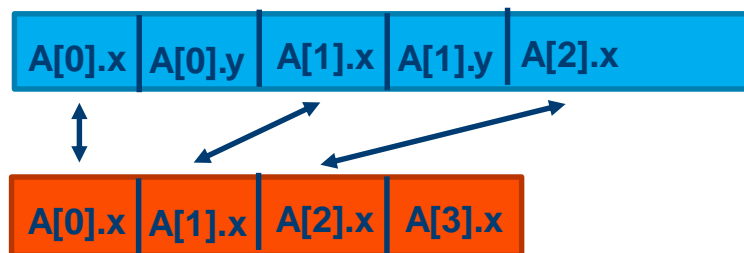
Unit strided (contiguous):



Arbitrary access:



Constant strided:



Memory access patterns

Unit strided (contiguous):



Efficient

Arbitrary access:



Very inefficient

Constant strided:

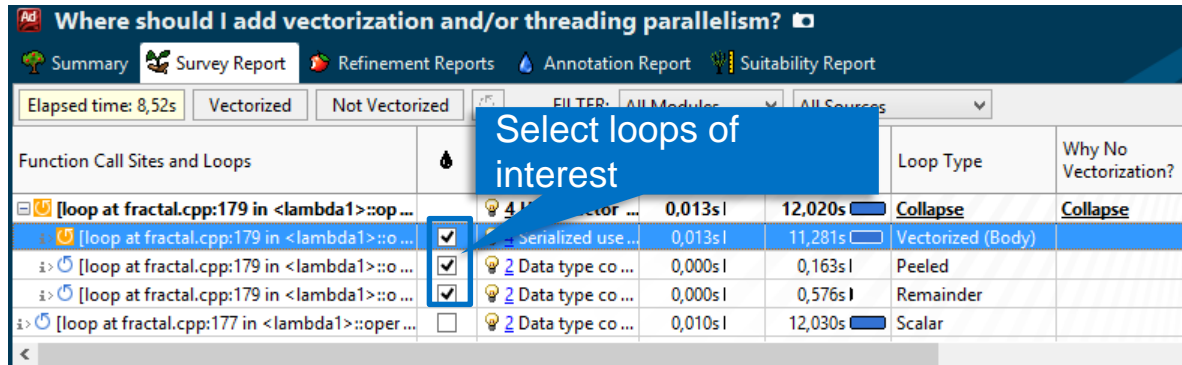


Less efficient

Extract/insert, shuffle,
gather/scatter instructions
are used

Improve Vectorization

Memory Access pattern analysis



Function Call Sites and Loops	Vectorized	Not Vectorized	Loop Type	Why No Vectorization?
[loop at fractal.cpp:179 in <lambda1>::op ...]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Collapse	Collapse
[loop at fractal.cpp:179 in <lambda1>::o ...]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Vectorized (Body)	
[loop at fractal.cpp:179 in <lambda1>::o ...]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Peeled	
[loop at fractal.cpp:179 in <lambda1>::o ...]	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Remainder	
[loop at fractal.cpp:177 in <lambda1>::oper ...]	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Scalar	

2.2 Check Memory Access Patterns

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



Command Line



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

Irregular access patterns decreases performance!



Gather profiling

Run Memory Access Pattern Analysis


2.2 Check Memory Access Patterns ²


▶ Collect  

💧 -- Nothing to analyze --

  0%: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

 Uniform stride (stride 0) = Instruction accesses the same memory from iteration to iteration

 50%: percentage of memory instructions with fixed or constant non-unit stride accesses


Constant stride (stride N) = Instruction accesses memory that consistently changes 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

  50%: 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

Enhanced Memory Access Analysis

Are you bandwidth or compute limited?

Measure Footprint

- Compare to cache size
Does it fit in L2 cache?

Variable References

- Map data to variable names
for easier analysis

Gather/Scatter

- Detect unneeded
gather/scatters that reduce
performance

The screenshot displays the Intel VTune memory access analysis tool interface. It is divided into several sections:

- Site Location:** A table showing memory access patterns for different sites. The third site, `[loop in s272_ at loopstl.cpp:3447]`, is highlighted in blue and has a Max. Site Footprint of 320B.
- Memory Access Patterns Report:** A table showing the type of memory access pattern. The selected site shows a "Gather stride" pattern with a footprint of 320B.
- Variable References:** A table showing the variables accessed in the selected site, which are `a, c, d`.
- Code Snippet:** A view of the source code for the selected site, showing a loop with an `if` statement and two arithmetic operations: `a[i_] += c_[i_] * d_[i_];` and `b[i_] += c_[i_] * c_[i_];`.
- Module: lcd_cox!0x432340:** A table showing the assembly instructions for the selected module. The instructions are: `vgatherdpsz`, `leaq`, `vgatherdpsz`, `vfmadd213ps`, `leaq`, and `vmovupsz`.
- Gather/scatter details:** A panel providing information about the "Unit" pattern, including the instruction access values, unit stride, horizontal stride (4 bytes), vertical stride (64 bytes), mask, and active elements in the mask (100.0%).

References

Roofline model proposed by Williams, Waterman, Patterson:
<http://www.eecs.berkeley.edu/~waterman/papers/roofline.pdf>

“Cache-aware Roofline model: Upgrading the loft” (Ilic, Pratas, Sousa, INESC-ID/IST, Thec Uni of Lisbon)
<http://www.inesc-id.pt/ficheiros/publicacoes/9068.pdf>

Additional Material

Intel® Advisor – Threading Design & Prototyping:

- [Product page](#) – overview, features, FAQs, support...
- [Training materials](#) – movies, tech briefs, documentation...
- [Evaluation guides](#) – step by step walk through
- [Reviews](#)

Additional Analysis Tools:

- [Intel® VTune Amplifier](#) – performance profiler
- [Intel® Inspector](#) - memory and thread checker / debugger

Additional Development Products:

- [Intel® Software Development Products](#)

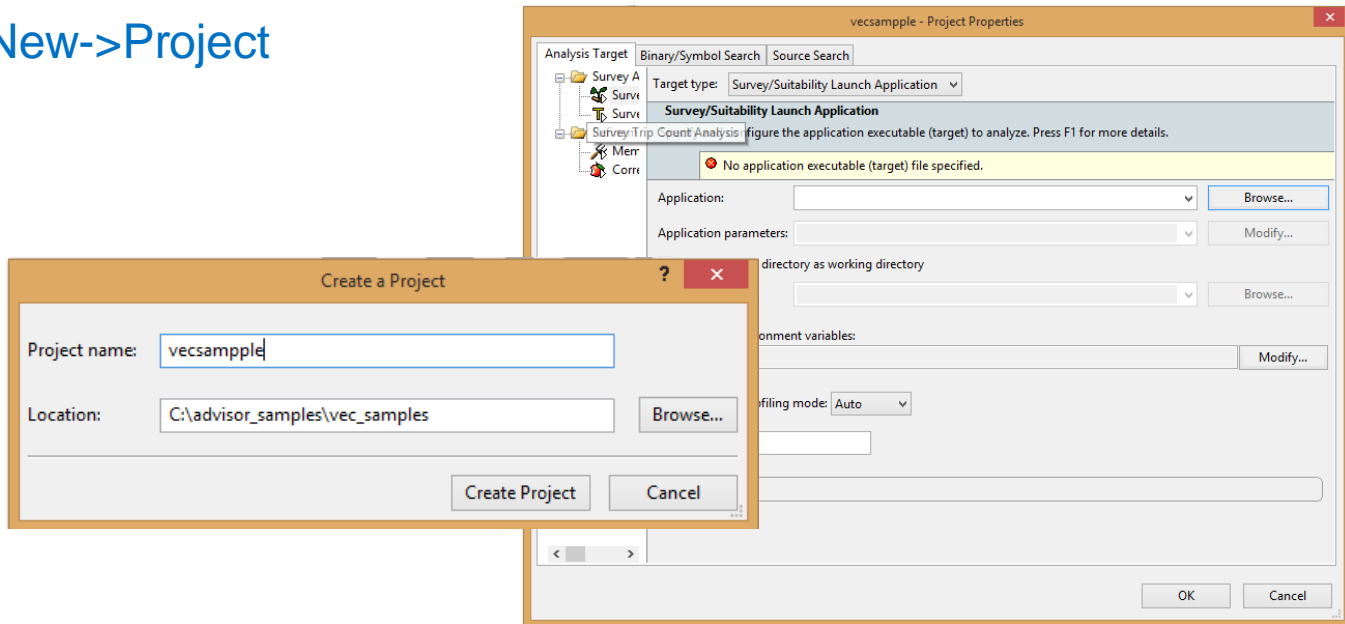
BACKUP

GETTING STARTED

Before you analyze

Create Project

- File->New->Project





Analyze what loops you are spending your time in and how they have been vectorized!

The screenshot shows the Intel Advisor XE 2016 interface. On the left, a sidebar contains several sections: '1. Survey Target', '1.1 Find Trip Counts', 'Mark Loops for Deeper Analysis', and '2.2 Check Memory Access Patterns'. Each section has a 'Collect' button and a 'Command Line' link. A blue callout box labeled 'Click Collect' points to the 'Collect' button in the '1. Survey Target' section. Another blue callout box labeled 'Survey Report' points to the 'Survey Report' button in the 'Mark Loops for Deeper Analysis' section. A green callout box labeled 'Command line created by GUI' points to the 'Command Line' link in the '2.2 Check Memory Access Patterns' section. The main window displays a report titled 'Where should I add vectorization and threading parallelism?' with a table of 'Vectorized Loops'.

Vectorized Loops	Estim...	Vector Length	Compiler Estimated Gain	Traits	Data Types
14.030s	14.030s	Scalar		vector dependence ...	Float64
0.985s	15.015s	Scalar		outer loop was not a ...	Float64
0.000s	15.035s	Scalar		loop with function c ...	Float64



Next analyze how many times your loops are iterating and how many times they are called.

1. Survey Target
Explore where to add efficient vectorization and/or threading.

▶ Collect  

Command Line

1.1 Find Trip Counts
Find how many iterations are executed.









▶ Collect  

Command Line



Mark Loops for Deeper Analysis
Select loops in the Survey result for deeper analysis.

Correctly mark loops for deeper analysis.


Loops

Loops		Vector Issues	Self Time	Total Time	Trip Counts					Loop Type
					Median	Min	Max	Call Count	Iteration Duration	
  [loop at Multiply.c:55 in matvec]	<input type="checkbox"/>	 1 Assumed de...	14.030s	14.030s	50	50	50	101000000	< 0.0001s	Scalar
  [loop at Multiply.c:44 in matvec]	<input type="checkbox"/>		0.985s	15.015s	101	101	101	1000000	< 0.0001s	Scalar
  [loop at Driver.c:145 in main]	<input type="checkbox"/>		0.000s	15.035s	1000000	1000000	1000000	1	< 0.0001s	Scalar



2.1 Check Loop Dependencies
Identify dependencies for marked loops. Fix the reported problems.

▶ Collect  

Command Line

 -- Nothing to analyze --

2.2 Check Memory Access Patterns
Identify and explore complex memory accesses for marked loops. Fix the reported problems.

▶ Collect  

Command Line

Click Collect

Trip Counts				
Median	Min	Max	Call Count	Iteration Duration
50	50	50	101000000	< 0.0001s
101	101	101	1000000	< 0.0001s
1000000	1000000	1000000	1	< 0.0001s

Specify loops for deeper analysis

Ad Where should I add vectorization and/or threading parallelism? 📷

🌳 Summary 🌱 Survey Report 🍎 Refinement Reports 💧 Annotation Report 🏆 Suitability Report

Elapsed time: 15.47s Vectorized Not Vectorized FILTER: All Modules All Sources

Loops	🔥	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?
↳ [loop at Multiply.c:55 in matvec]	<input checked="" type="checkbox"/>	💡 1 Assumed ...	14.030s	14.030s	Scalar	📌 vector dependence p ...
↳ [loop at Multiply.c:44 in matvec]	<input checked="" type="checkbox"/>		0.985s	15.015s	Scalar	📌 outer loop was not a ...
↳ [loop at Driver.c:145 in main]	<input checked="" type="checkbox"/>		0.000s	15.035s	Scalar	📌 loop with function c ...

Deeper analysis

Check dependencies

1. Survey Target
Explore where to add efficient vectorization and/or threading.
▶ Collect [Icons]
Command Line

1.1 Find Trip Counts
Find how many iterations are executed.
▶ Collect [Icons]
Command Line

Mark Loops for Deeper Analysis
Select loops in the Survey result for Correctness and/or Memory Access Patterns analysis.
— There are NO marked loops —

2.1 Check Correctness
Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.
▶ Collect [Icons]
Command Line
💧 — Nothing to analyze —

2.2 Check Memory Access Patterns
Identify and explore complex memory accesses for marked loops. Fix the reported problems.
▶ Collect [Icons]
Command Line

Click Collect

We marked 3 loops for a dependency analysis. Two of the loops had no dependencies. One of the loops has Read-After-Write dependency and can't be vectorized.

Check memory access patterns in your application

Summary Survey Report **Refinement Reports** Annotation Report Suitability Report

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
loop at Driver.c:145 in main	✔ No dependencies found	No information available	No information available	loop_site_6
loop at Multiply.c:44 in matvec	✔ No dependencies found	No information available	No information available	loop_site_10
loop at Multiply.c:55 in matvec	✘ RAW:1	No information available	No information available	loop_site_8

Deeper analysis

Memory Access Pattern analysis

Stride distribution

1. Survey Target
Explore where to add efficient vectorization and/or threading.
▶ Collect [Icons]
Command Line

1.1 Find Trip Counts
Find how many iterations are executed.
▶ Collect [Icons]
Command Line

Mark Loops for Deeper Analysis
Select loops in the Survey result for Correctness and/or Memory Access Patterns analysis.
-- There are NO marked loops --

2.1 Check Correctness
Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.
▶ Collect [Icons]
Command Line
-- Nothing to analyze --

2.2 Check Memory Access Patterns
Identify and explore complex memory accesses for marked loops. Fix the reported problems.
▶ Collect [Icons]
Command Line

Ad Check memory access patterns in your application

Summary Survey Report Refinement Reports Annotation Report Suitability Report

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
loop at Driver.c:145 in main	✔ No dependencies found	100% / 0% / 0%	All unit strides	loop_site_6
loop at Multiply.c:44 in matvec	✔ No dependencies found	85% / 15% / 0%	Mixed strides	loop_site_10
loop at Multiply.c:55 in matvec	✘ RAW:1	74% / 26% / 0%	Mixed strides	loop_site_8

Memory Access Patterns Report Correctness Report

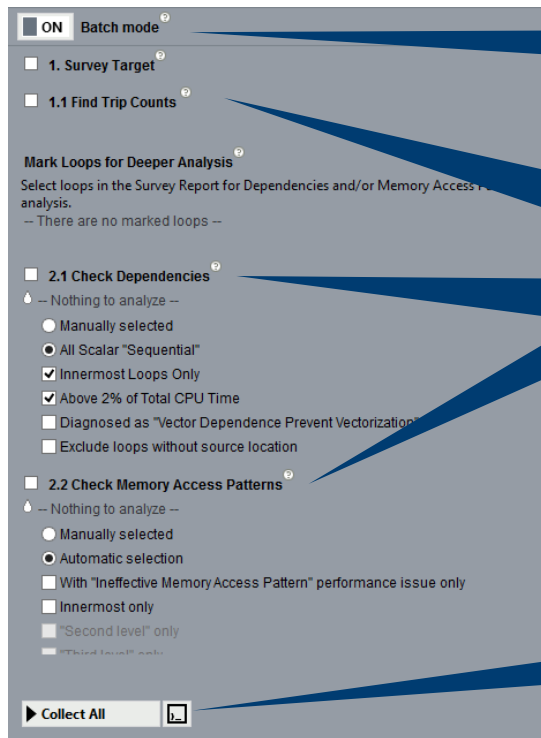
ID	Stride	Type	Source	Nested Function	Modules	Alignment
P3		Parallel site information	Driver.c:145		matrix_vector_multiplication_c.exe	
P9	0	Unit stride	Driver.c:157		matrix_vector_multiplication_c.exe	
P10	0	Unit stride	Multiply.c:39	matvec	matrix_vector_multiplication_c.exe	
P12	0	Unit stride	Multiply.c:44	matvec	matrix_vector_multiplication_c.exe	
P14	0; 1	Unit stride	Multiply.c:45	matvec	matrix_vector_multiplication_c.exe	

```
43     int i, j;
44
45     for (i = 0; i < size1; i++) {
46         b[i] = 0;
47     }
```

Click Collect

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

Command Line: Intel® Advisor XE

Collecting survey and tripcounts

```
advixe-cl --collect survey --project-dir ./advi -- mult.exe
```

```
advixe-cl --collect tripcounts --project-dir ./advi -- mult.exe
```

Creating snapshot in command line, e.g:

```
advixe-cl --snapshot --project-dir ./advi \  
--pack --cache-sources --cache-binaries -- /tmp/new_snapshot
```

Viewing the results

```
advixe-gui ./advi
```

```
advixe-cl --report survey --project-dir ./advi
```

Advisor works with GCC and Microsoft Compilers

Adds bonus capabilities with the Intel Compiler

Advisor using GCC, Microsoft or Intel Compiler:

- Finds un-vectorized loops
- Analyze SIMD, AVX, AVX2, AVX-512
- Dependency Analysis – safely force vectorization with a pragma
- Memory Access Pattern Analysis - optimize stride and caching
- Trip Counts
- FLOPS metrics with masking
- Roofline Analysis – balance memory vs. compute optimization

Intel Compiler Adds:

- Usually better optimized vectorization
- Better compiler optimization messages

Intel Advisor with Intel Compiler Adds:

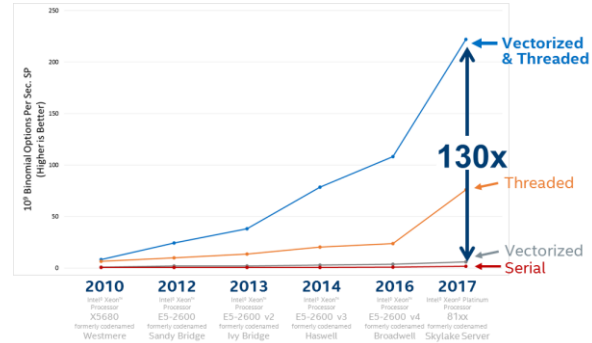
- Finds inefficiently vectorized loops and estimates performance gain
- Compiler optimization report messages displayed on the source
- More tips for improving vectorization
- Optimize for AVX-512 even without AVX-512 hardware

Configurations for 2010-2017 Benchmarks

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors 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	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
WSM	Intel® Xeon™ X5680 Processor	3.33 GHZ	6	2	32K	256K	12 MB	48 MB	1333 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 17.0.2
SNB	Intel® Xeon™ E5 2690 Processor	2.9 GHZ	8	2	32K	256K	20 MB	64 GB	1600 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 17.0.2
IVB	Intel® Xeon™ E5 2697v2 Processor	2.7 GHZ	12	2	32K	256K	30 MB	64 GB	1867 MHz	NUMA	Y	Y	Y	Disabled	RHEL 7.1	3.10.0-229.el7.x86_64	icc version 17.0.2
HSW	Intel® Xeon™ E5 2600v3 Processor	2.2 GHz	18	2	32K	256K	46 MB	128 GB	2133 MHz	NUMA	Y	Y	Y	Disabled	Fedora 20	3.15.10-200.fc20.x86_64	icc version 17.0.2
BDW	Intel® Xeon™ E5 2600v4 Processor	2.3 GHz	18	2	32K	256K	46 MB	256 GB	2400 MHz	NUMA	Y	Y	Y	Disabled	RHEL 7.0	3.10.0-123.el7.x86_64	icc version 17.0.2
BDW	Intel® Xeon™ E5 2600v4 Processor	2.2 GHz	22	2	32K	256K	56 MB	128 GB	2133 MHz	NUMA	Y	Y	Y	Disabled	CentOS 7.2	3.10.0-327.el7.x86_64	icc version 17.0.2
SKX	Intel® Xeon® Platinum 81xx Processor	2.5 GHz	28	2	32K	1024K	40 MB	192 GB	2666 MHz	NUMA	Y	Y	Y	Disabled	CentOS 7.3	3.10.0-514.10.2.el7.x86_64	icc version 17.0.2

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. For more information go to <http://www.intel.com/performance>

Optimization Notice

Copyright © 2020, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Legal Disclaimer & Optimization Notice

Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.

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. For more complete information visit www.intel.com/benchmarks.

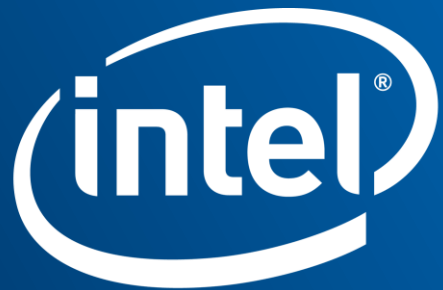
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING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

Copyright © 2018, 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



Software