

Intel® Software Tools Training, CERN, 2-3 March 2022

Intel® Advisor

Vectorization and Roofline Analysis

Klaus-Dieter.Oertel@intel.com



intel®

Intel® Advisor for High Performance Code Design

Rich Set of Capabilities



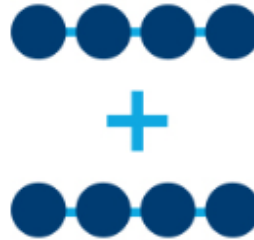
Offload Modelling

Design offload strategy and model performance on GPU.



Roofline Analysis

Optimize your application for memory and compute.



Vectorization Optimization

Enable more vector parallelism and improve its efficiency.



Thread Prototyping

Model, tune, and test multiple threading designs.



Build Heterogeneous Algorithms

Create and analyze data flow and dependency computation graphs.

Vectorization Analysis

Intel® Advisor – Vectorization Advisor

Get breakthrough vectorization performance

- Faster Vectorization Optimization:
 - Vectorize where it will pay off most
 - Quickly identify 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, there are filters for 'All Modules', 'All Sources', 'Loops And Functions', and 'All Threads'. Below the filters, there are tabs for 'Summary', 'Survey & Roofline', and 'Refinement Reports'. The main area displays a table with columns for 'Function Call Sites and Loops', 'Vector Issues', 'Self Time', 'Total Time', 'Type', 'FLOPS' (subdivided into GFLOPS and AI), 'Why No Vectorization?', 'Vectorized Loops' (subdivided into Vector..., Efficiency, Gain..., VL ..), and 'Trip Counts'. The table contains several rows of data, with the first row highlighted in blue. The first row shows a loop in S252 at loops90.f:1172, which is vectorized and has a self time of 3.129s (7.0% of total time). The second row shows a loop in S2101 at loops90.f:1749, which is scalar and has a self time of 2.765s (6.2% of total time). The third row shows a loop in s442_somp\$parallel_for..., which is vectorized+ and has a self time of 1.492s (3.4% of total time). The fourth row shows a function call site f_svmf_sinf8_l9, which is a vector function and has a self time of 1.108s (2.5% of total time). The fifth row shows a loop in S353 at loops90.f:2381, which is vectorized and has a self time of 0.989s (2.2% of total time).

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_svmf_sinf8_l9		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

Part of oneAPI Base Toolkit

software.intel.com/advisor

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

Amdahl's law

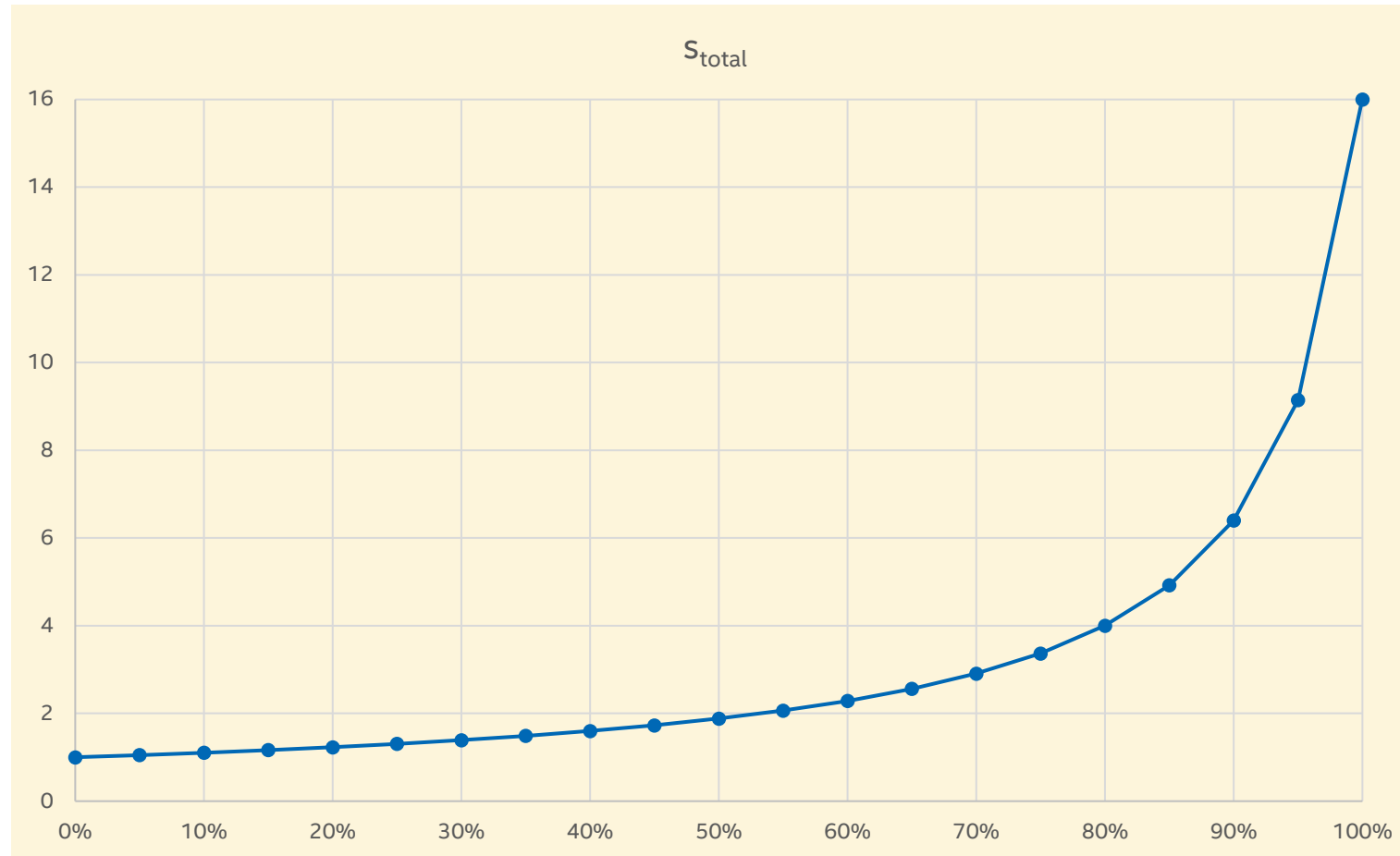
$$S_{total} = \frac{100\%}{(100\% - p) + \frac{p}{s_p}}$$

S = speedup (in parallelized part or total)

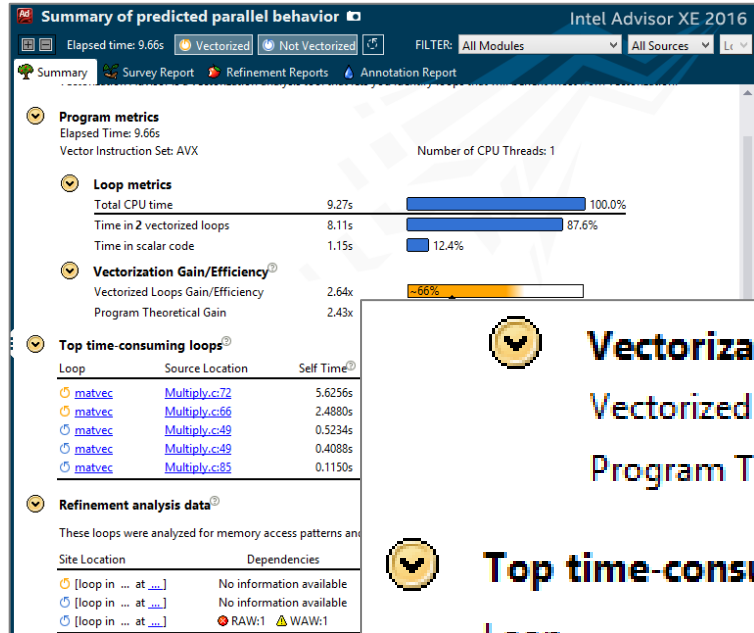
P = proportion of execution time that benefits from parallelization

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

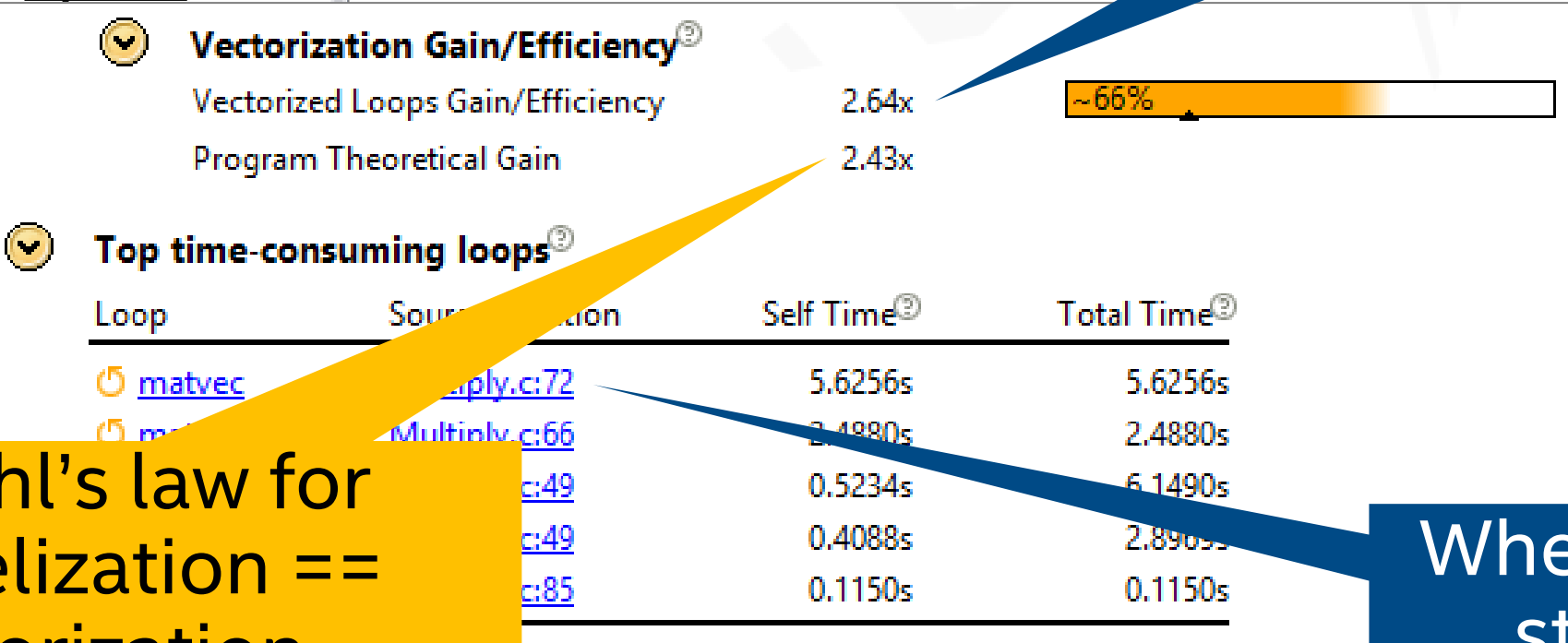
Amdahl's law



Summary View: Plan Your Next Steps



What can I expect to gain?



Amdahl's law for parallelization == vectorization

Where do I start?

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?

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops		
							Vecto...	Efficiency	Vector L..
[loop at stl_algo.h:4740 in std::tr...		0.170s	0.170s		Scalar	non-vectorizable loop ins...			
[loop at loopstl.cpp:2449 in s234_]	2 Ineffective peeled/rem...	0.170s	0.170s	12; 4	Collapse	Collapse	AVX	~100%	4
[loop at loopstl.cpp:2449 in s...		0.150s	0.150s	12	Vectorized (Body)		AVX		4
[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...			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?

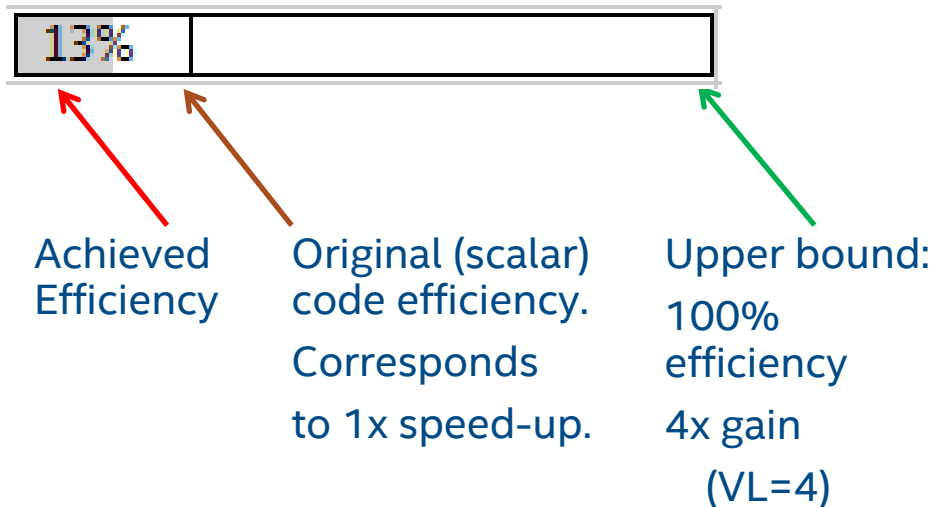
Get Faster Code Faster!

Vector Efficiency: All The Data In One Place

My “performance thermometer”

Elapsed time: 8,01s

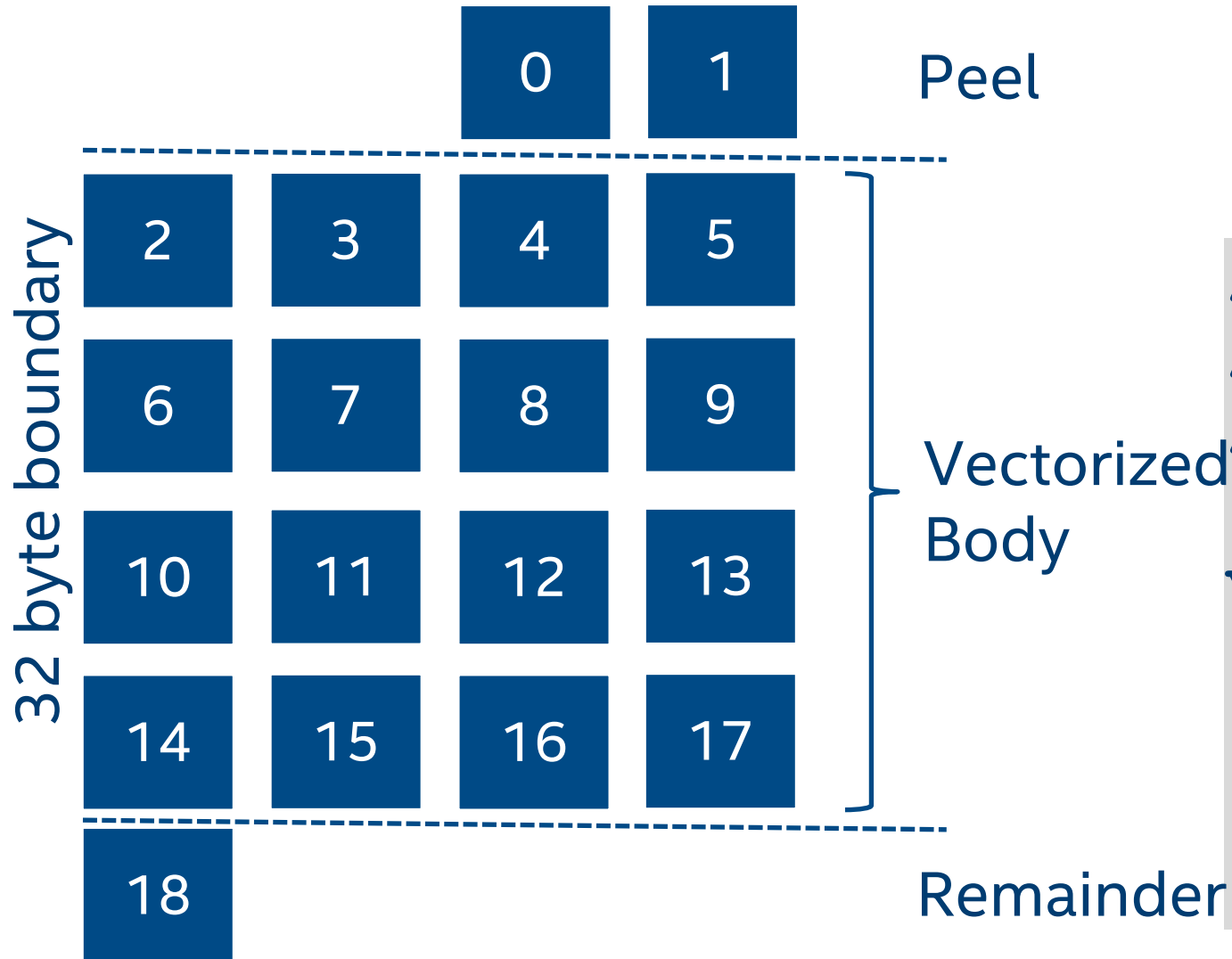
Loops	Vecto...	Efficiency ▲	Estimated Gain	Vect...	Co	Traits	Vector Widths	Self Time
[loop at lbpSUB.cpp:1280 in fPropagationS...	AVX	13%	0,53	4	0,53	Blends; Extracts; Inserts; Shuffles	128/256	2,312s
[loop at lbpGET.cpp:152 in fGetFracSite]	AVX	30%	2,38	8	2,34	Blends; Inserts; Masked Stores	128/256	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



- **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

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] = ...;
}
```

Spend your time in the most efficient place!

A typical vectorized loop consists of...

- Optional peel part

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

- Main vector body

- Fastest among the three!

- 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!

Fastest!

Less
Fast

Get Specific Advice For Improving Vectorization

Intel® Advisor – Vectorization Advisor

Function Call Sites and Loops		Time	Type
[-] [loop in runCforallLambdaLoops at runCFor	💡 2 Ineffective peeled/remainder loop(s) ..	2.703s	Vectorized (Body; Peeled; Remainder)
[-] [loop in runCforallLambdaLoops at runC	💡 1 Possible inefficient memory access patt ..	2.109s	Vectorized (Body)
[-] [loop in runCforallLambdaLoops at runC		0.500s	Vectorized (Remainder)
[-] [loop in runCforallLambdaLoops at runC		0.094s	Remainder
[-] [loop in runCforallLambdaLoops at runC		0.000s	Peeled

Click to see recommendation

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

All Advisor-detectable issues: [C++](#) | [Fortran](#)

!!! Issue: Ineffective peeled/remainder loop(s) detected

All or some [source loop](#) iterations are being moved from [peeled/remainder](#) loops to the loop body.

💡 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.

Advisor shows hints to fix performance issue

Critical Data Made Easy

Loop Trip Counts

Knowing the time spent in a loop is not enough!

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 win if we can get it to vectorize.

Why No Vectorization?

Intel Advisor – Vectorization Advisor

Elapsed time: 125.72s Vectorized Not Vectorized OFF Smart Mode

FILTER: All Modules | All Sources | Loops And Functions | All Threads

Summary | Survey & Roofline | Refinement Reports

Function Call Sites and Loops	Perfor... Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops				Instruction Set		
						Vect...	Efficiency	Gain...	VL ..	Com...	Traits	Da
[loop in main at roofline.cpp:295]	<input type="checkbox"/>	18.538s	18.538s	Vectorized (Bo...		AVX	~100%	5.34x	4	5.34x		Flo
[loop in main at roofline.cpp:310]	<input type="checkbox"/>	18.394s	18.394s	Vectorized (Bo...		AVX	~100%	5.34x	4	5.34x		Flo
[loop in main at roofline.cpp:221]	<input checked="" type="checkbox"/>	14.741s	14.741s	Scalar	novector dir ...							Flo
[loop in main at roofline.cpp:234]	<input type="checkbox"/>	11.117s	11.117s	Scalar	novector dire ...							Flo
[loop in main at roofline.cpp:173]	<input checked="" type="checkbox"/>	10.788s	10.788s	Scalar	novector dire ...							Flo
[loop in main at roofline.cpp:125]	<input checked="" type="checkbox"/>	10.427s	10.427s	Scalar	novector dire ...							Flo

Source | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

File: roofline.cpp:221 main

Line	Source	Total Time	%	Loop/Function Time	%	Traits
210	#pragma omp simd					
217	for (int r = 0; r < REPEAT_1; r++)					
218	{					
219	#pragma unroll (UNROLL_COUNT)					
220	#pragma novector					
221	for (int i = 0; i < ARRAY_SIZE_1; i++)	2.002s		14.741s		
	[loop in main at roofline.cpp:221]					
	Scalar loop. Not vectorized: novector directive used					
	Loop was unrolled by 2					
Selected (Total Time):		2.002s				

Data Dependencies

Is it safe to force the compiler to vectorize?

```
for (i = 0; i < N; i++) // Loop carried dependencies!  
    A[i] = A[i - K] * 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>!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`

Is It Safe to Vectorize?

Loop-carried dependencies analysis verifies correctness

Intel Advisor XE 2016

Where should I add vectorization and/or threading parallelism?

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...

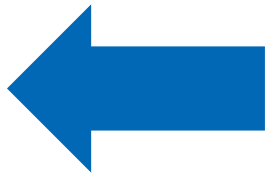
2.1 Check Correctness

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

▶ 📁

Command Line

Select loop for Correct Analysis and press play!



Vector Dependency prevents Vectorization!

Find vector optimization opportunities

Memory Access pattern analysis

Unit-Stride access

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



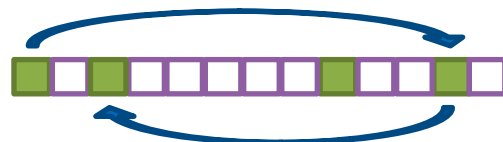
Constant stride access

```
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]
```



Elapsed time: 2,13s Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey & Roofline Refinement Reports

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Footprint Estimate
[loop in main at Source.cpp:162]	No Information Available	67% / 0% / 33%	Mixed Strides	5MB

```
160 {  
161     #pragma nounroll  
162     for (int k = 0; k < N; k += 2)  
163     {  
164         D[q][k] = D[q][k] - G[k][i] * C[i-1] + A[...]/ B[q];
```

Stride distribution

Memory Access Patterns Report Dependencies Report Recommendations

All Advisor-detectable issues: [C++](#) | [Fortran](#)

! Inefficient memory access patterns present

There is a high of percentage memory instructions with irregular (variable or random) stride accesses. Improve performance by investigating and handling accordingly.

💡 Check memory access patterns for the outer loop

This loop has inefficient memory access patterns. If the memory access patterns are more efficient for the outer loop, reorder the loops if possible.

Improve Vectorization

Memory Access pattern analysis

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				Loop Type	Why No Vectorization?
[loop at fractal.cpp:179 in <lambda1>::op ...	<input type="checkbox"/>	4	0,013s 12,020s	Collapse	Collapse
[loop at fractal.cpp:179 in <lambda1>::o ...	<input checked="" type="checkbox"/>	Serialized use ...	0,013s 11,281s	Vectorized (Body)	
[loop at fractal.cpp:179 in <lambda1>::o ...	<input checked="" type="checkbox"/>	Data type co ...	0,000s 0,163s	Peeled	
[loop at fractal.cpp:179 in <lambda1>::o ...	<input checked="" type="checkbox"/>	Data type co ...	0,000s 0,576s	Remainder	
[loop at fractal.cpp:177 in <lambda1>::oper ...	<input type="checkbox"/>	Data type co ...	0,010s 12,030s	Scalar	

Select loops of interest

2.2 Check Memory Access Patterns

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



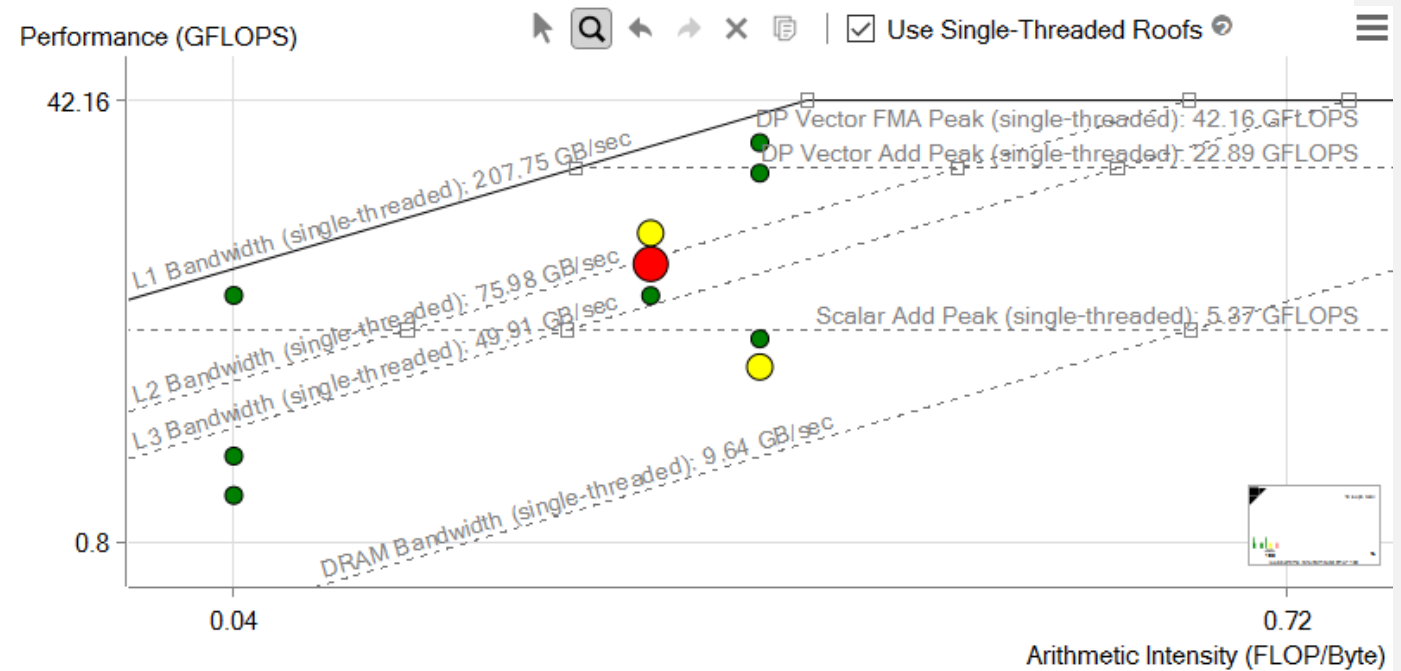
Command Line

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

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

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

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

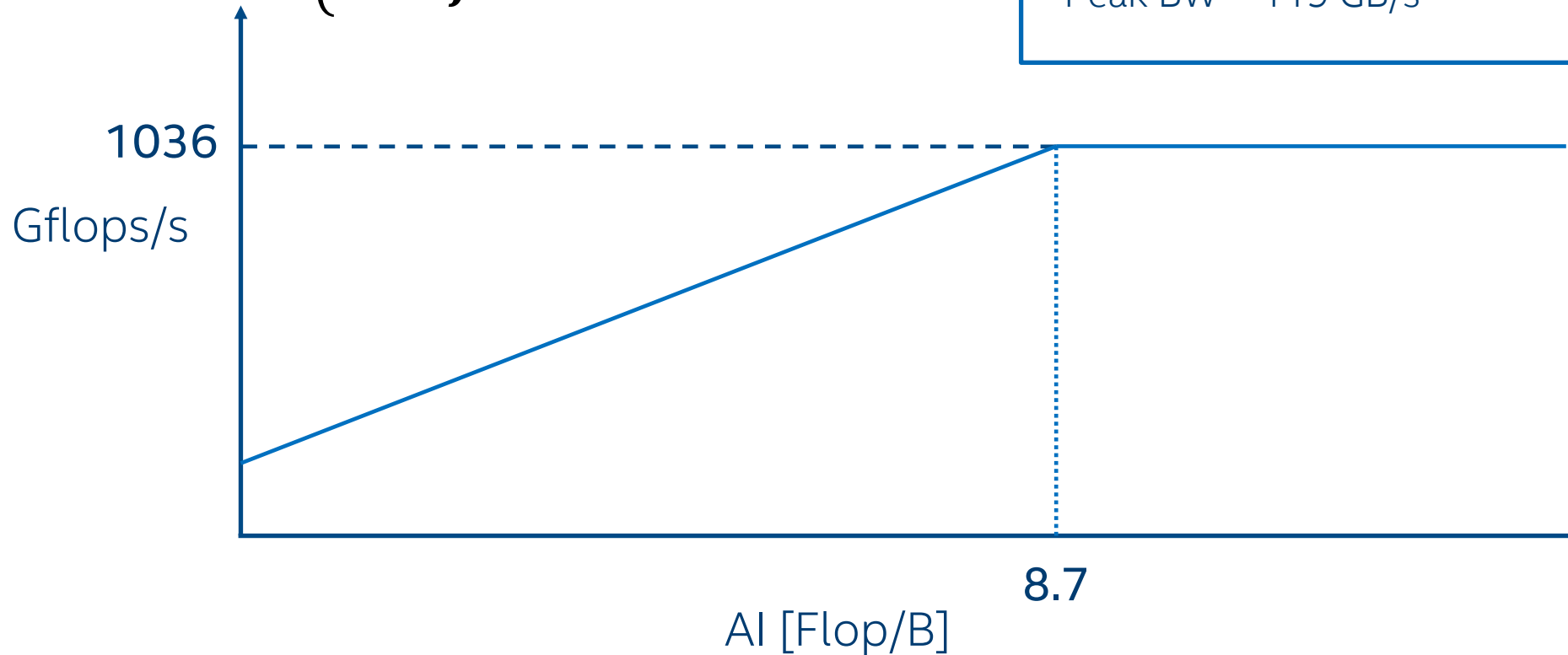
Arithmetic Intensity (Flops/Bytes) 

DRAWING THE ROOFLINE

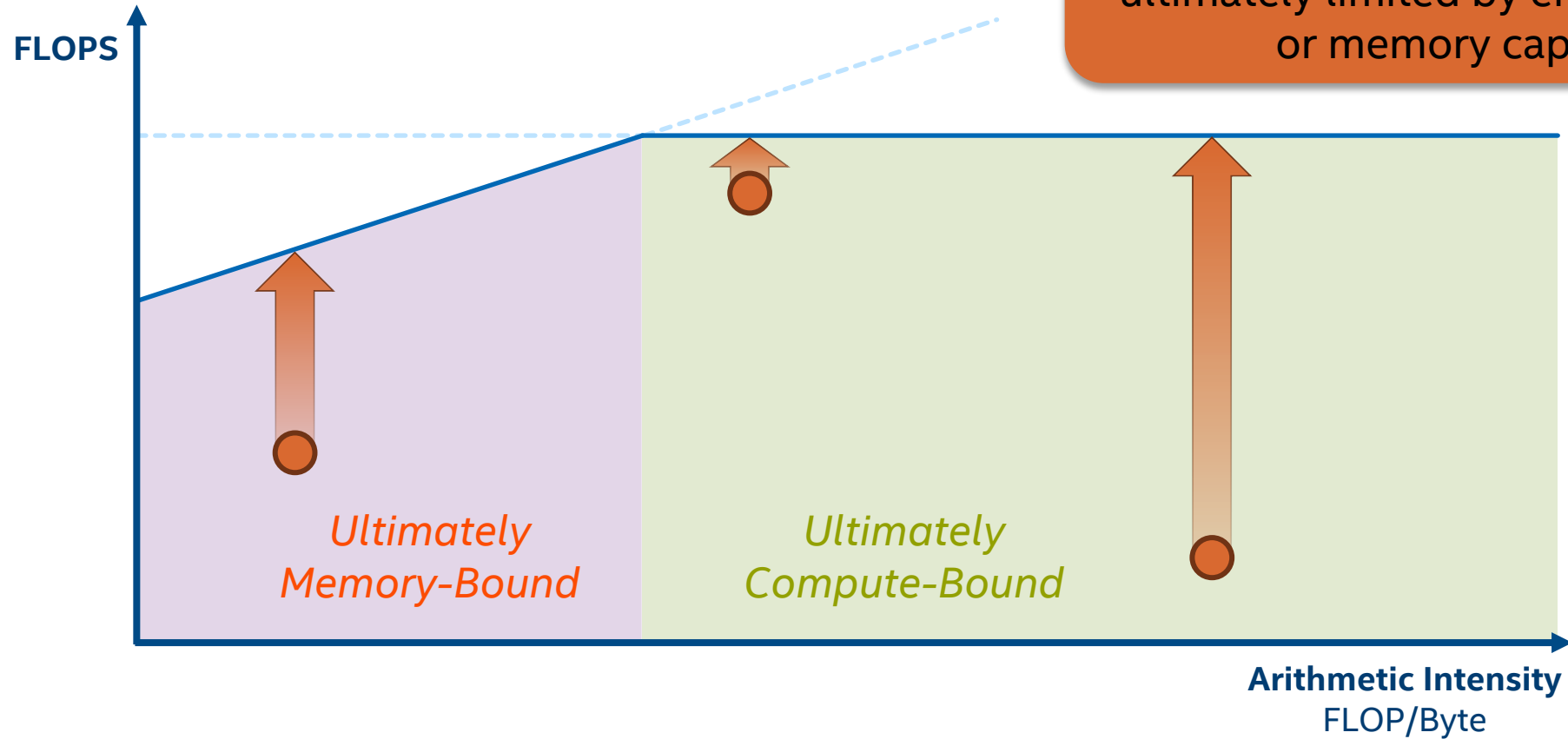
Defining the speed of light

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

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



Ultimate Performance Limits



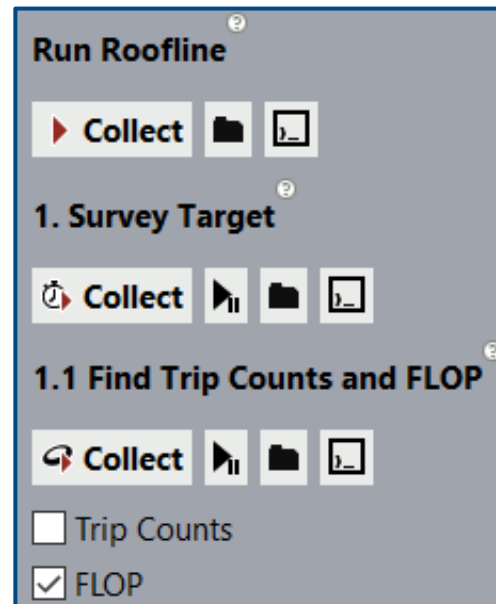
Roofline Metrics

- 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.



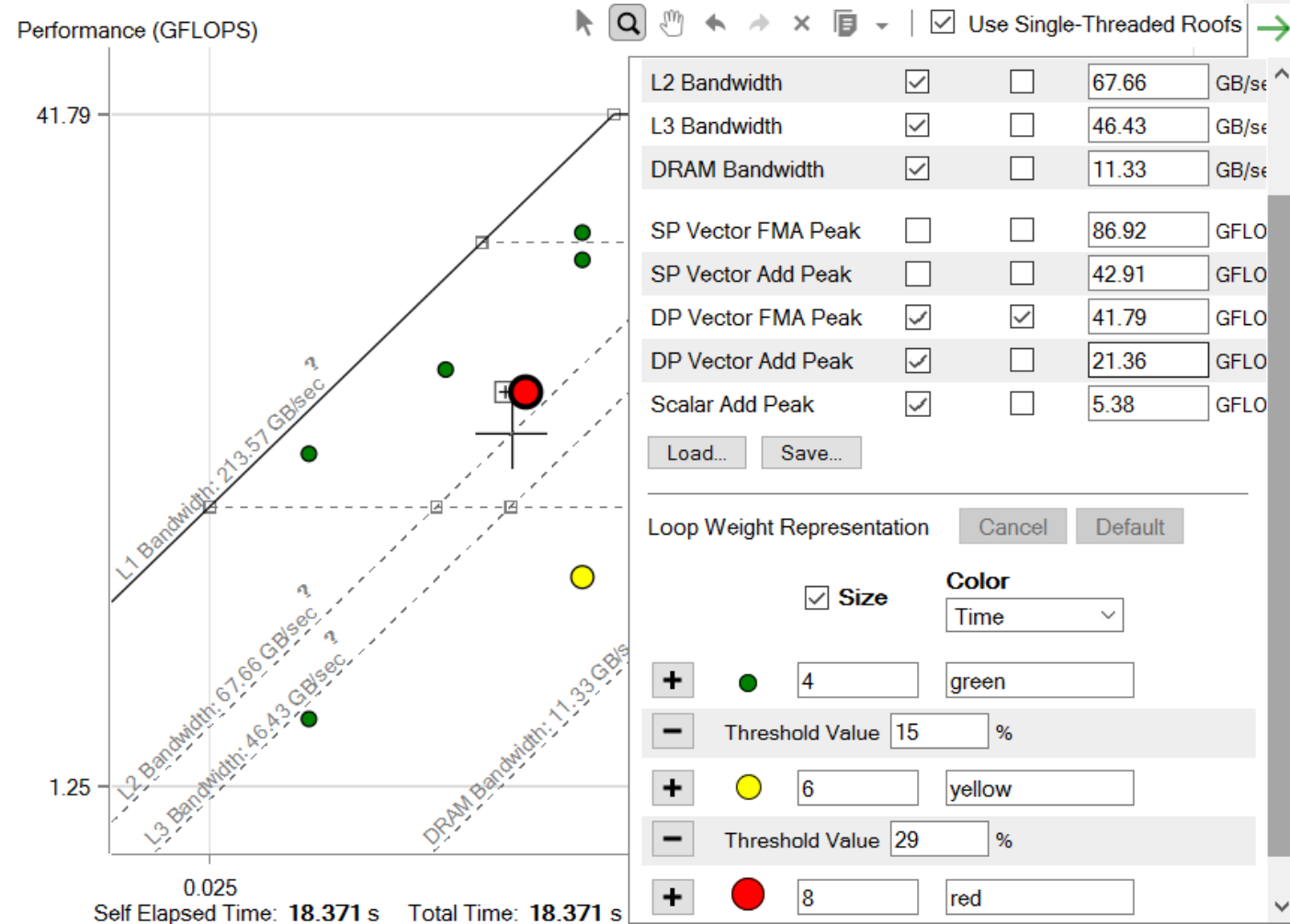
Shortcut to run Survey followed by Trip Counts + FLOPs

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.

The Intel® Advisor Roofline Interface

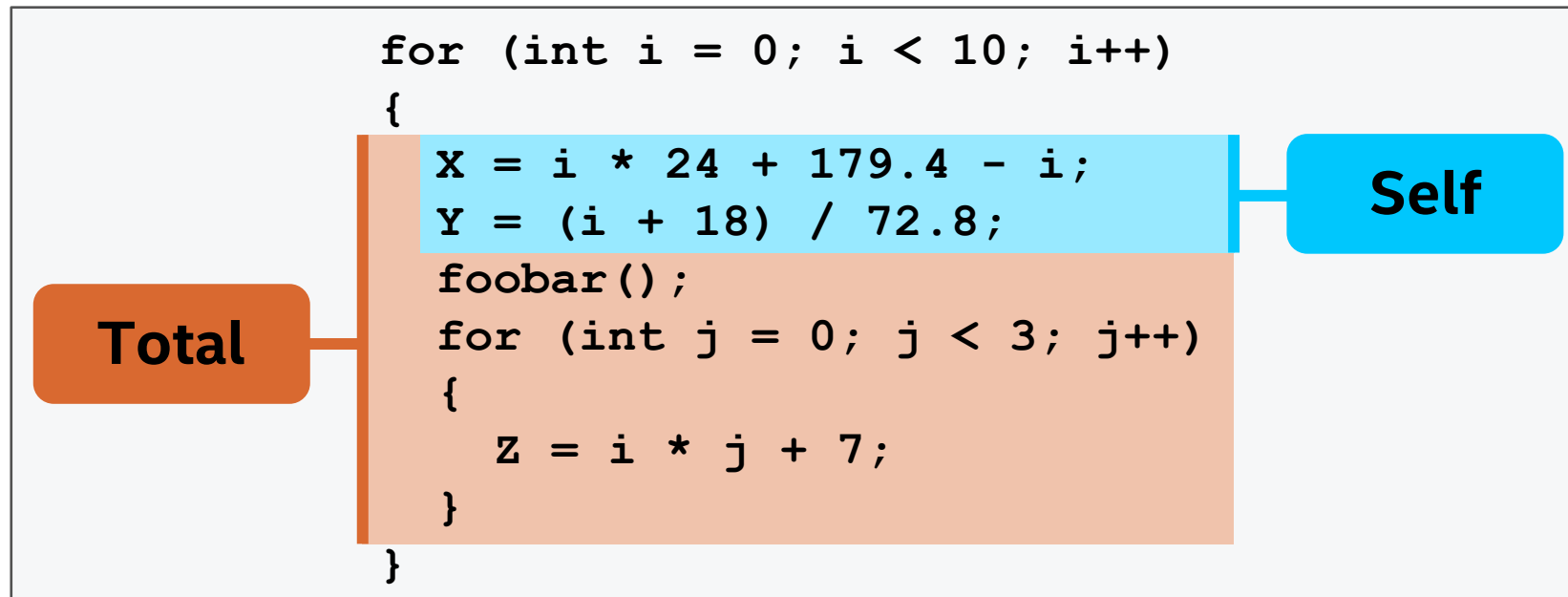
- Roofs are based on benchmarks run before the application.
- Roofs can be hidden, highlighted, or adjusted.
- Intel® Advisor has size- and color-coding for dots.
- Color code by duration or vectorization status
- Categories, cutoffs, and visual style can be modified.



Roofline with call stacks

Self Data vs Total Data

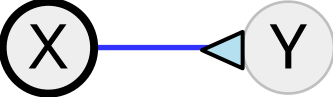
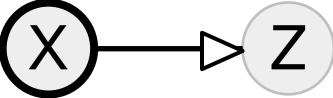
- The original Roofline used only **self data**: only work done directly is recorded.
- The Roofline with call stacks uses both **self data and total data**, which includes work done in functions or loops called as well as work done directly.

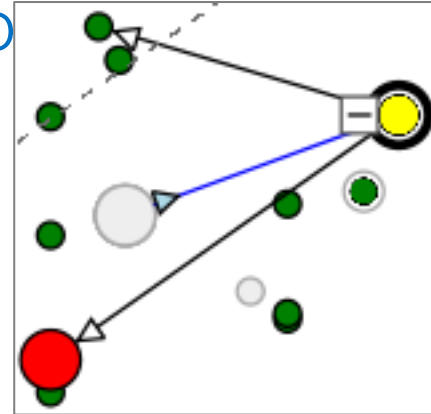


Reading the Roofline with Call Stacks

Visualizing the Call Chain

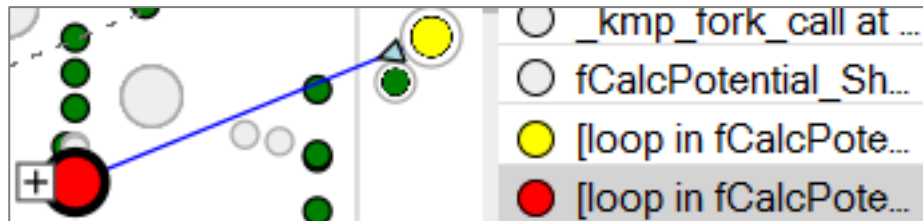
- Arrows indicate relationships between dots

-  X is called directly by Y.
-  X directly calls Z

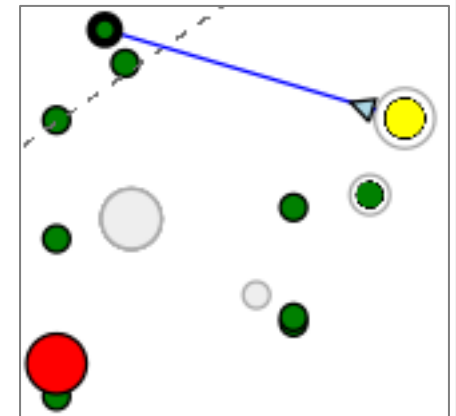


The selected yellow dot was called by the gray dot, and it calls the red and green dots.

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



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



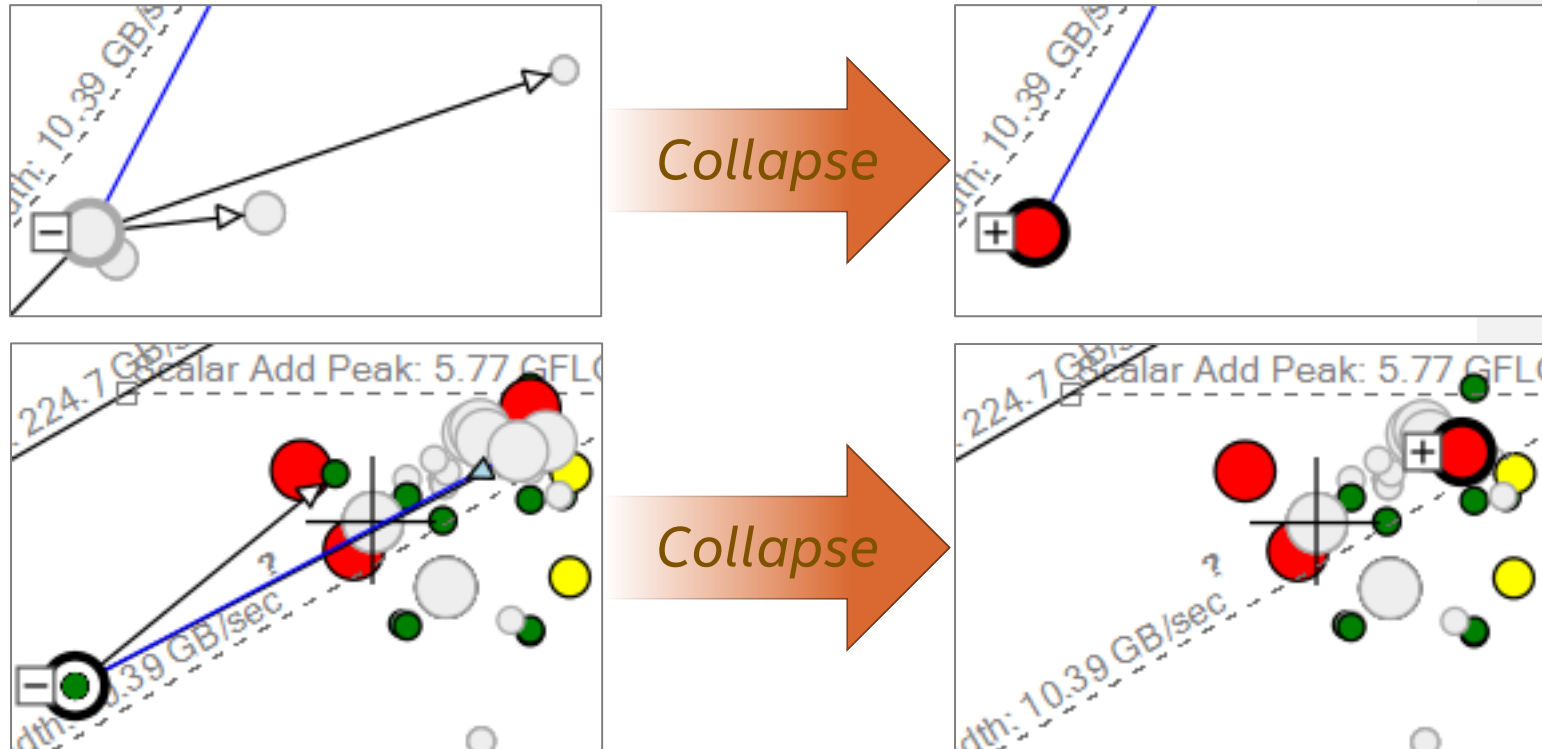
Reading the Roofline with Call Stacks

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.

Memory-level Roofline Model

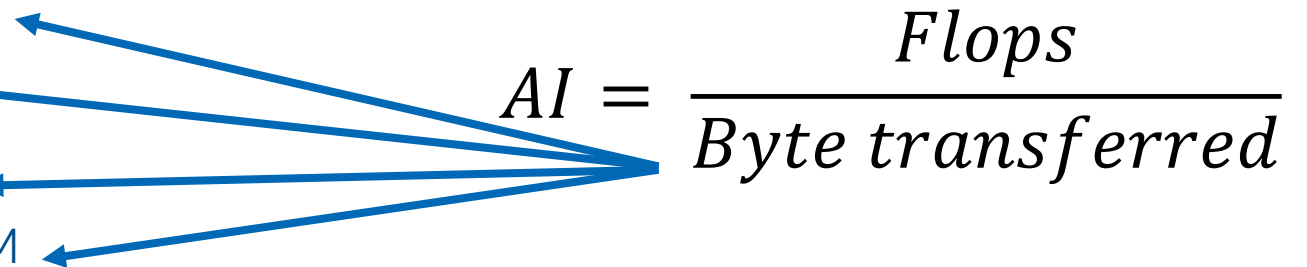
The Roofline Model with Intel® Advisor

- First Implementation: Cache-Aware Roofline Model (CARM)
 - Based on instrumentation
 - 2 runs, one for sampling, timing loops & functions (low overhead), second one for instrumentation
 - Algorithmic version of the Roofline Model; optimization usually doesn't impact AI
 - 😊 Really powerful to characterize an algorithm
 - 😞 Not easy to interpret
- New Implementation: Memory Level roofline (MLR)
 - Based on cache simulation, evaluate the traffic between each memory subsystem (L1/L2/LLC/DRAM)
 - 😊 Much closer to the original Roofline model, provide meaningful information for improvement
 - 😞 Requires more time to run

Memory Level Roofline

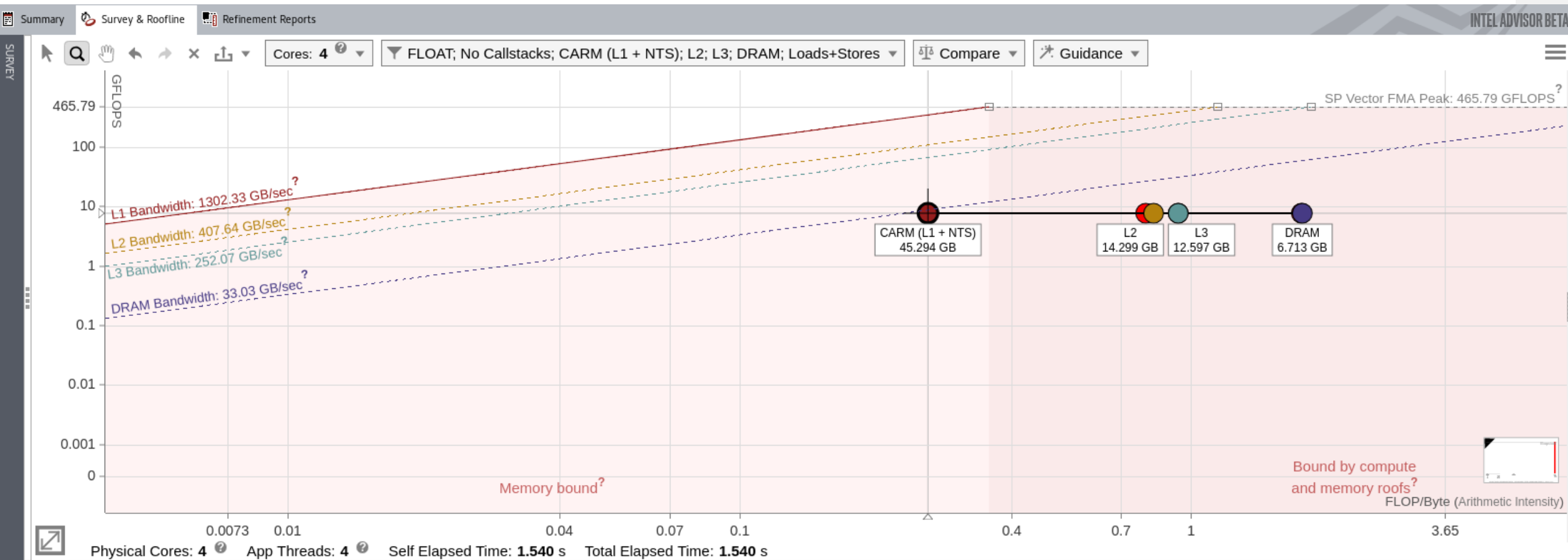
- Single loop generates up to 4 dots
 - Same performance for each dot (it's the same loop) but with different data transfers

- 1st dot comes from CARM (L1)
- 2nd dot comes from traffic L1 <-> L2
- 3rd dot comes from traffic L2 <-> L3
- 4th dot comes from traffic L3 <-> DRAM



- What can we expect?
 - Due to data locality : $AI(L1) = < AI(L2) = < AI(L3) = < AI(DRAM)$
 - **This only applies in general if you do unit-strided access**

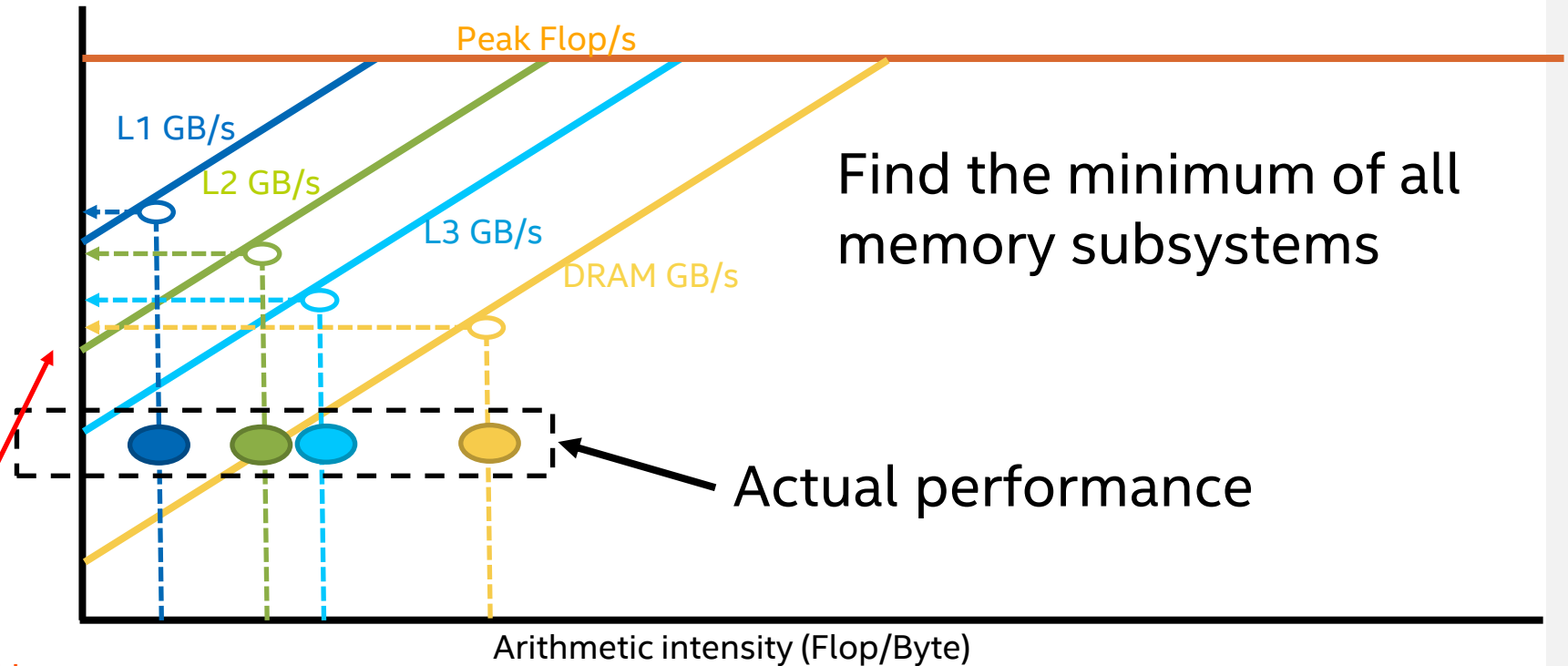
Memory-Level Roofline Model in Intel® Advisor



How to Interpret Your Current Limitation?

- Each dot must be compared to its corresponding roof
- A dot can't break its corresponding roof
- A first idea of potential performance can be achieved by projections

Performance might be limited by DRAM



GUI and Command Line

Get Roofline data using GUI

The image shows the Intel Advisor GUI with the 'Run Roofline' workflow selected. A green box highlights the 'Command line created by GUI' text, which is generated from the GUI settings. The 'Project Properties' dialog box is also shown, with the 'Launch Application' section expanded to show the 'Collect information about Loop Trip Counts' and 'Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage' options checked.

Command line created by GUI

Project Properties - Launch Application

- Collect information about Loop Trip Counts
- Collect information about FLOP, L1 memory traffic, and AVX-512 mask usage
- Collect stacks
- Use MPI launcher

Performance (GFLOPS) Graph

Bandwidth	GFLOPS
L1 Bandwidth: 357.04 GB/sec	~100
L2 Bandwidth: 124.27 GB/sec	~10
L3 Bandwidth: 50.7 GB/sec	~1
Bandwidth: 11.04 GB/sec	~0.1

Summary Table

Metric	Value	Description
Total GFLOPS	1,06375	Giga Floating-point Operations Per Second
Self AI	0,12500	Self Arithmetic Intensity - Ratio Of Total Floating-Point Operations To Self L1 Transferred Bytes
Total AI	0,12500	Total AI - Total Arithmetic Intensity - Ratio Of Total Floating-Point Operations To Total L1 Transferred Bytes
Self GFLOP	0,41943	Giga Floating-Point Operations, Not Including GFLOP For Functions Called In The Loop Or Function
Total GFLOP	0,41943	Giga Floating-Point Operations Of Function/Loop And Its Callees
Self FLOP Per Iteration	32	Floating-point Operations Per Loop Iteration

Get roofline data using command line. Example:

- Roofline collection runs executable twice implicitly: survey and tripcounts

```
advisor -collect roofline -project-dir <dir> -- <app> <params>
```

- Alternative method collects survey and tripcounts explicitly, required for MPI!

```
advisor -collect survey -project-dir <dir> -- <app> <params>
```

```
advisor -collect tripcounts -flop -project-dir <dir> -- <app> <params>
```

Additional flags for tripcounts, e.g.: **-stacks**, **-enable-cache-simulation** (see **-help collect**)

- Analyze roofline and other Advisor data in the GUI

```
advisor-gui <dir>
```

Resources

References

Roofline model proposed by Williams, Waterman, Patterson:

<https://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-134.html>

“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>

Advisor Resources

Intel® Advisor

- [Product page](#) – overview, features, FAQs...
- [What's New?](#)
- Training materials – [Cookbook](#), [User Guide](#), [Tutorials](#)
- [Support Forum](#)
- [Priority Support](#) - Online Service Center

Additional Analysis Tools

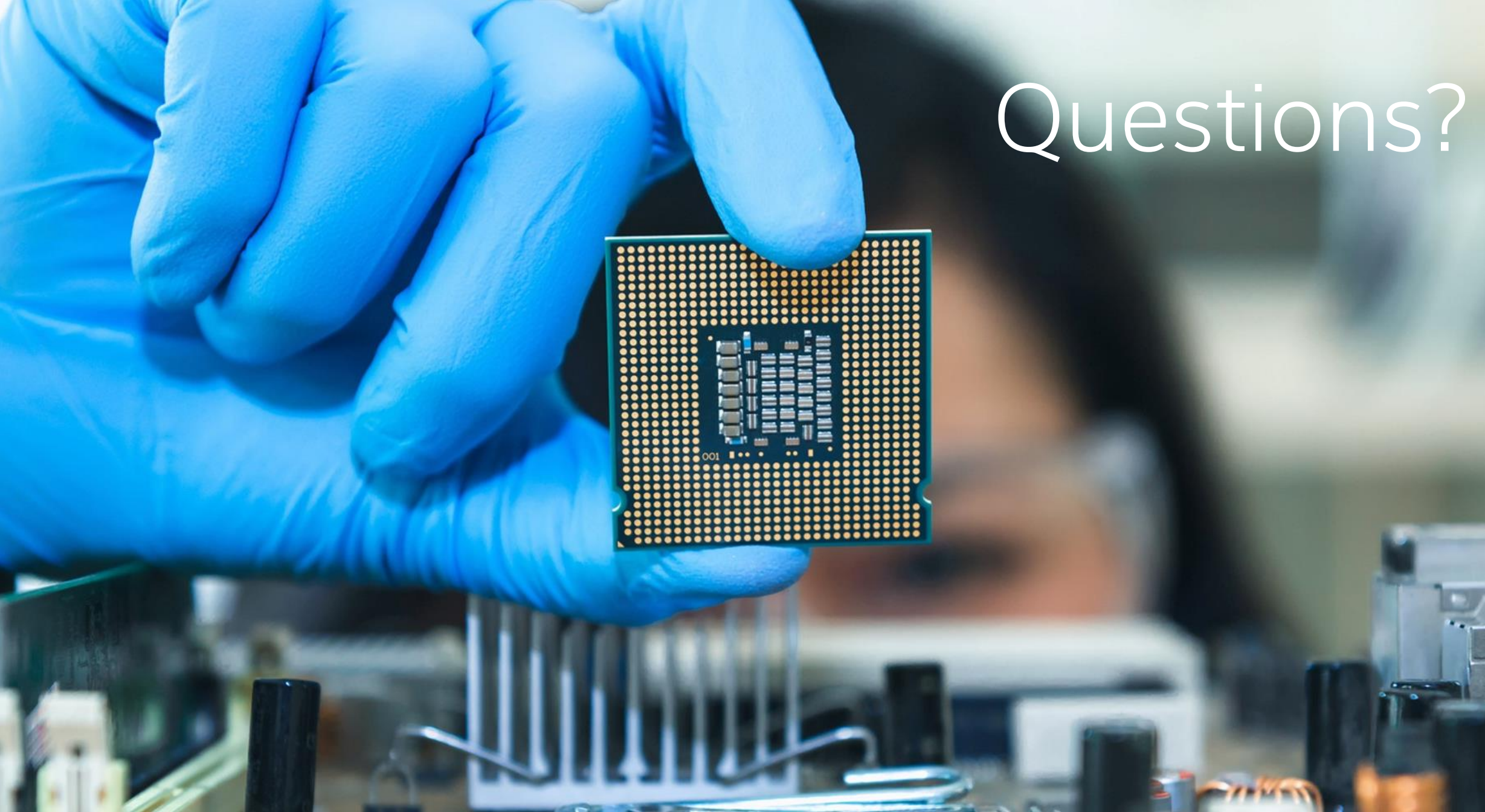
- [Intel® VTune™ Profiler](#) – performance profiler
- [Intel® Inspector](#) – memory and thread checker/ debugger
- [Intel® Trace Analyzer and Collector](#) - MPI Analyzer and Profiler

All Development Products

- [Intel® oneAPI Toolkits](#)



Questions?



Notices & Disclaimers

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

intel®

Backup

Running Intel Advisor with MPI

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

```
mpirun -n 27 advisor -collect survey-project-dir <dir> <app>
```

```
mpirun -n 13 <app> \  
      : -n 1 advisor -collect survey -project-dir <dir> <app> \  
      : -n 13 <app>
```

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

```
mpirun -gtool "advisor -collect survey -project-dir <dir> :1,5,14" \  
      -n 27 <app>
```

- or using the environment variable I_MPI_GTOOL:

```
export I_MPI_GTOOL="advisor -collect survey --project-dir <dir> :1,5,14"  
mpirun -n 27 <executable>
```

Non-Intel Compilers

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