aiXcelerate, RWTH Aachen, December 5th 2022

Intel® Advisor

Vectorization and Roofline Analysis

Heinrich.Bockhorst@Intel.com

intel®

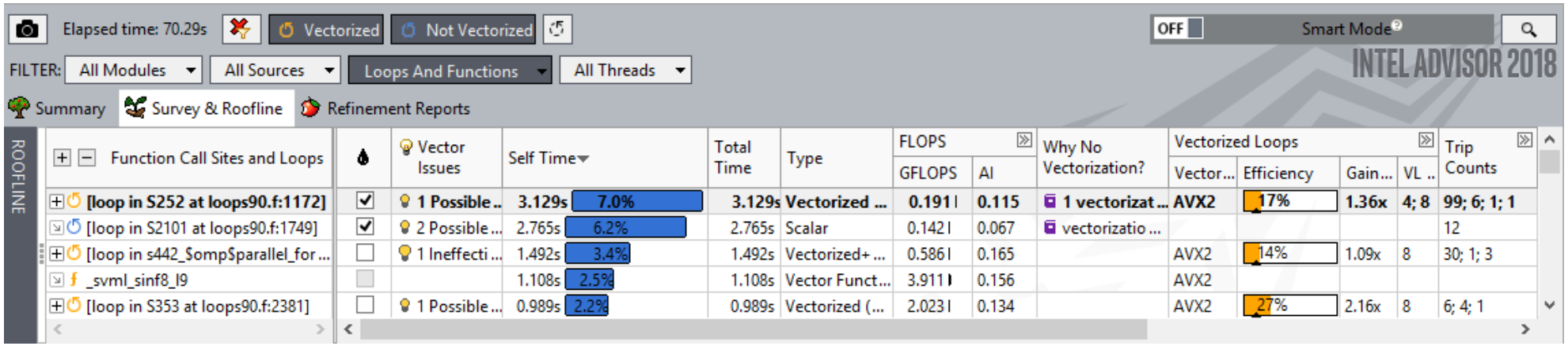
Agenda

- Intel® Advisor – Vectorization analysis
- Roofline analysis
- Roofline analysis – how to generate

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



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

Part of oneAPI Base Toolkit

software.intel.com/advisor

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 shou vectorization and/or threading parallelism?

Intel Advisor XE 2016

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

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

Function Call Sites and Loops	Vector Issues	Self Time	Total Time	Trip Counts	Loop Type	Why No Vectorization?	Vectorized Loops: Vecto...	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 loopstl.cpp:7900 in vas_]		0.170s	0.170s	500	Scalar	vectorization possible but...			4
[loop loopstl.cpp:3509 in s2...	**1 High vector register ...**	**0.160s**	**0.160s**	**12**	**Expand**	**Expand**	**AVX**	**~69%**	**8**
[loop loopstl.cpp:3891 in s279_]	2 Ineffective peeled/rem...	0.150s	0.150s	125; 4	Expand	Expand	AVX	~96%	8
[loop loopstl.cpp:6249 in s414_]		0.150s	0.150s	12	Expand	Expand	AVX	~100%	4
[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?

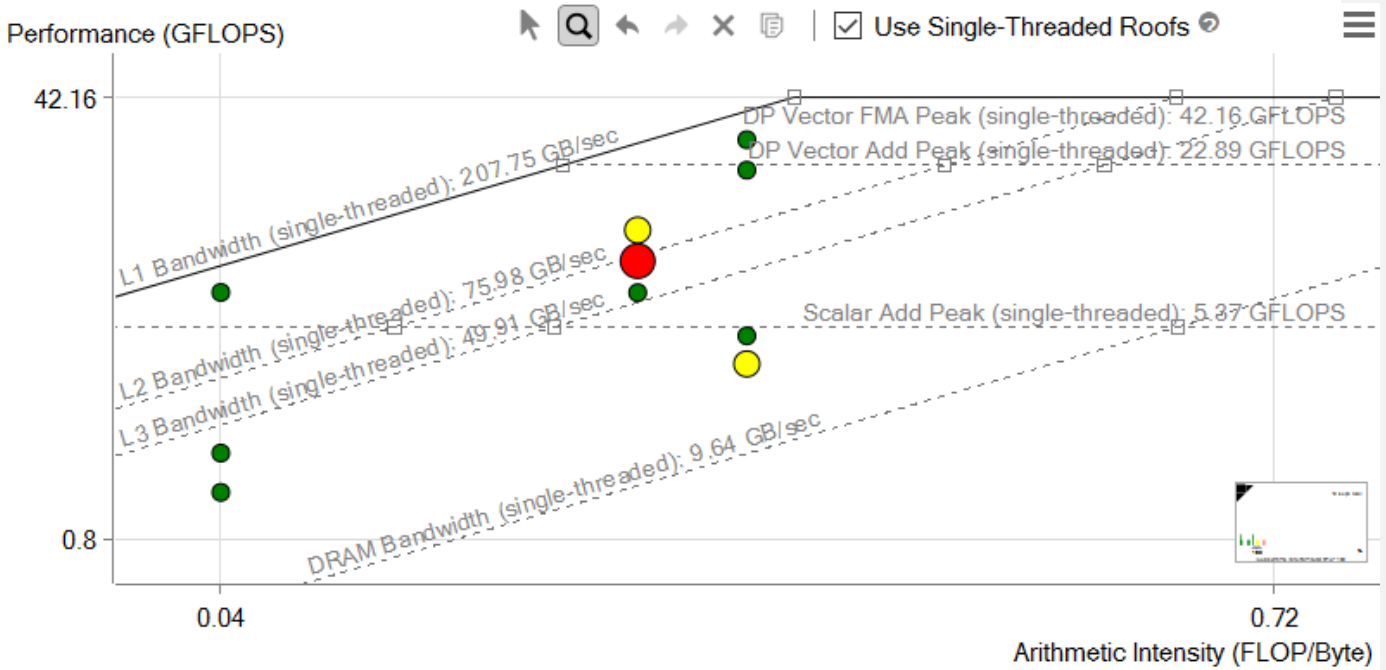
How efficient is the code?

Get Faster Code Faster!

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

$$\text{Gflop/s} = \min \begin{cases} \textit{Platform PEAK} \\ \textit{Platform BW} * \textit{AI} \end{cases}$$

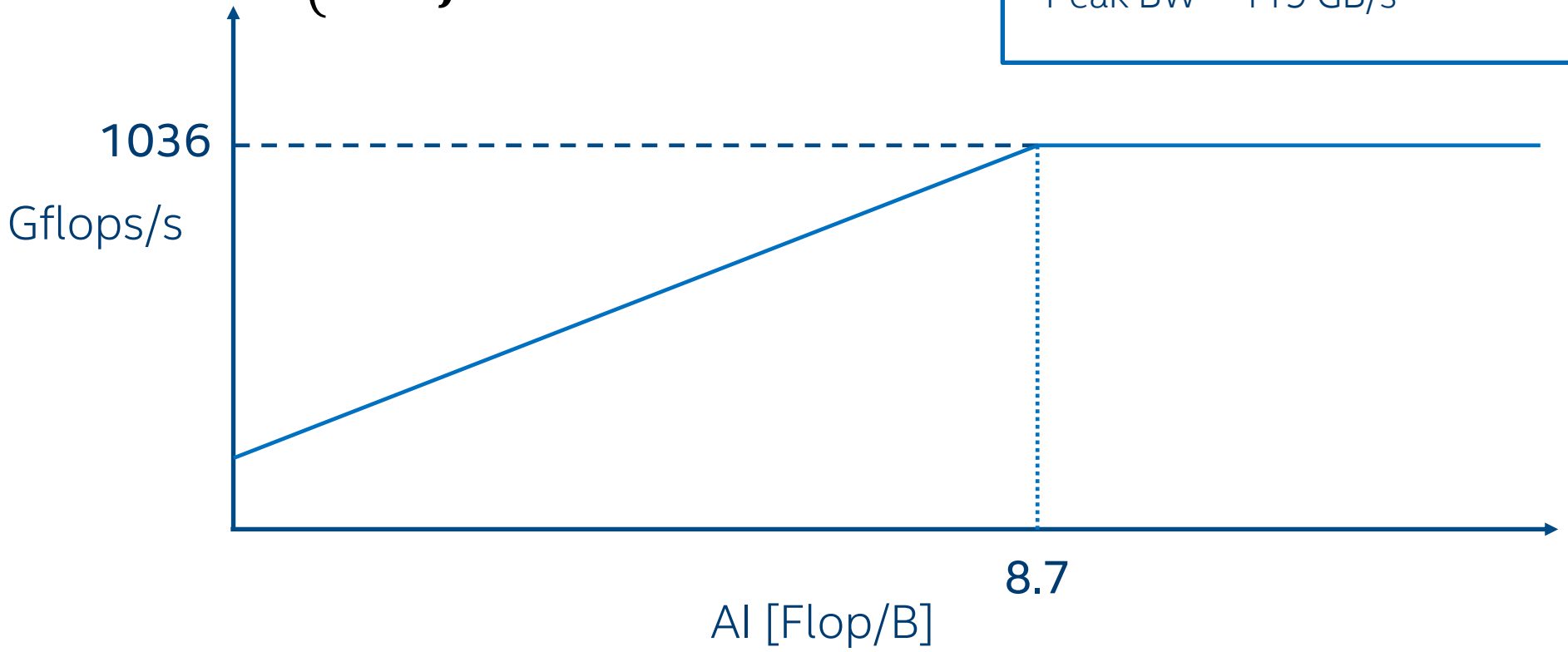
Arithmetic Intensity (Flops/Bytes)

DRAWING THE ROOFLINE

Defining the speed of light

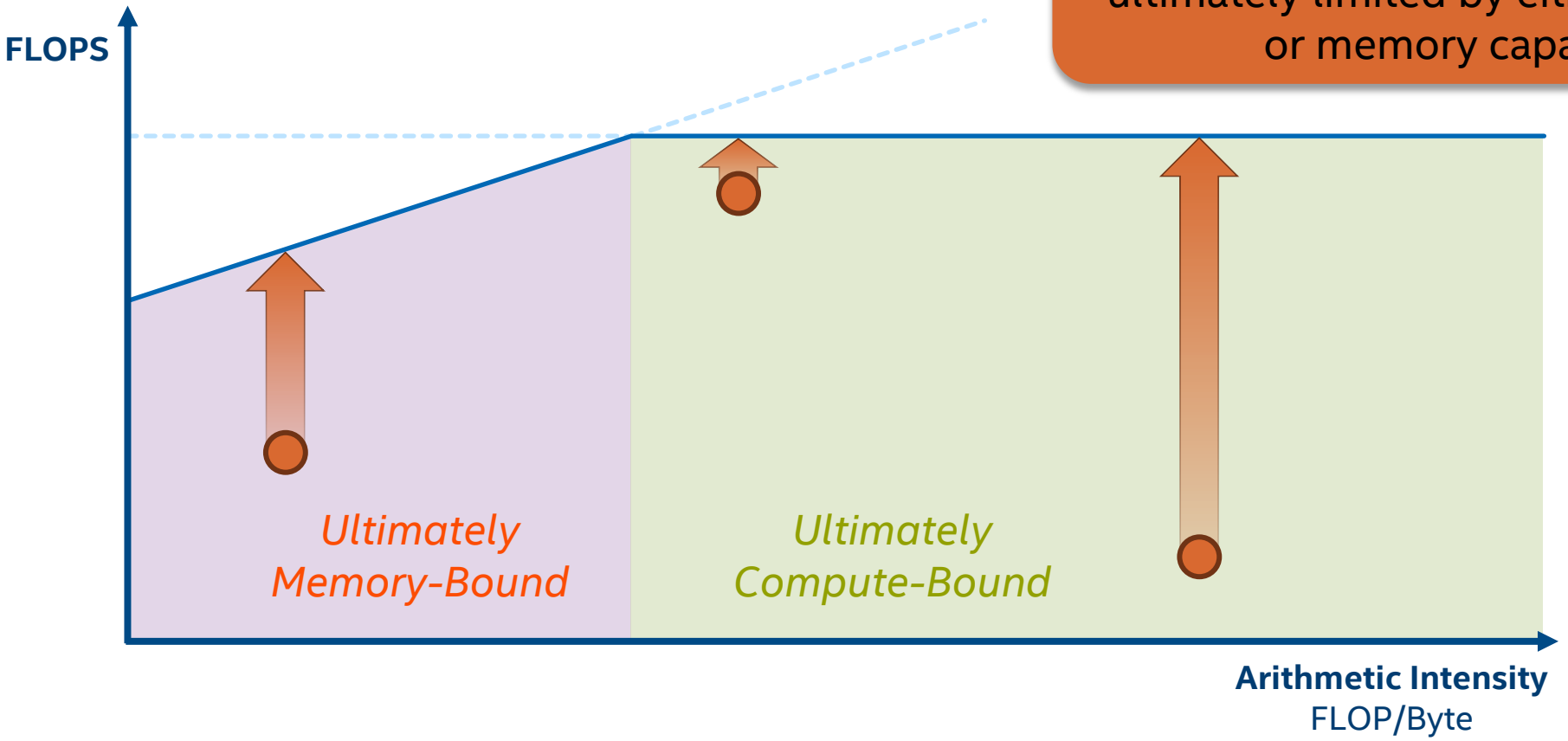
$$\text{Gflop/s} = \min \begin{cases} \textit{Platform PEAK} \\ \textit{Platform BW} * \textit{AI} \end{cases}$$

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



Ultimate Performance Limits

Performance cannot exceed the machine's capabilities, so each loop is ultimately limited by either compute or memory capacity.

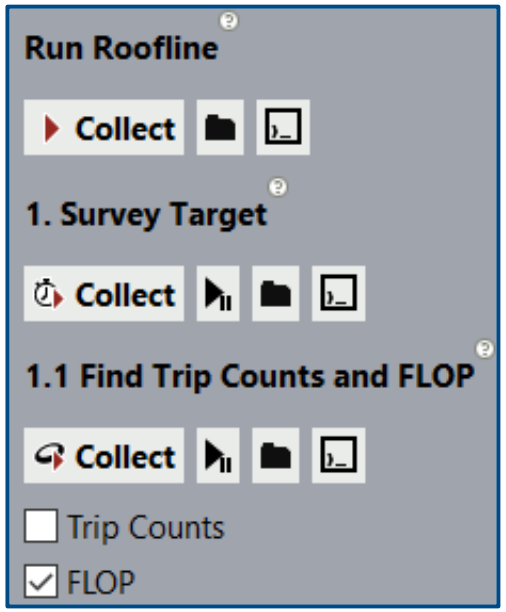


Roofline Metrics

- 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.

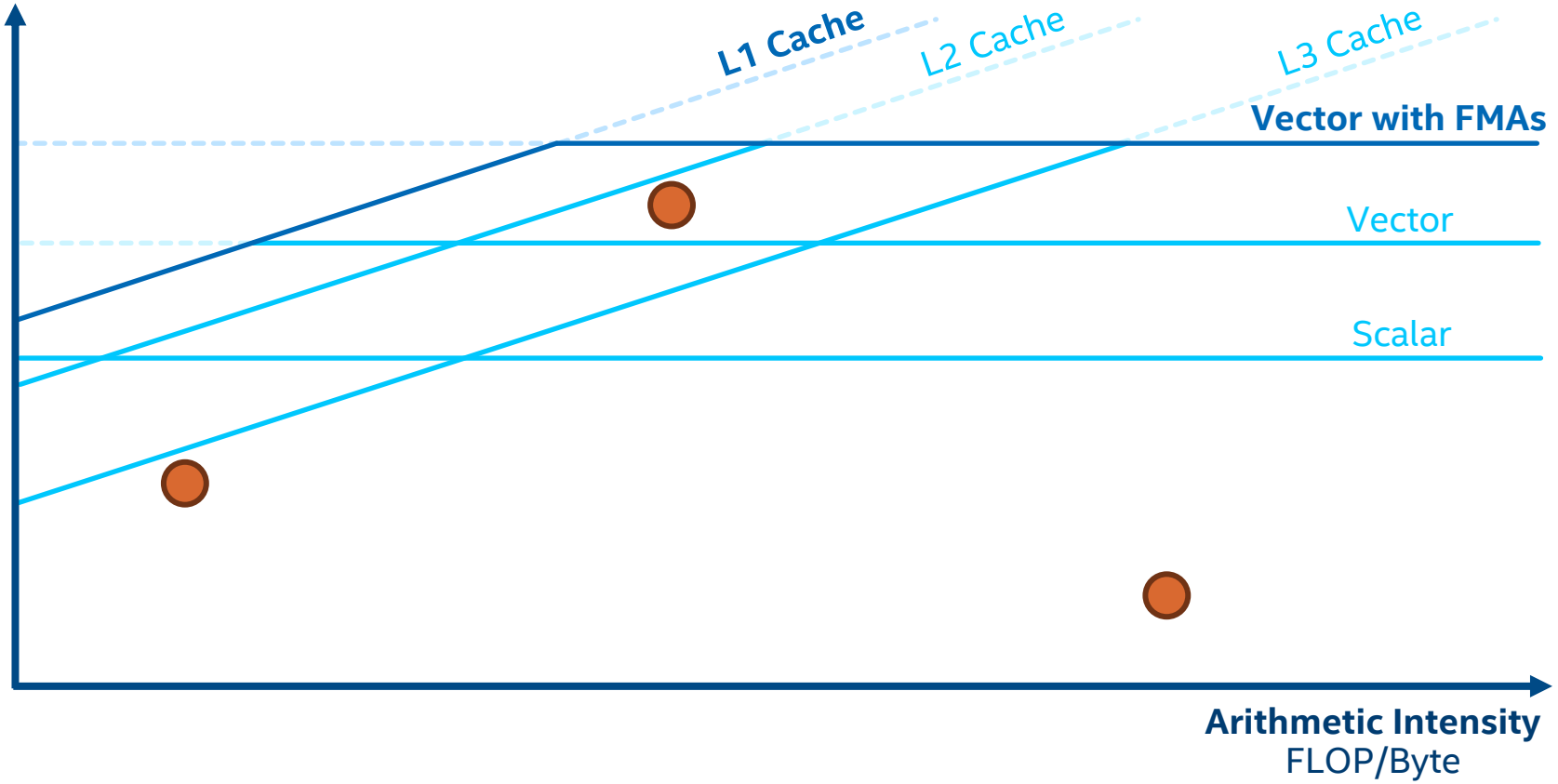


Shortcut to run Survey followed by Trip Counts + FLOPs

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.

Sub-Roofs and Current Limits

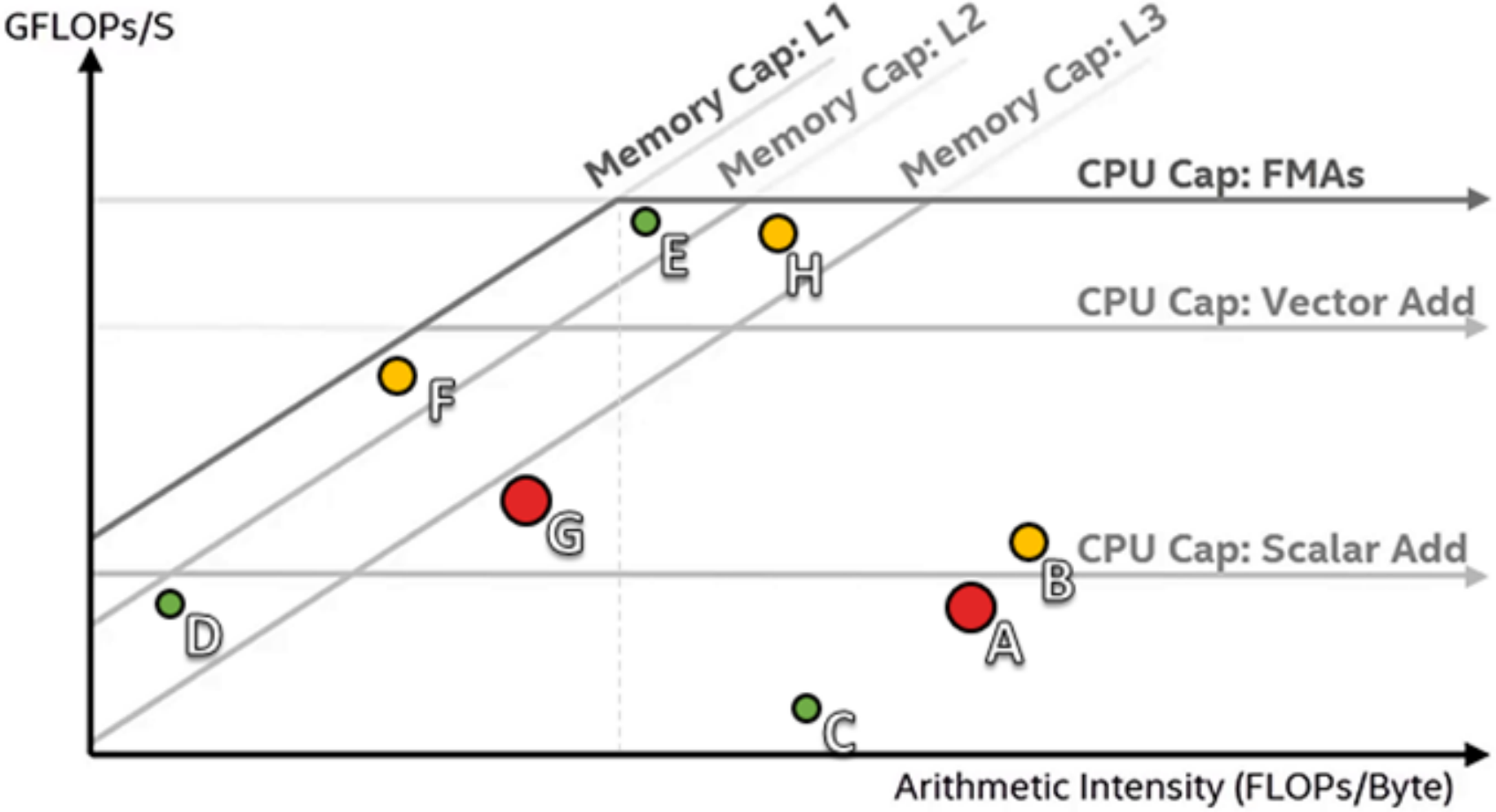


Additional roofs can be plotted for specific computation types or cache levels.

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

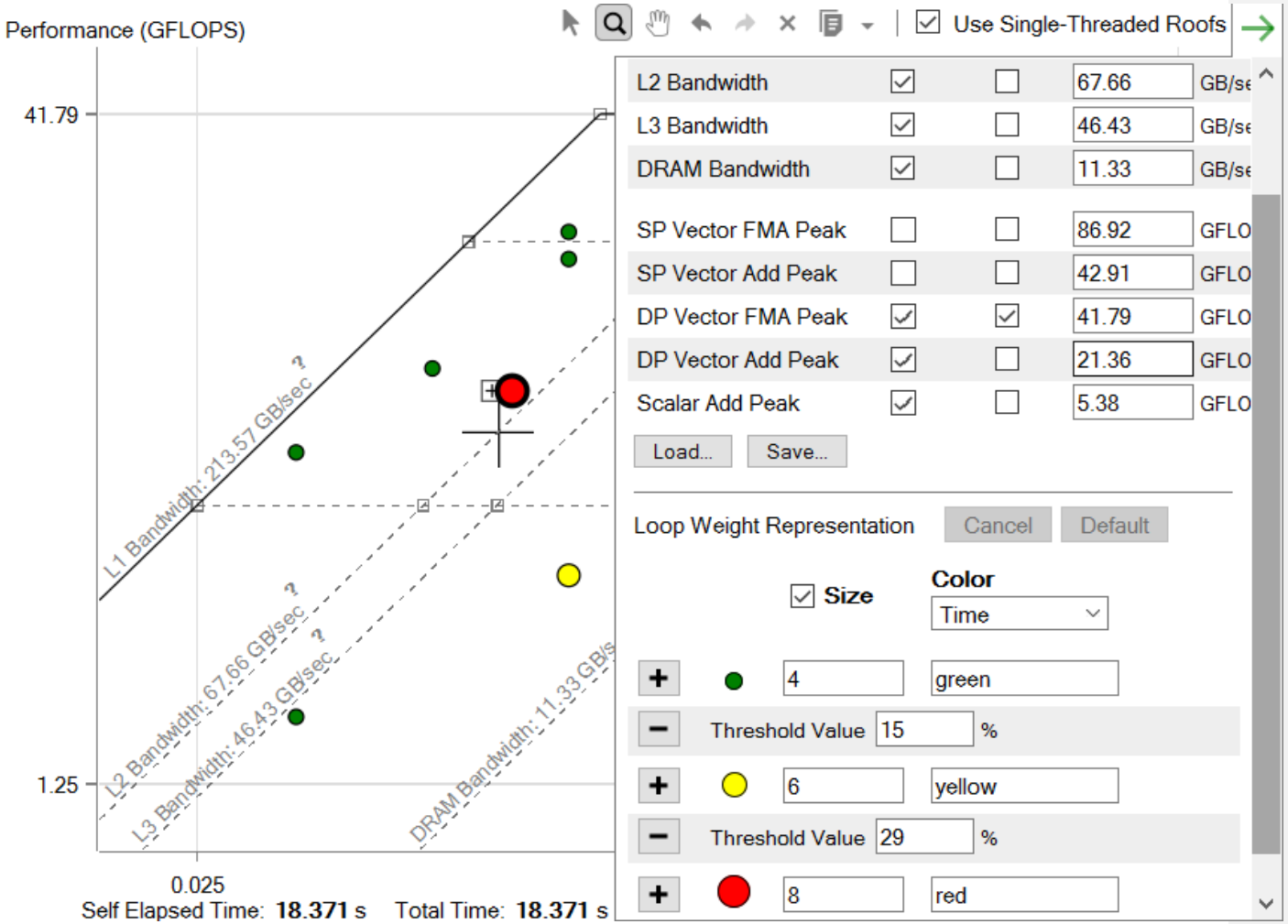
Identifying Good Optimization Candidates

- Focus optimization effort wher makes the most difference.
  - Large, red loops have the most impact.
  - Loops far from the upper roofs have more room to improve.



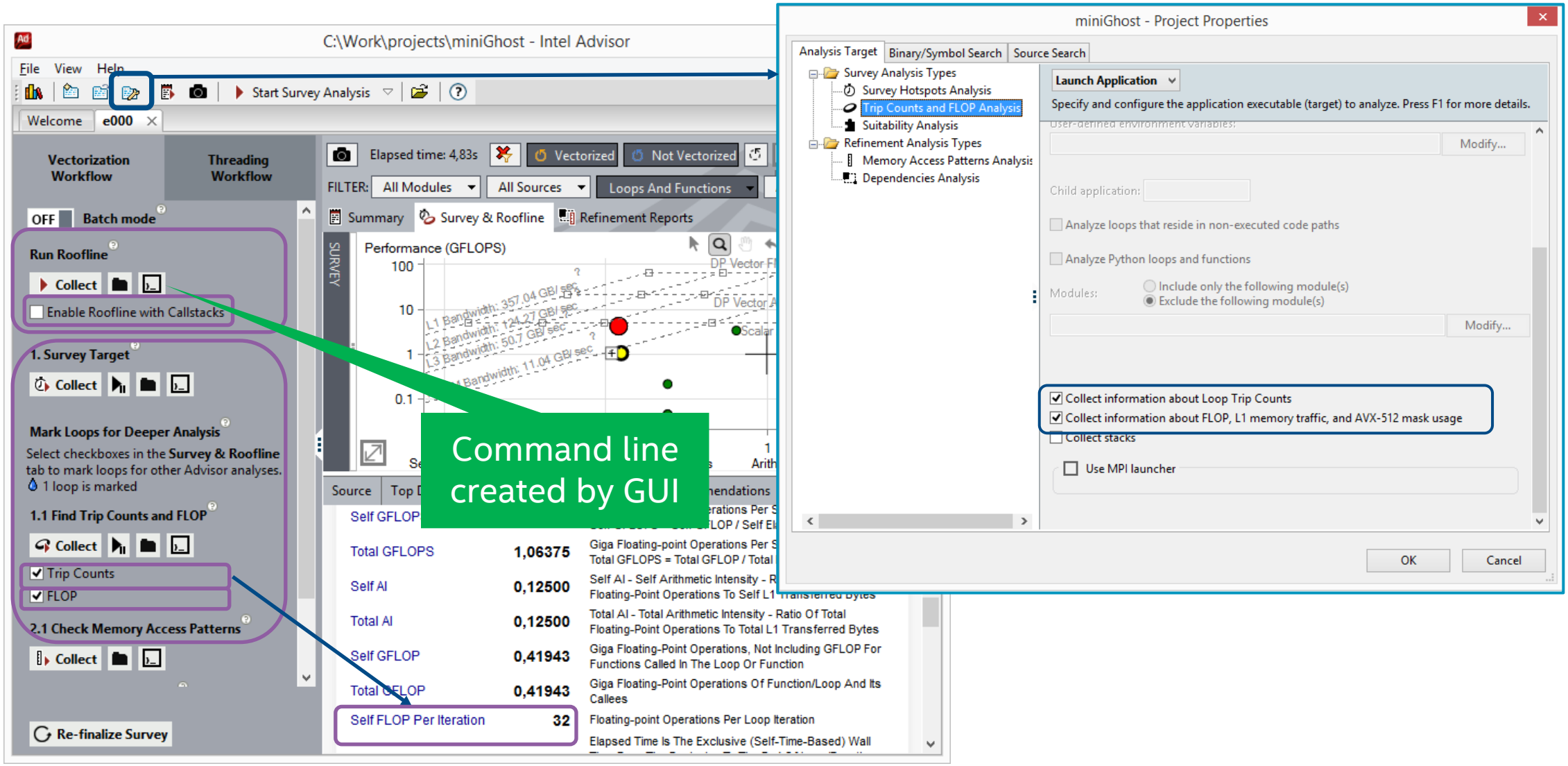
The Intel® Advisor Roofline Interface

- 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.



GUI and Command Line

Get Roofline data using **GUI**



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

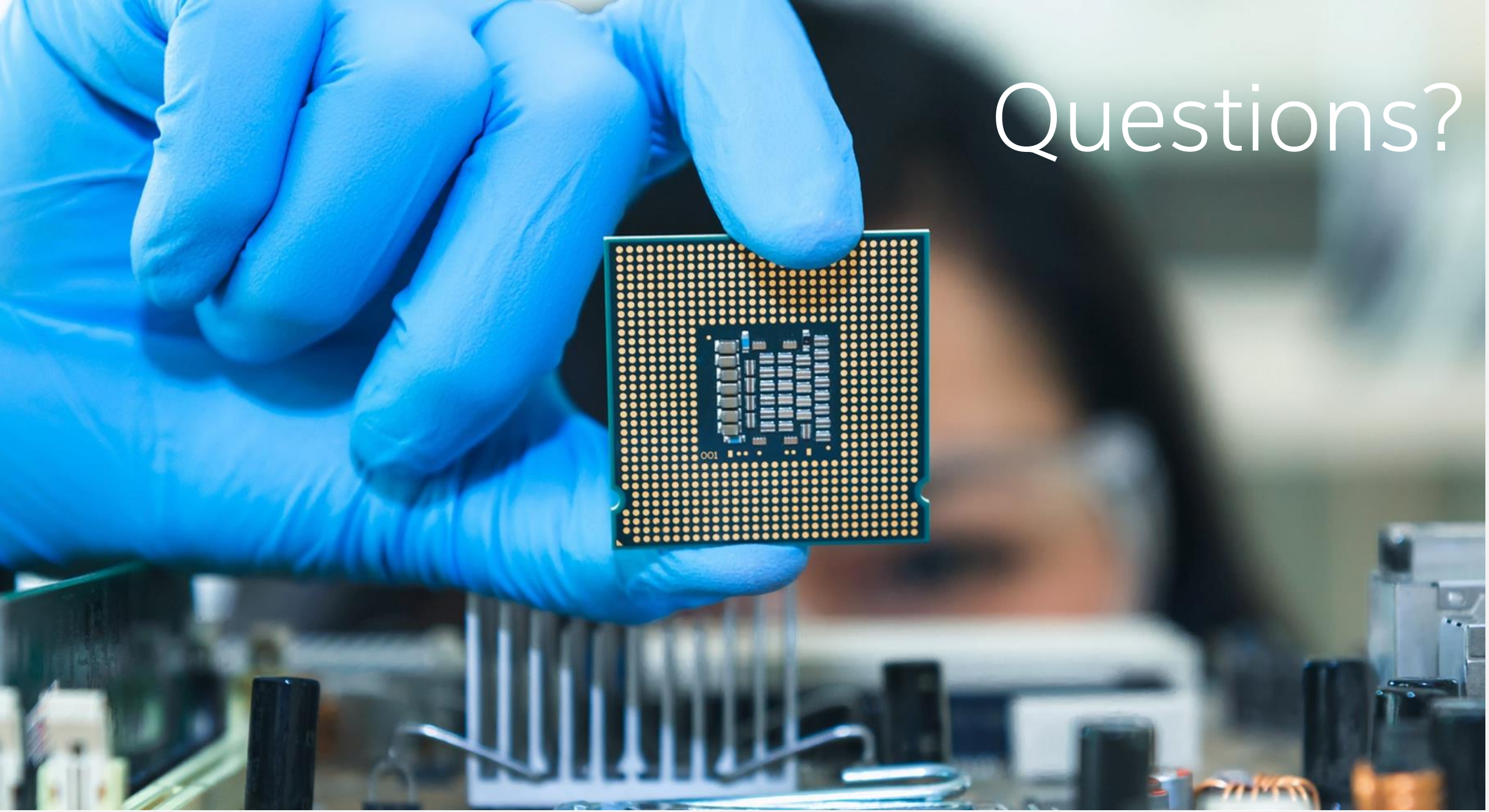
- Product page – overview, features, FAQs…
- What's New?
- Training materials – Cookbook, User Guide, Tutorials
- Support Forum
- Priority Support - Online Service Center

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®

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>
```

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.

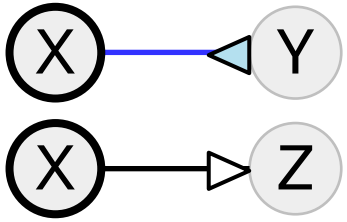
```
for (int i = 0; i < 10; i++)
{
  X = i * 24 + 179.4 - i;
  Y = (i + 18) / 72.8;
  foobar();
  for (int j = 0; j < 3; j++)
  {
    Z = i * j + 7;
  }
}
```

Self

Total

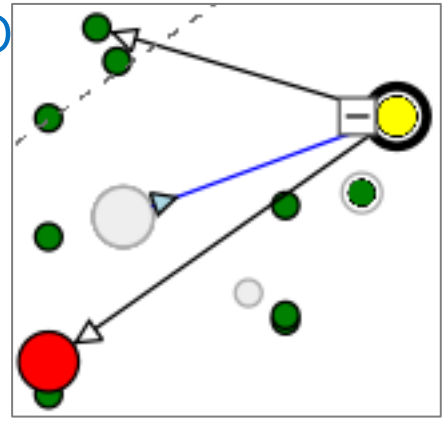
Reading the Roofline with Call Stacks

Visualizing the Call Chain

- Arrows indicate relationships between do
- 

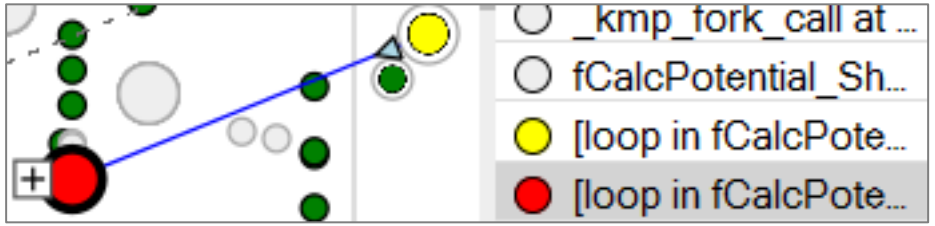
X is called directly by Y.
-

X directly calls Z

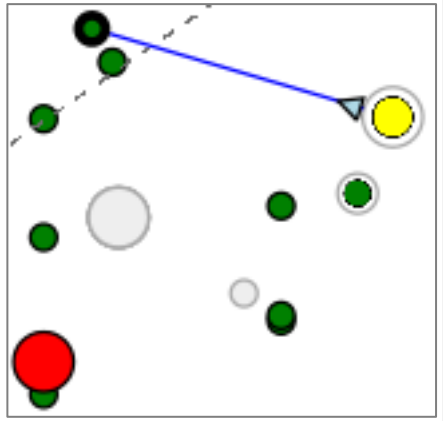


The selected yellow dot was called by the gray dot, and it calls the red and green dots.

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



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



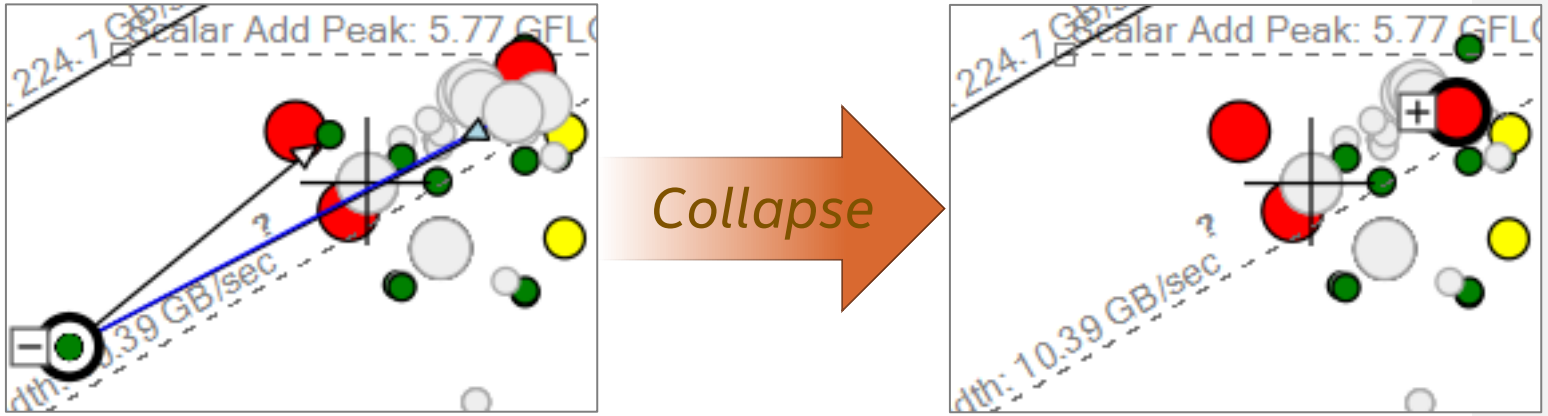
Reading the Roofline with Call Stacks

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

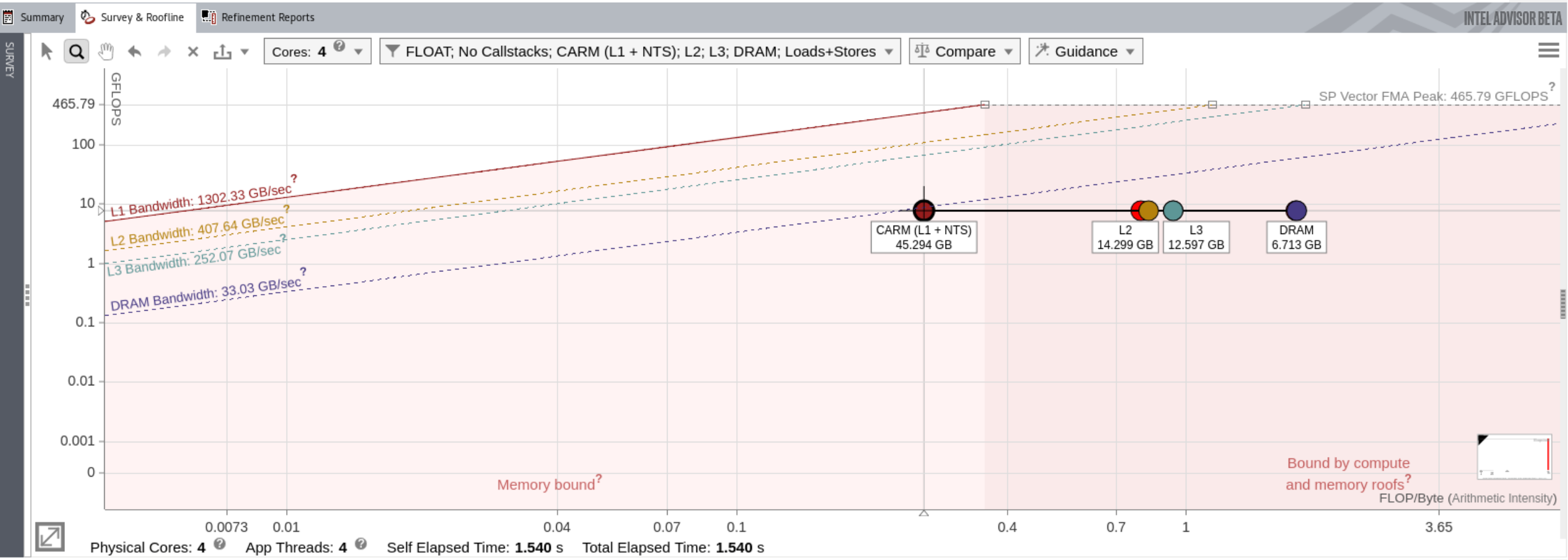
- Single loop generates up to 4 dots
  - Same performance for each dot (it's the same loop) but with different data transfers

  - 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

$$AI = \frac{Flops}{Byte\ transferred}$$

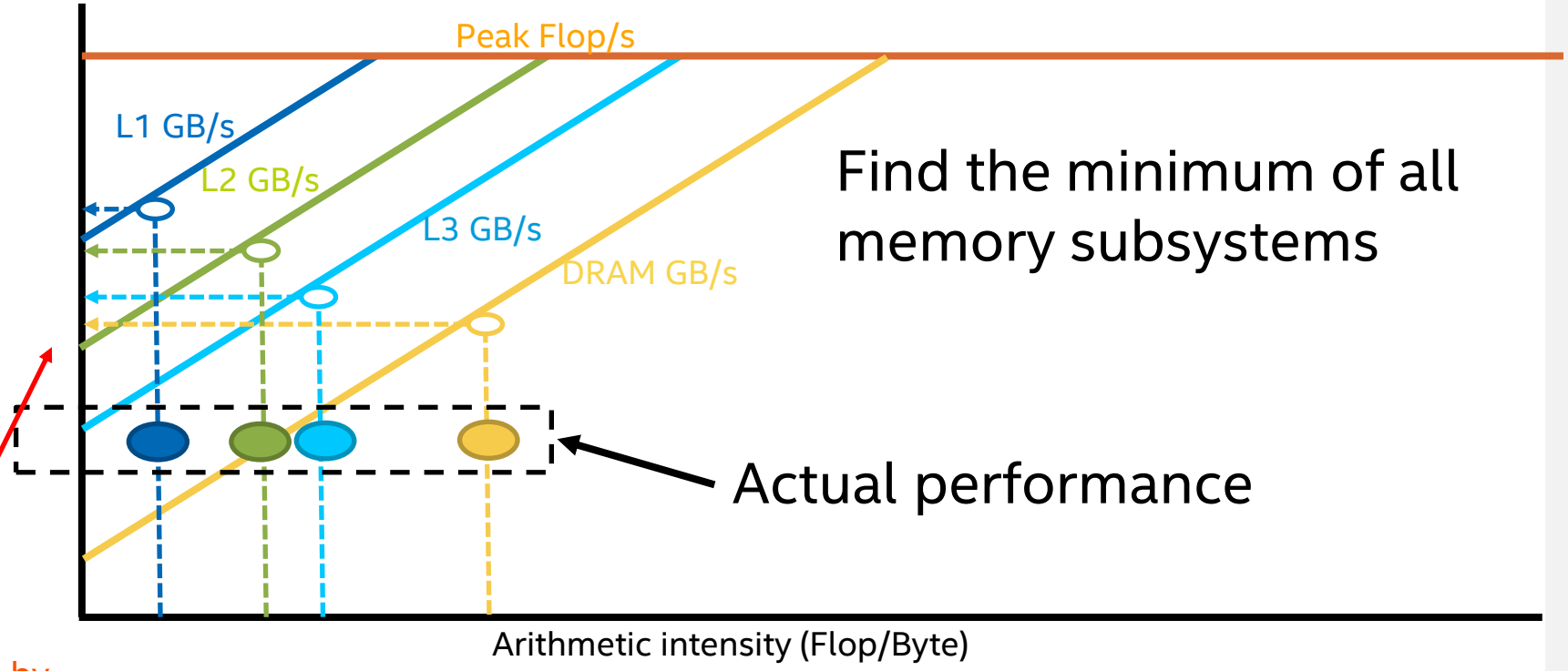
- What can we expect?
  - Due to data locality : AI(L1) =< AI(L2) =< AI(L3) =< AI(DRAM)
  - This only applies in general if you do unit-strided access

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

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