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

Intel® Advisor

Vectorization and Roofline Analysis

Klaus-Dieter.Oertel@intel.com

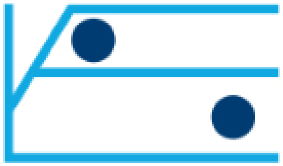
intel®

Intel® Advisor for High Performance Code Design

Rich Set of Capabilities

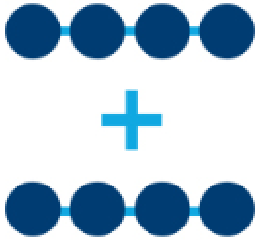
Offload Modelling

Design offload strategy and model performance on GPU.



Roofline Analysis

Optimize your application for memory and compute.

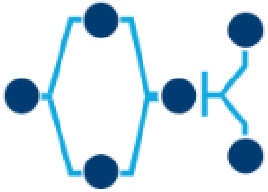


Vectorization Optimization

Enable more vector parallelism and improve its efficiency.

Thread Prototyping

Model, tune, and test multiple threading designs.



Build Heterogeneous Algorithms

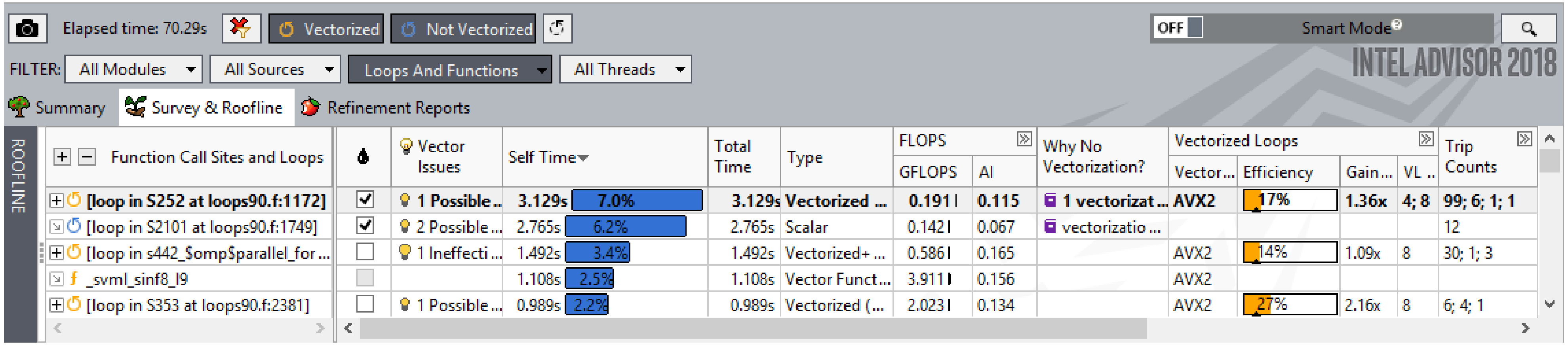
Create and analyze data flow and dependency computation graphs.

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



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

Part of oneAPI Base Toolkit

software.intel.com/advisor

Amdahl's law

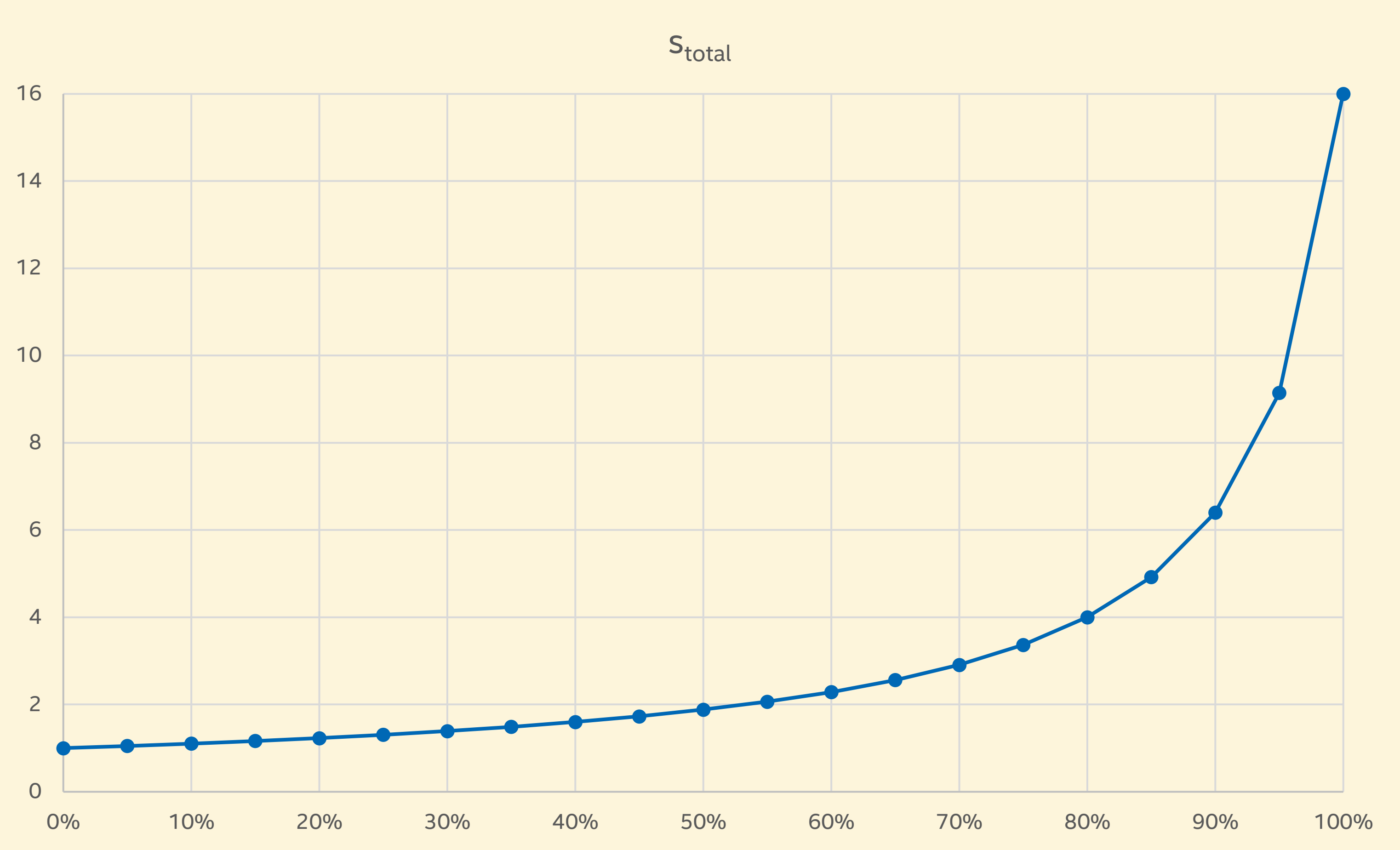
$$S_{total} = \frac{100\%}{(100\% - p) + \frac{p}{s_p}}$$

S = speedup (in parallelized part or total)

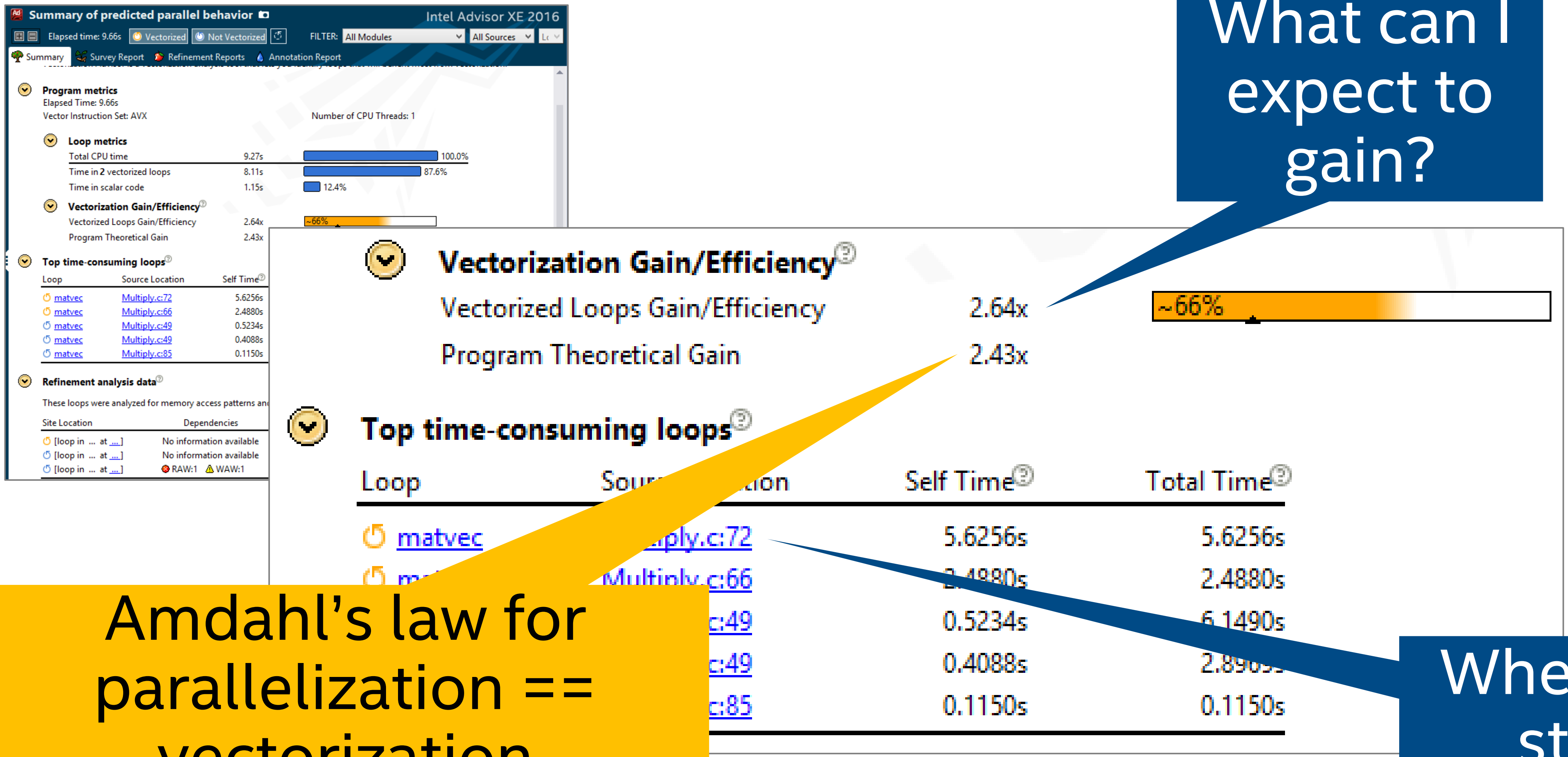
P = proportion of execution time that benefits from parallelization

Example: P=80%, s_p=16 [AVX-512] => 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?

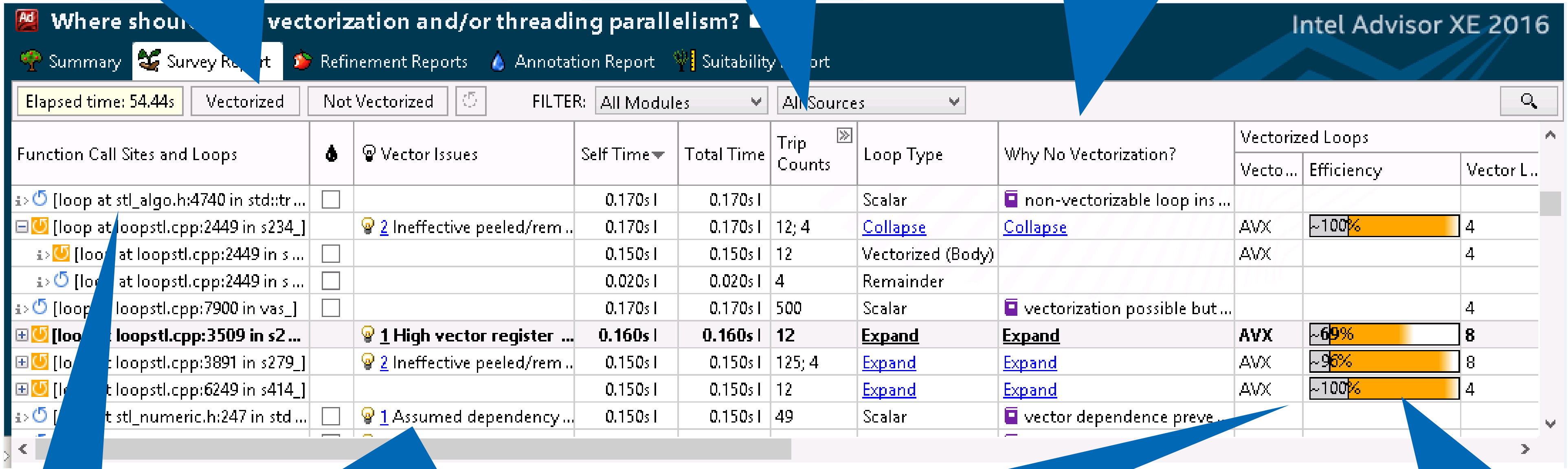
The Right Data At Your Fingertips

Get all the data you need for high impact vectorization

Filter by which loops are vectorized!

Trip Counts

What prevents vectorization?



Focus on hot loops

What vectorization issues do I have?

Which Vector instructions are being used?

How efficient is the code?

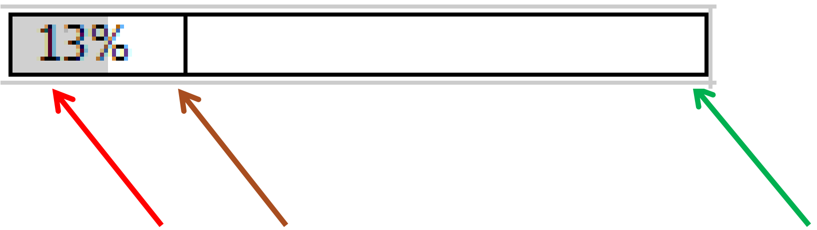
Get Faster Code Faster!

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



Achieved Efficiency

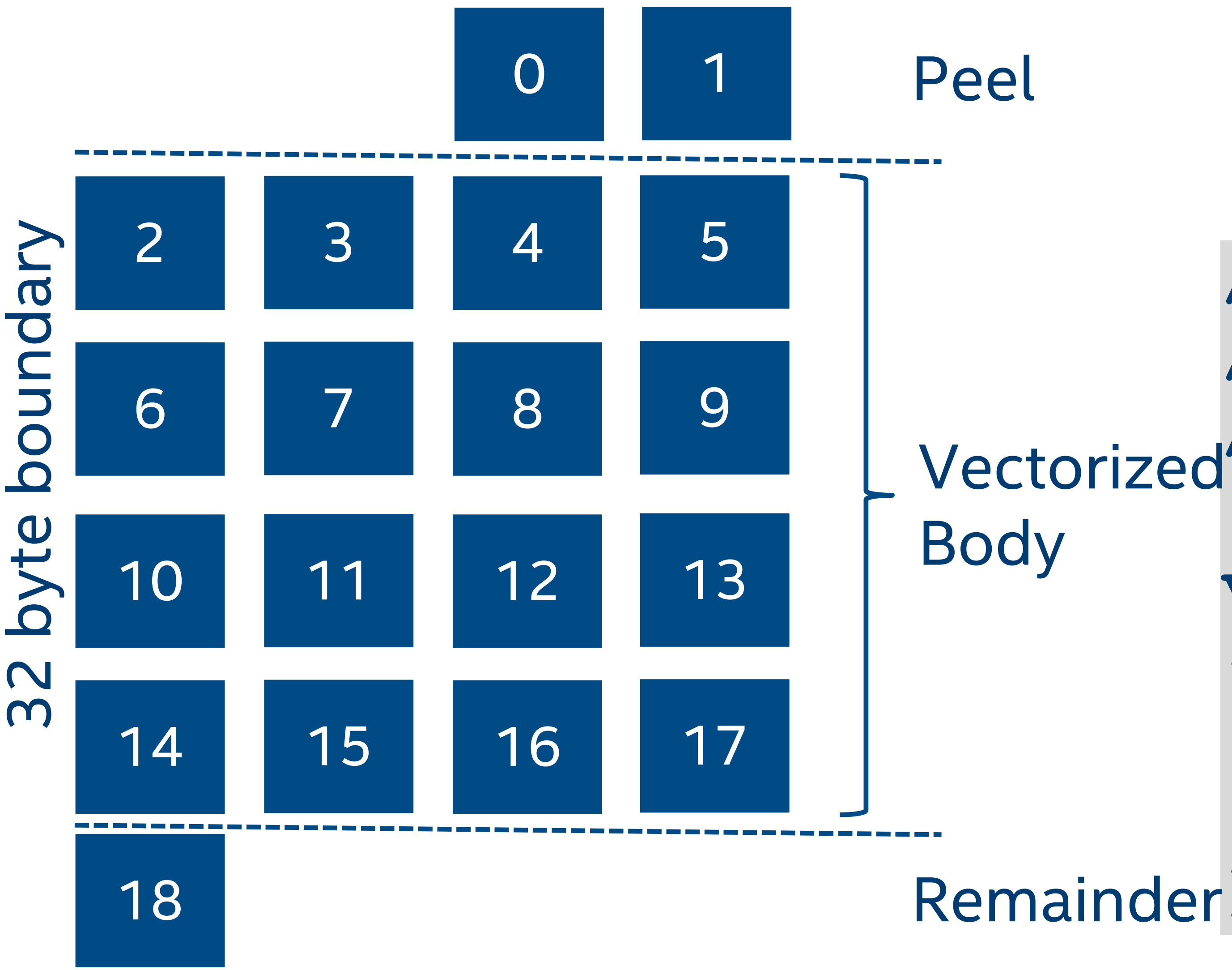
Original (scalar) code efficiency. Corresponds to 1x speed-up.

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

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…

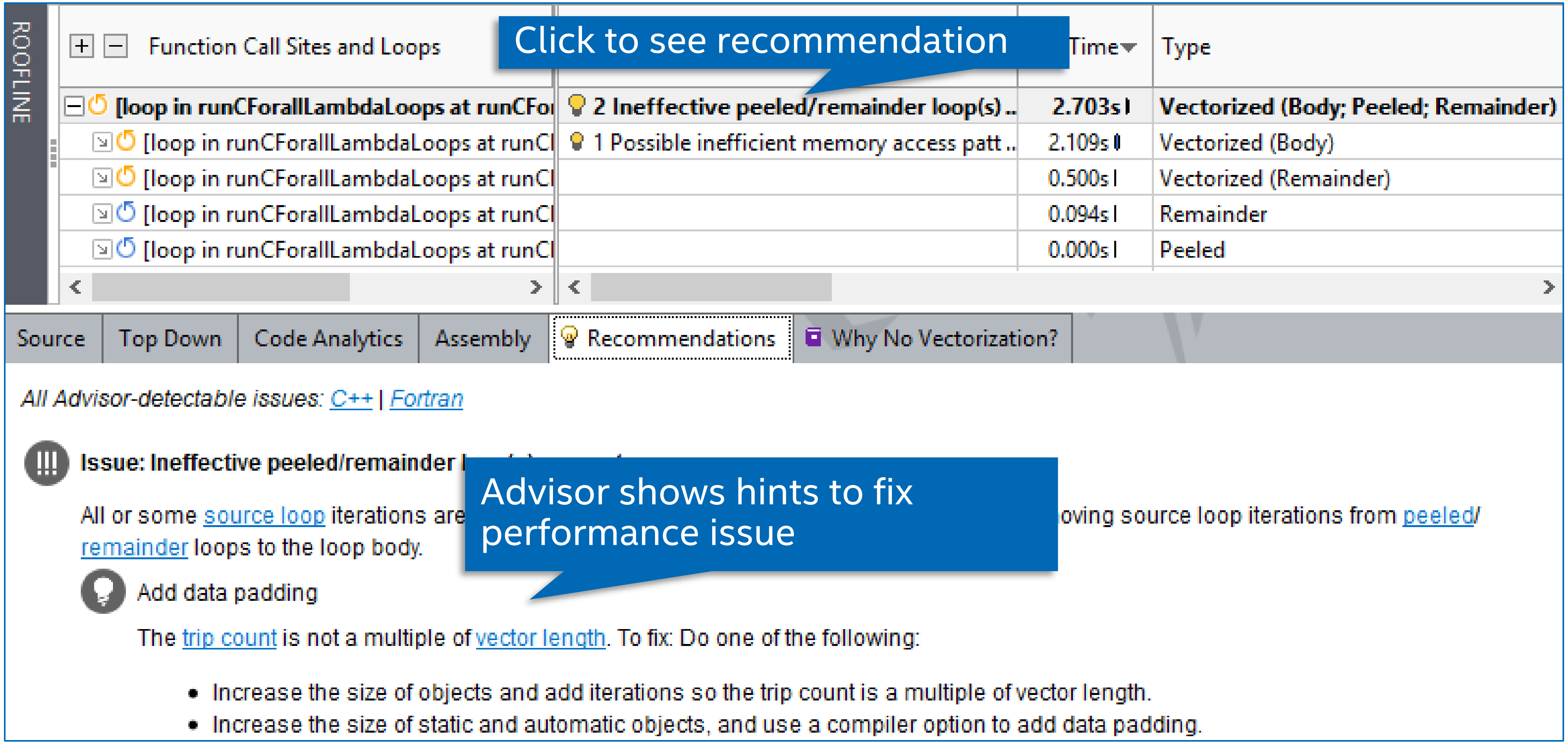
- Optional peel part
  - Used for the unaligned references in your loop. Uses Scalar or slower vector.
- Main vector body
  - Fastest among the three!
- Remainder part
  - Due to the number of iterations (trip count) not being divisible by vector length. Uses Scalar or slower vector.
- Larger vector 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

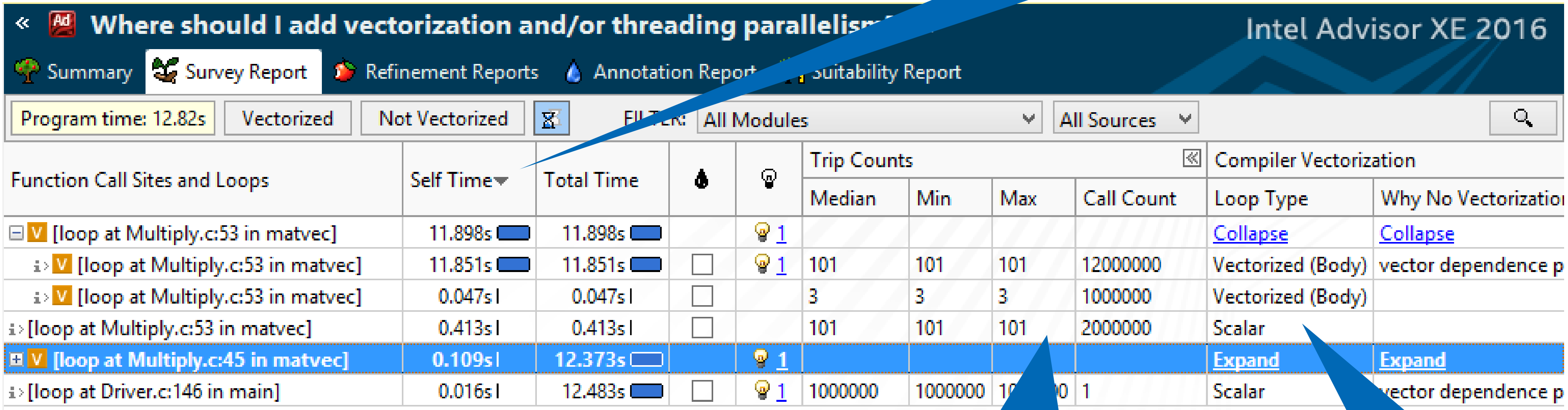
Intel® Advisor – Vectorization Advisor

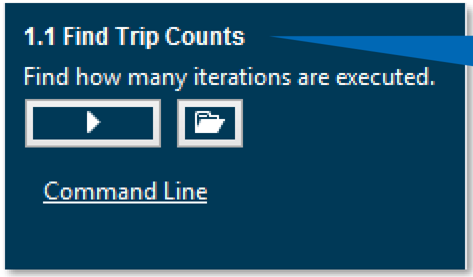


Critical Data Made Easy

Loop Trip Counts

Knowing the time spent in a loop is not enough!





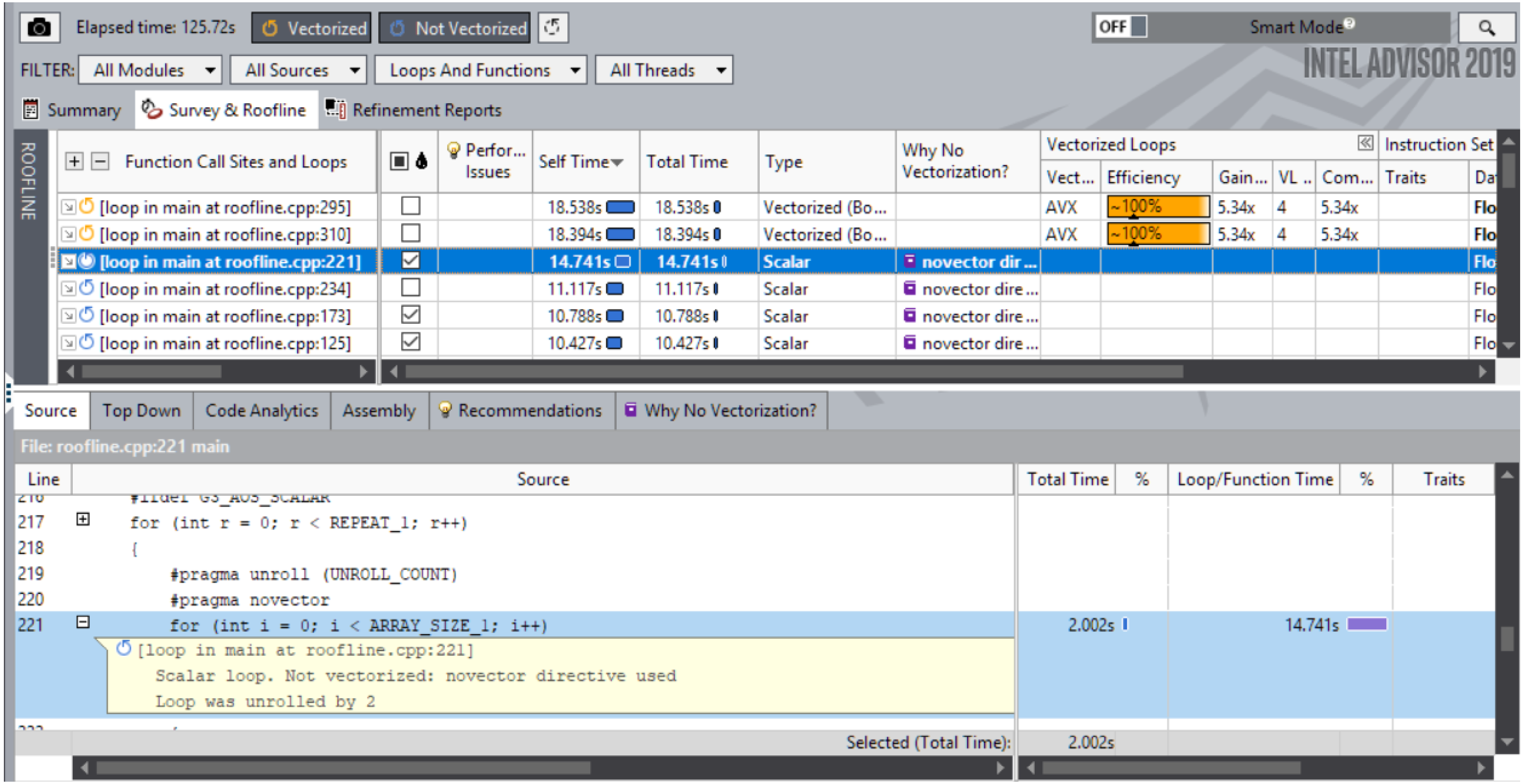
Check actual trip counts

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



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.

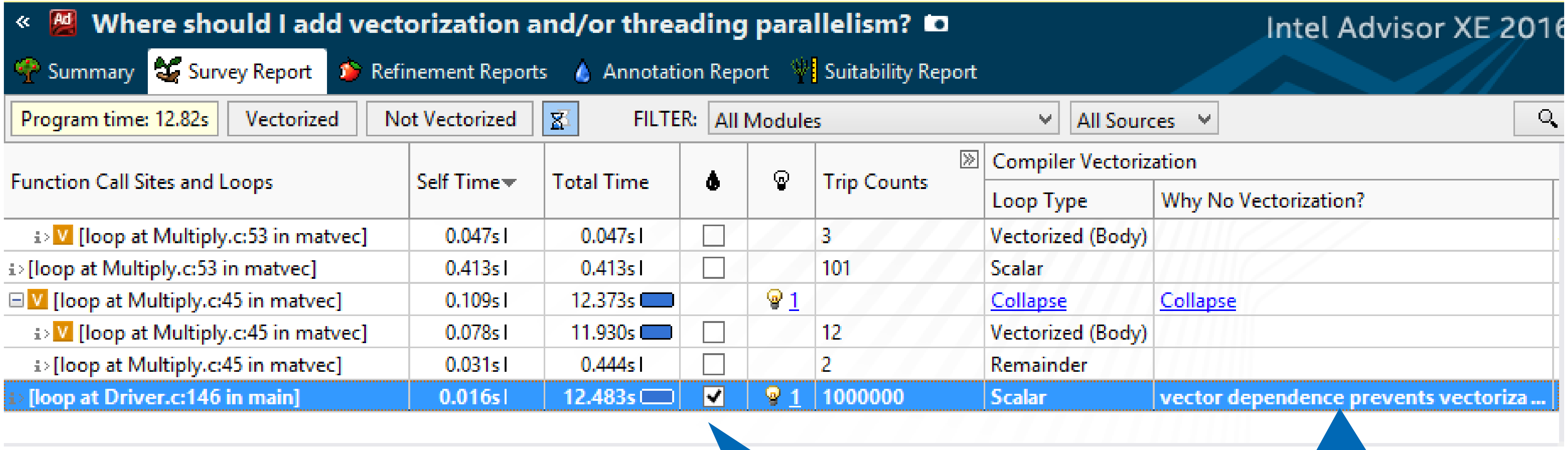
ICL/ICC/ICPC Directive	IFORT Directive	Outcome
#pragma simd or #pragma omp simd	!DIR$ SIMD or !$OMP SIMD	Ignores all dependencies in the loop
#pragma ivdep	!DIR$ IVDEP	Ignores only vector dependencies (which is safest)

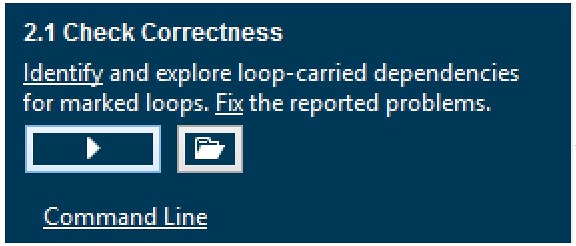
Read More:

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

Is It Safe to Vectorize?

Loop-carried dependencies analysis verifies correctness





Select loop for Correct Analysis and press play!

Vector Dependency prevents Vectorization!

Find vector optimization opportunities

Memory Access pattern analysis

Unit-Stride access

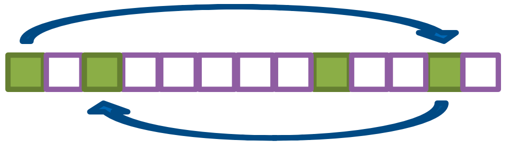
```
for (i=0; i<N; i++)
    A[i] = C[i]*D[i]
```

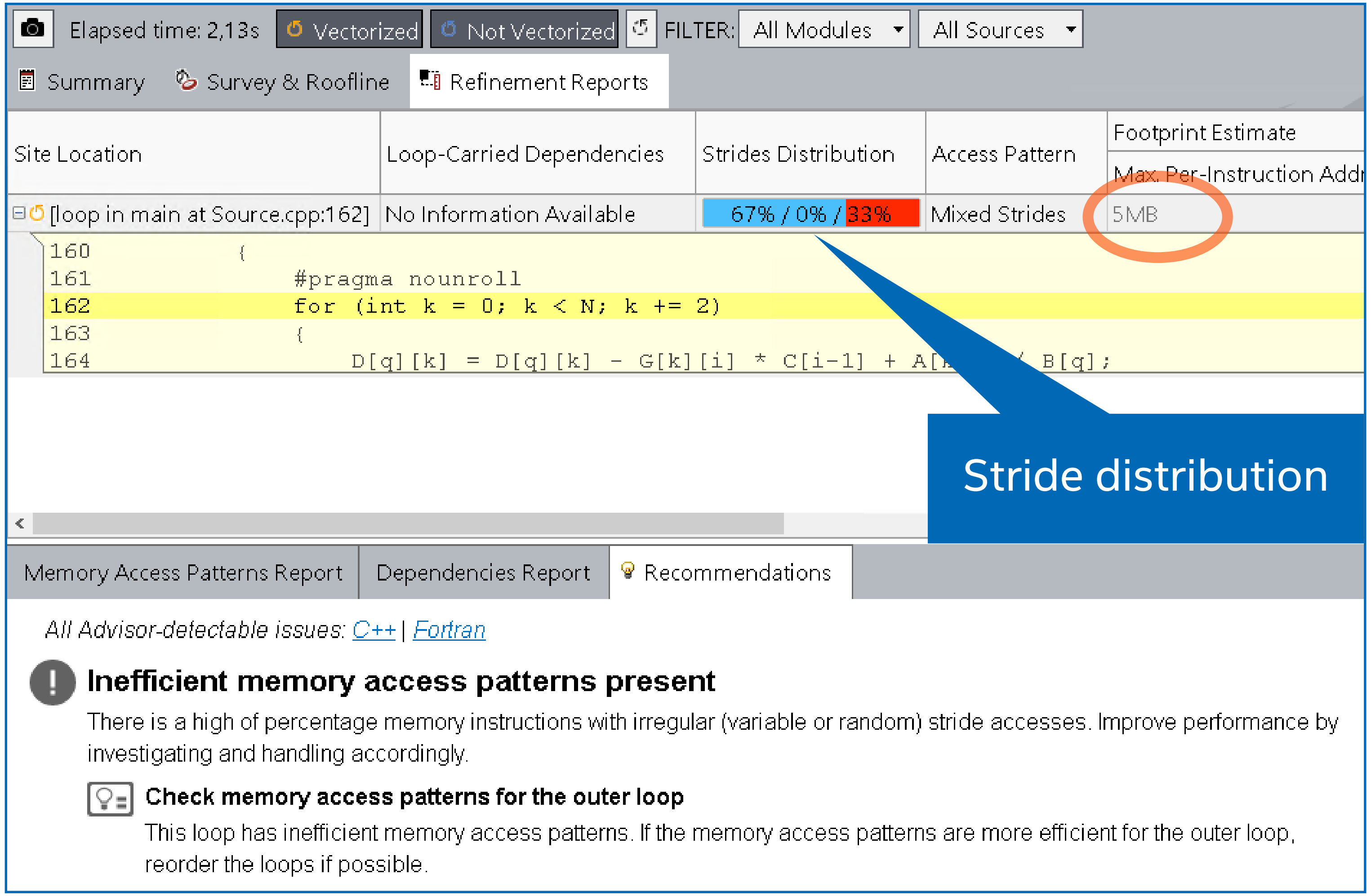
Constant stride access

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

Variable stride access

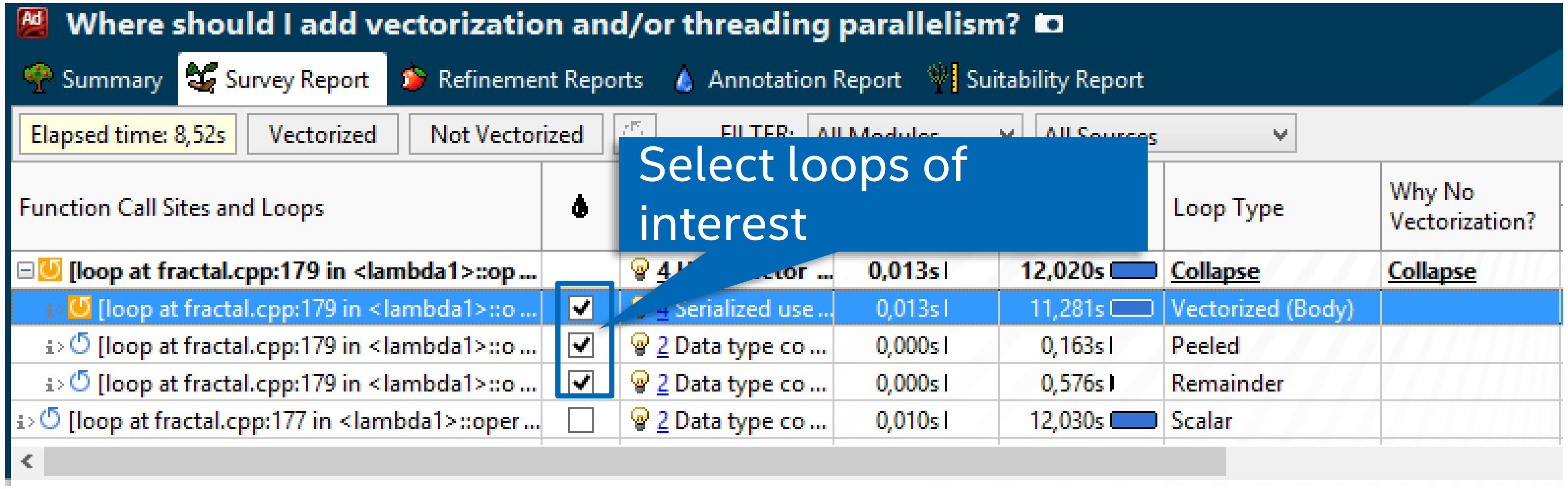
```
for (i=0; i<N; i++)
    A[B[i]] = C[i]*D[i]
```



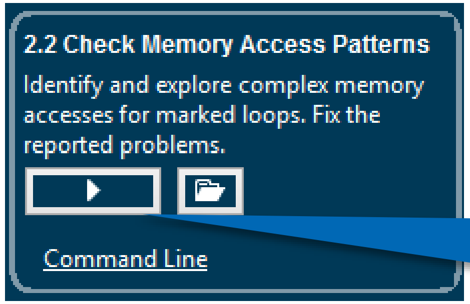


Improve Vectorization

Memory Access pattern analysis



Select loops of interest

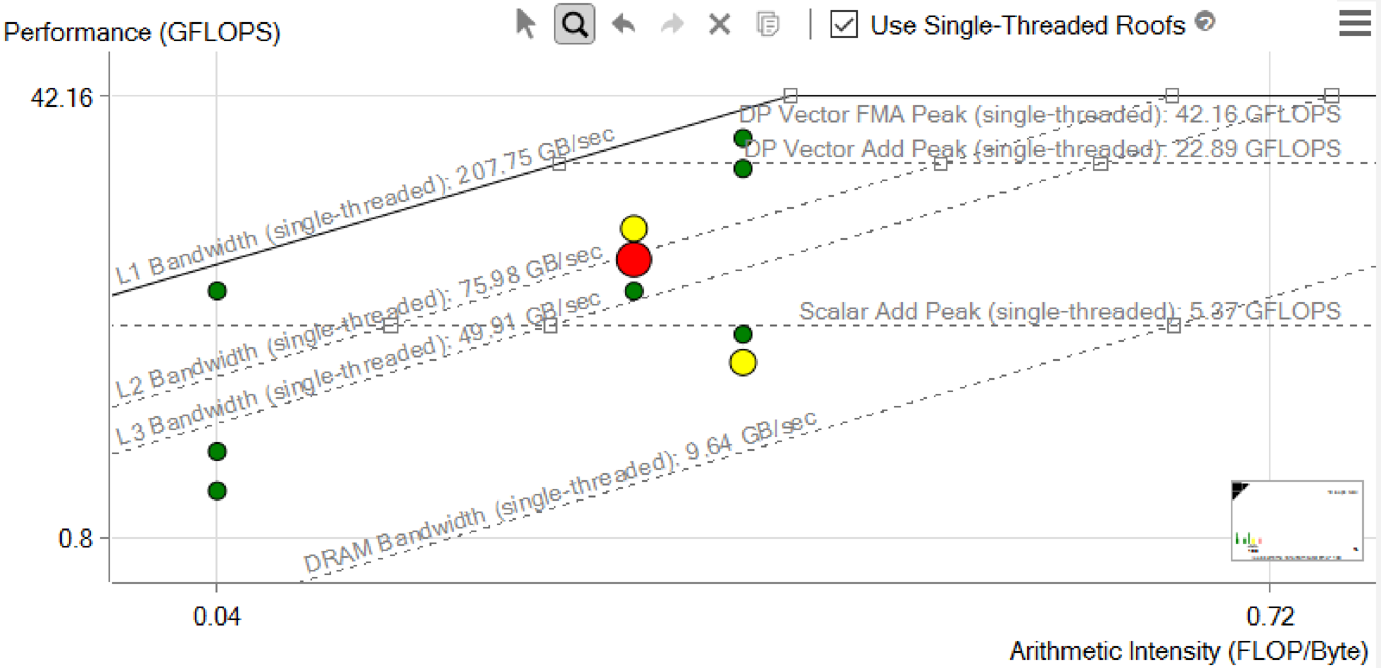


Run Memory Access Patterns analysis, just to check how memory is used in the loop and the called function

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

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

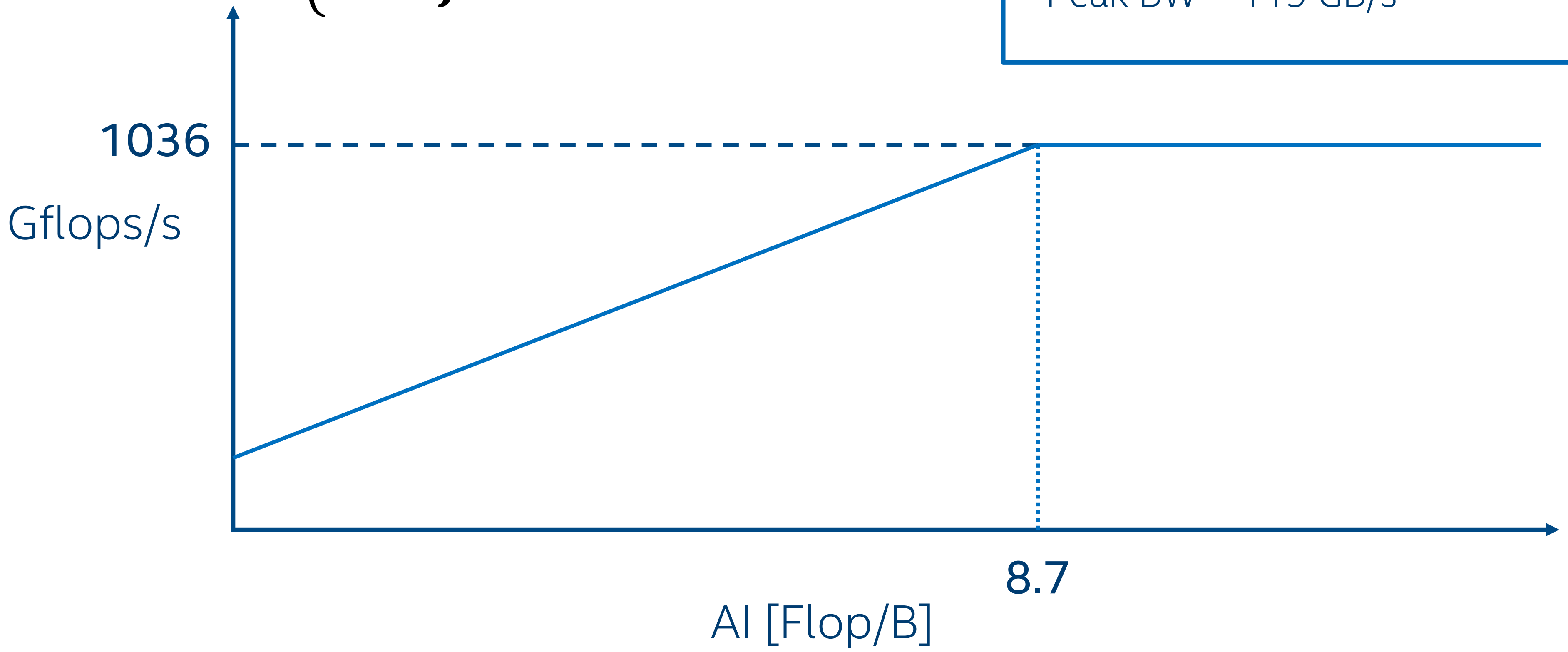
Arithmetic Intensity (Flops/Bytes)

DRAWING THE ROOFLINE

Defining the speed of light

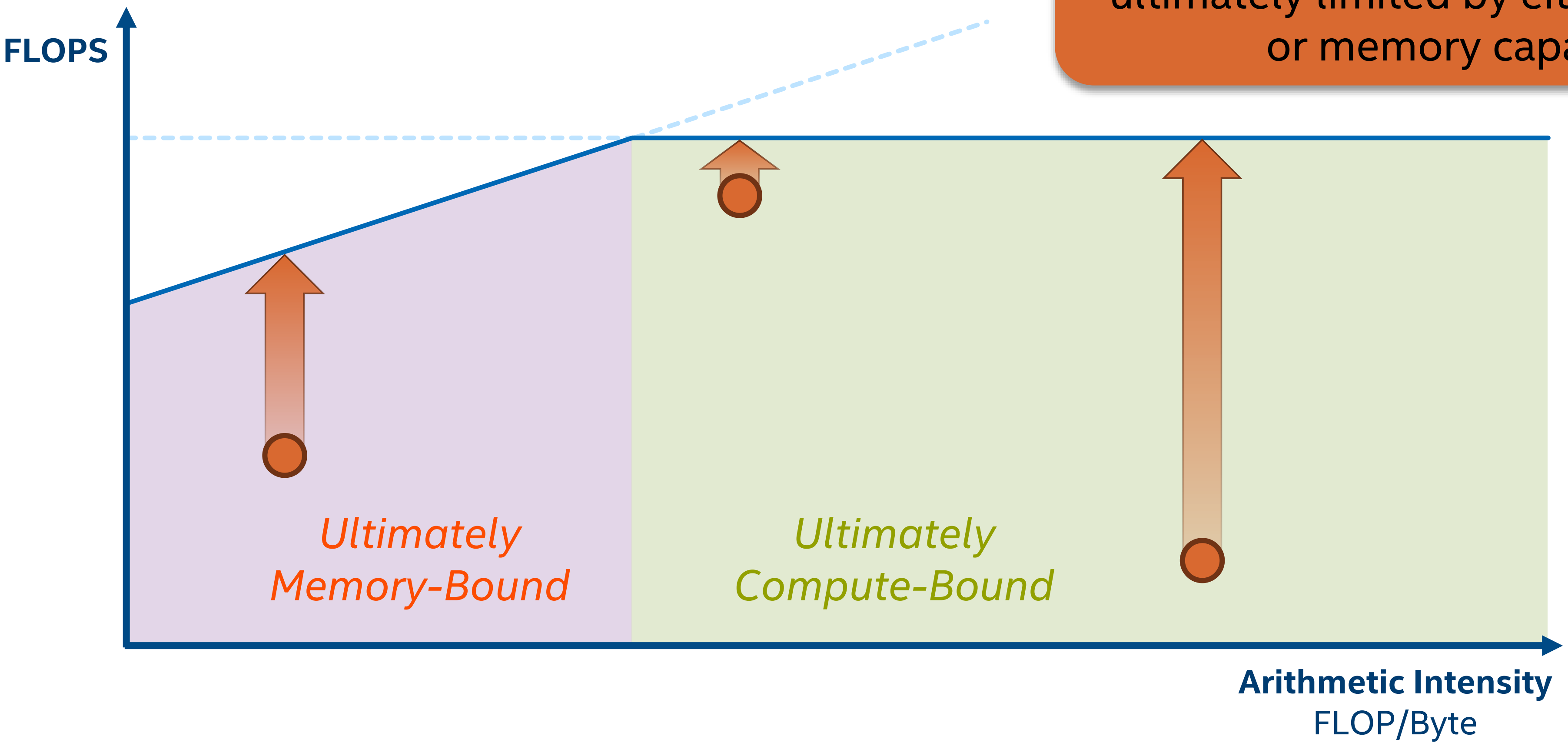
$$\text{Gflop/s} = \min \begin{cases} \textit{Platform PEAK} \\ \textit{Platform BW} * AI \end{cases}$$

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



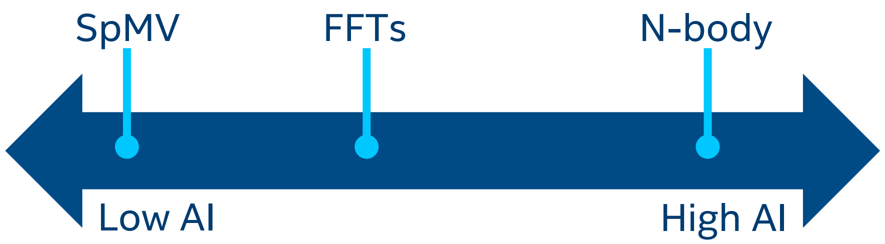
Ultimate Performance Limits

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

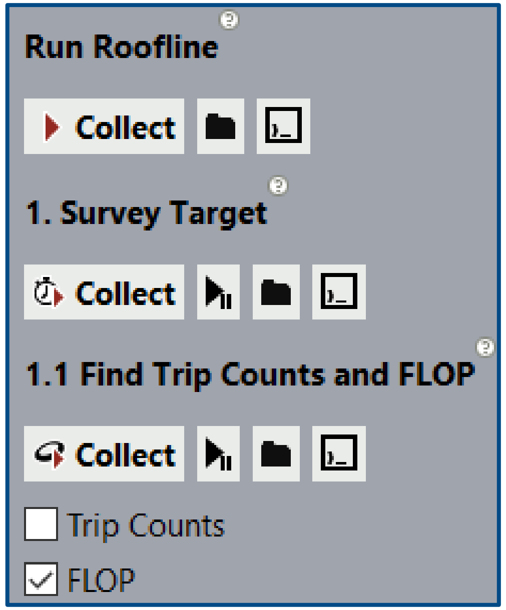


Roofline Metrics

- 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.



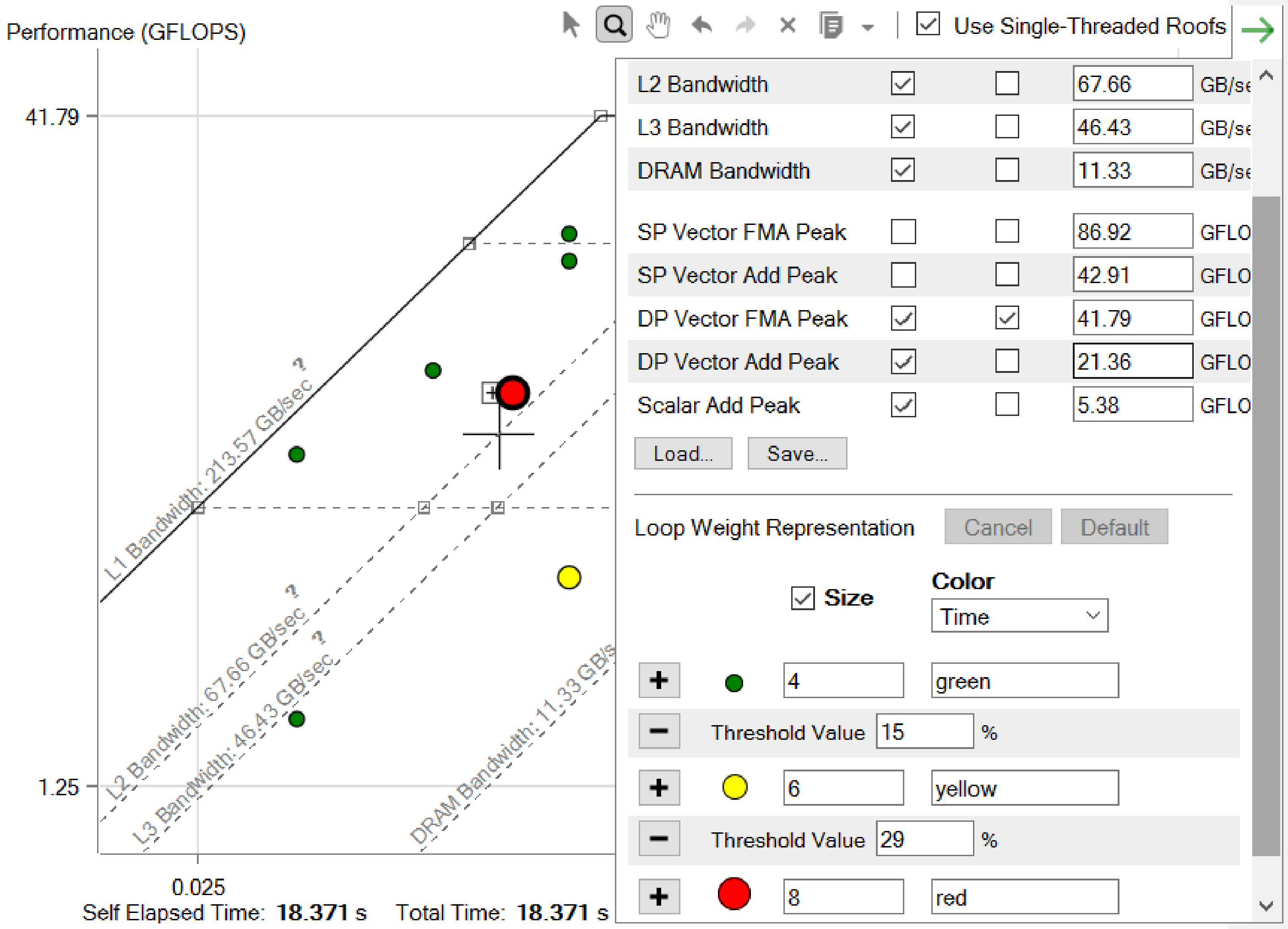
Shortcut to run Survey followed by Trip Counts + FLOPs

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.

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.

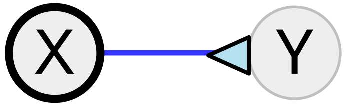
```
for (int i = 0; i < 10; i++)
{
  X = i * 24 + 179.4 - i;
  Y = (i + 18) / 72.8;
  foobar();
  for (int j = 0; j < 3; j++)
  {
    Z = i * j + 7;
  }
}
```

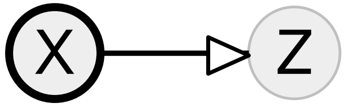
Self

Total

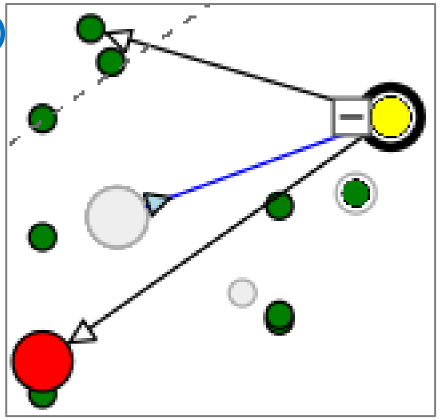
Reading the Roofline with Call Stacks

Visualizing the Call Chain

- Arrows indicate relationships between do
- 

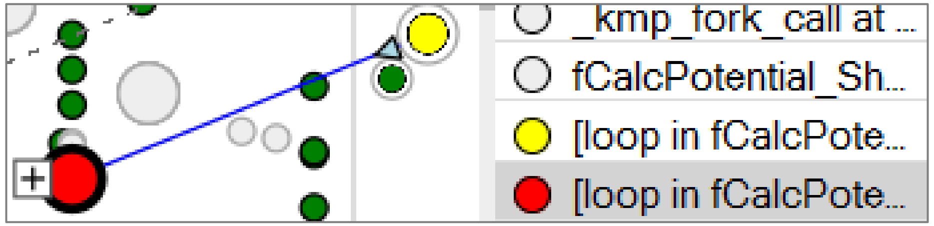
X is called directly by Y.
- 

X directly calls Z

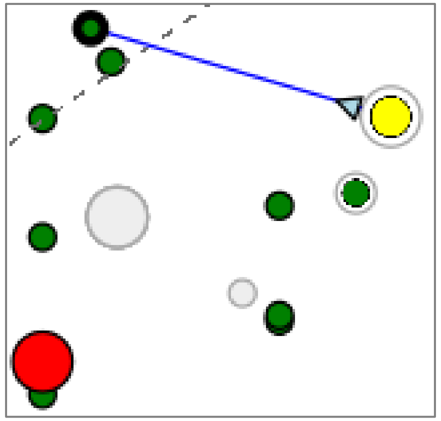


The selected yellow dot was called by the gray dot, and it calls the red and green dots.

- 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.



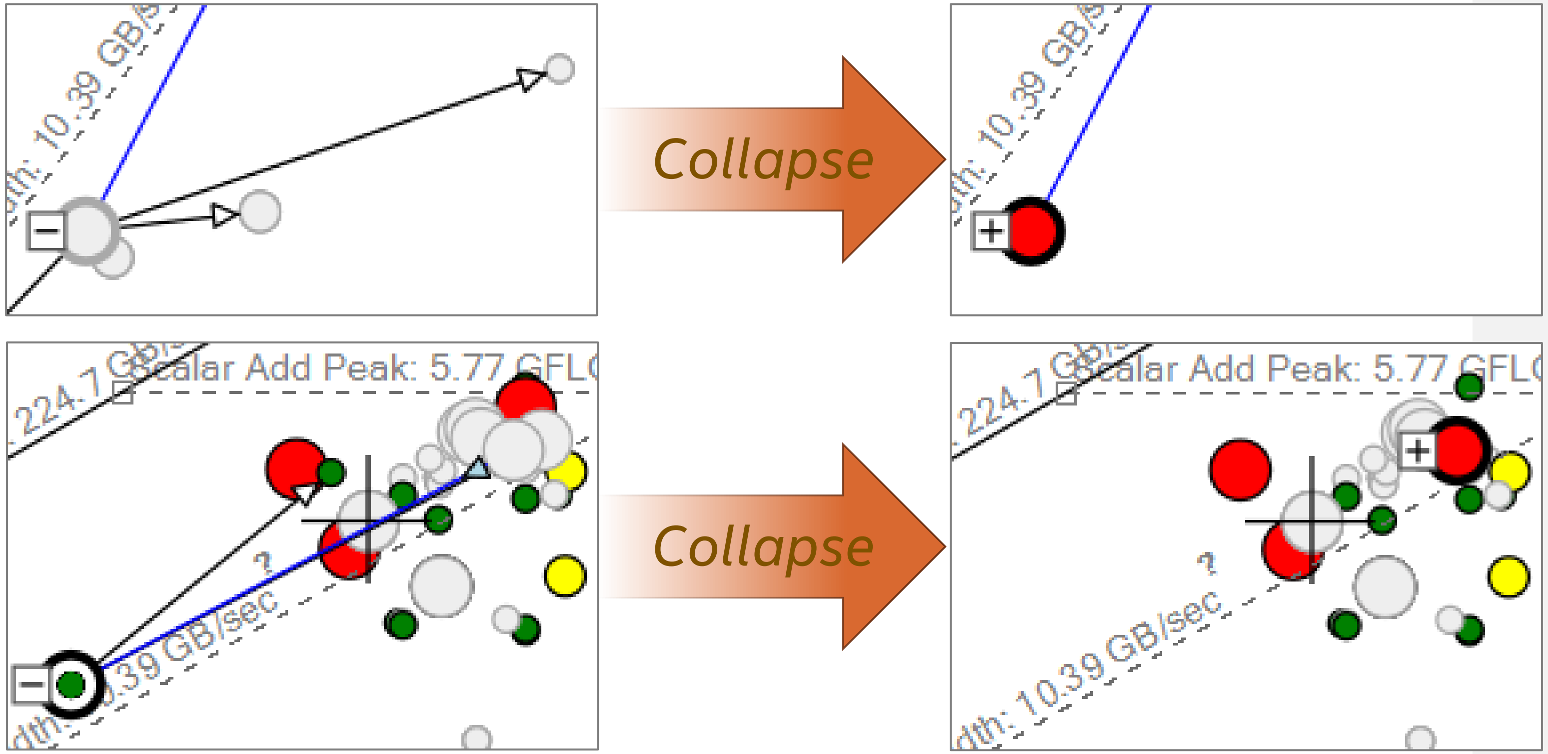
Reading the Roofline with Call Stacks

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

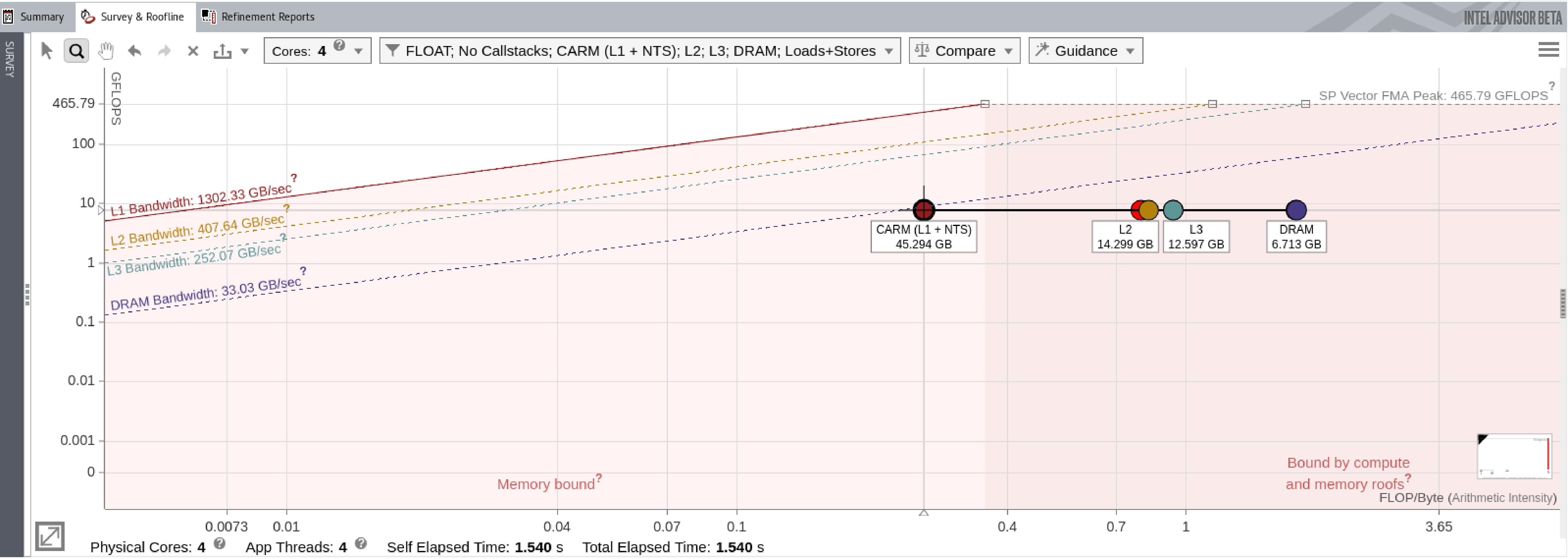
Memory Level Roofline

- 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

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

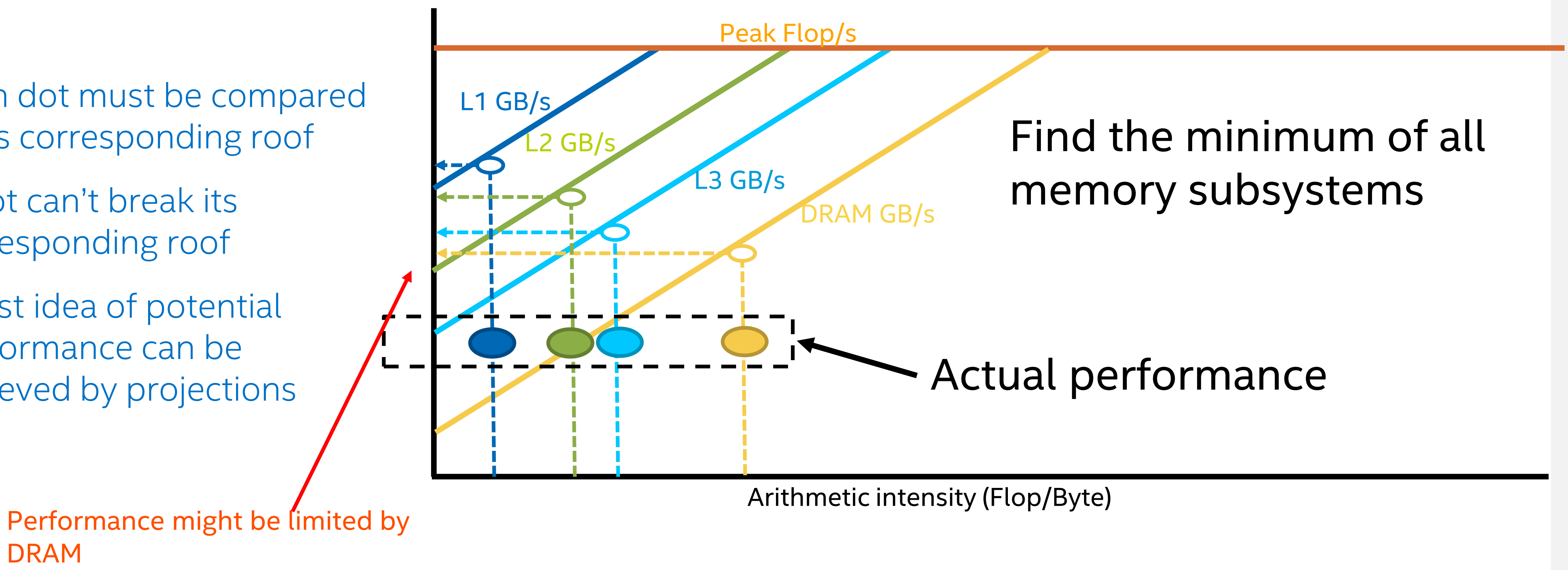
- What can we expect?
  - Due to data locality : AI(L1) =< AI(L2) =< AI(L3) =< AI(DRAM)
  - This only applies in general if you do unit-strided access

Memory-Level Roofline Model in Intel® Advisor



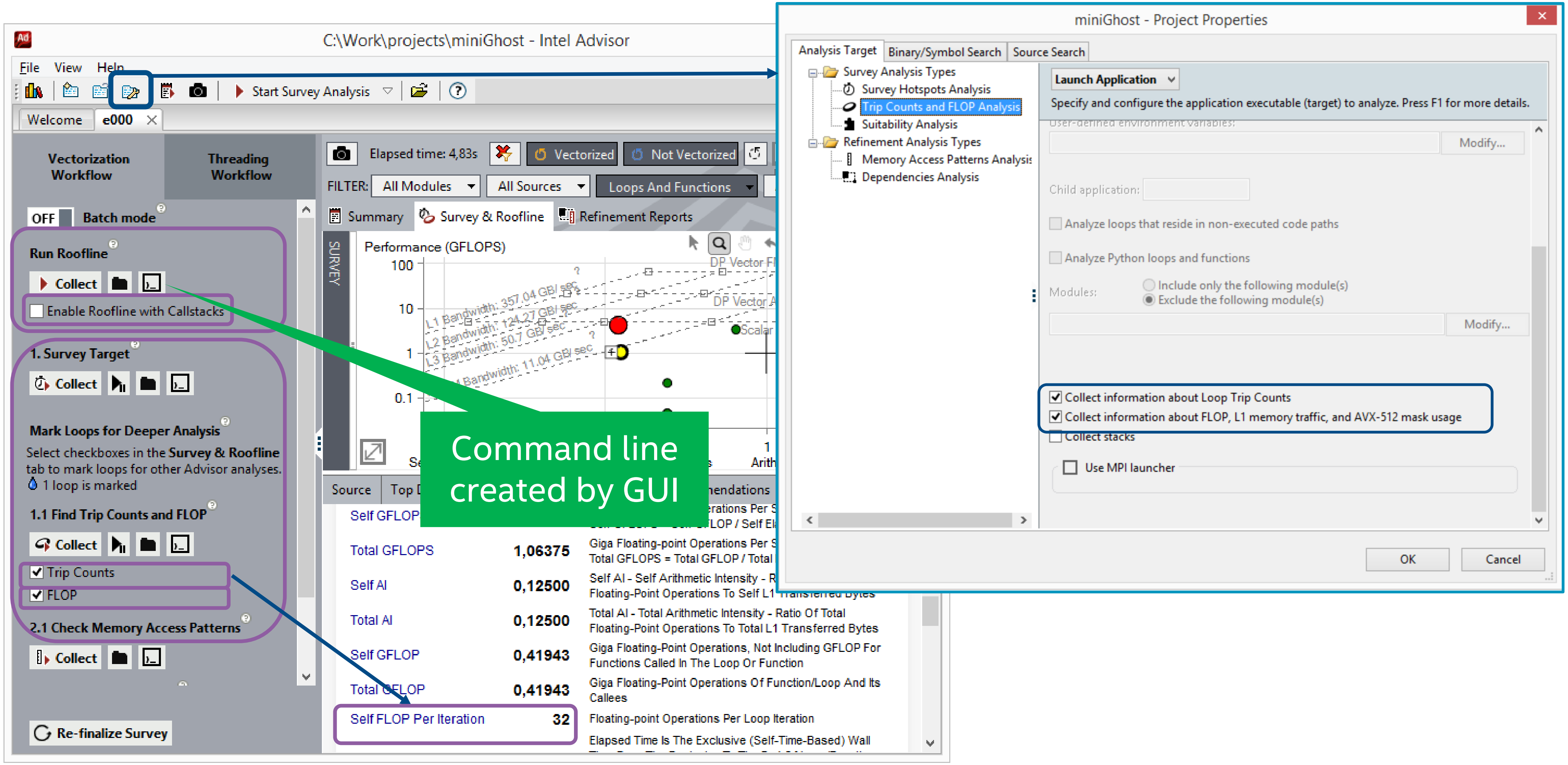
How to Interpret Your Current Limitation?

- 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



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

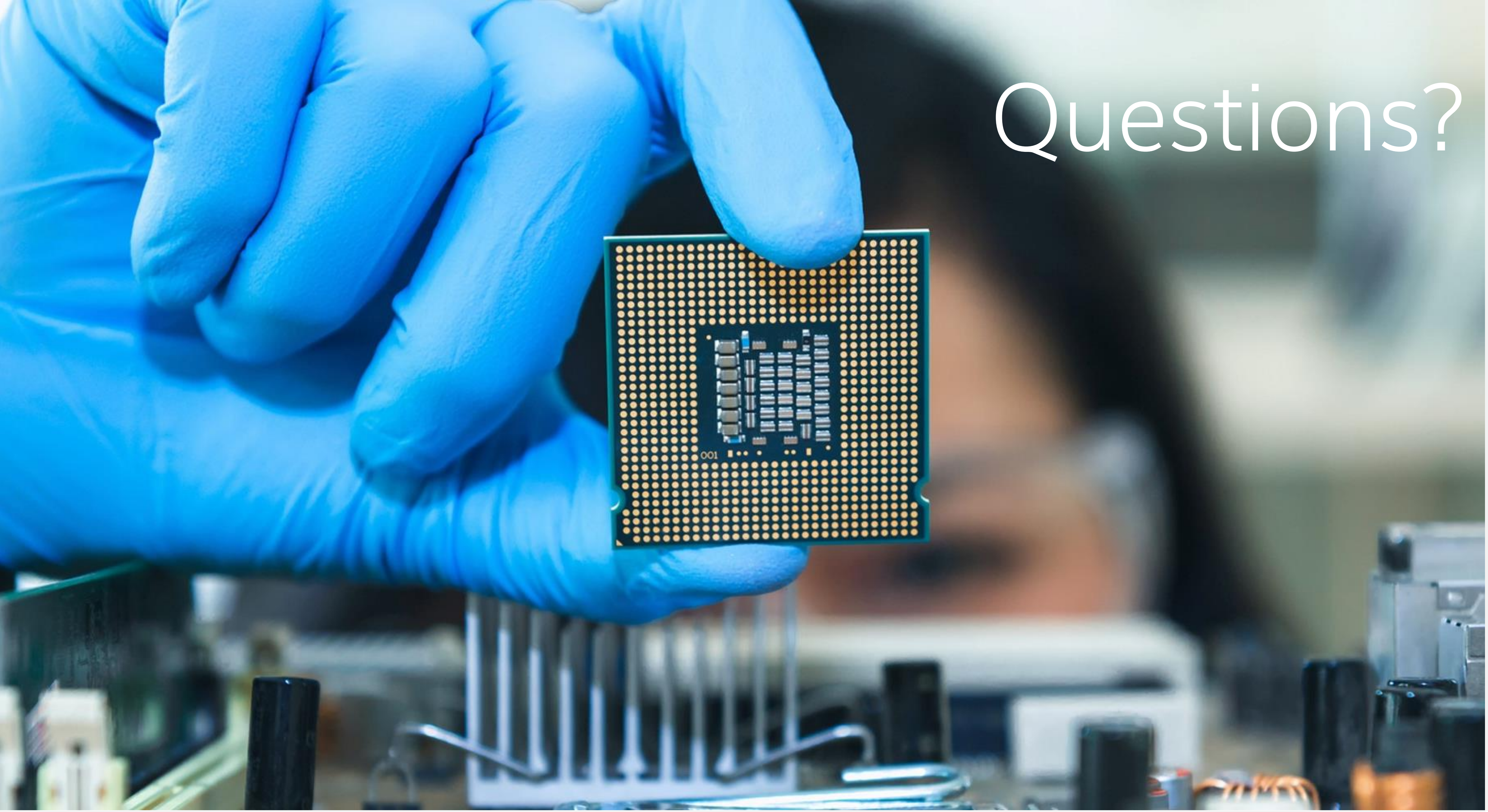
Additional Analysis Tools

- 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.

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