Intel® oneAPI Performance Analysis Tools

Dmitry Tarakanov
Technical Consulting Engineer

intel®

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.

Analysis Tools Overview

Intel® VTune™ Profiler
Performance Profiler

Intel® Advisor
Design and optimize vectorization, threading, accelerator offload and flow graphs.

Intel® Inspector
Memory & Thread Debugger

Optimize Performance with Intel® VTune™ Profiler

Optimize Performance

Intel® VTune™ Profiler

Get the Right Data to Find Bottlenecks

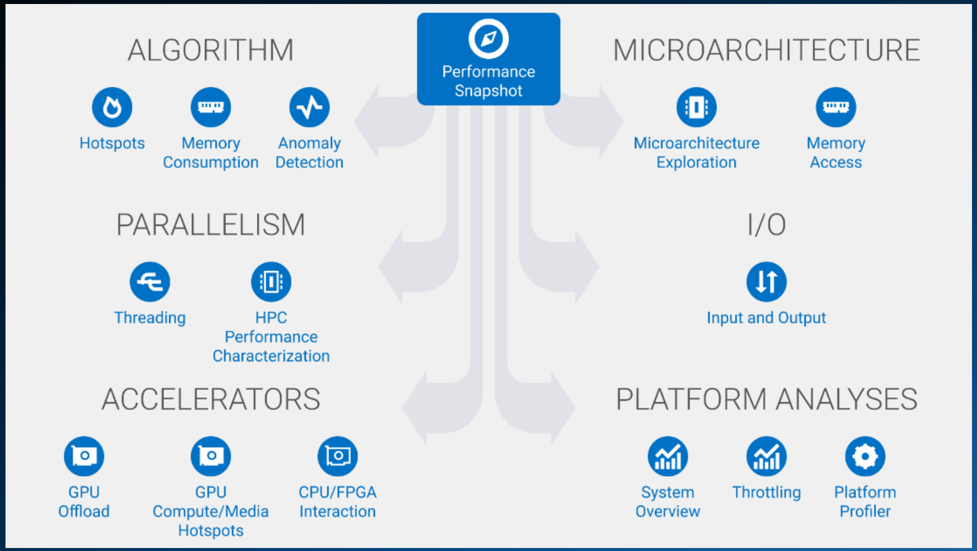
- A suite of profiling for CPU, GPU, FPGA, threading, memory, cache, storage, offload, power…
- DPC++, C, C++, Fortran, Python*, Go*, Java*, or a mix
- Linux, Windows, FreeBSD, Android, Yocto and more

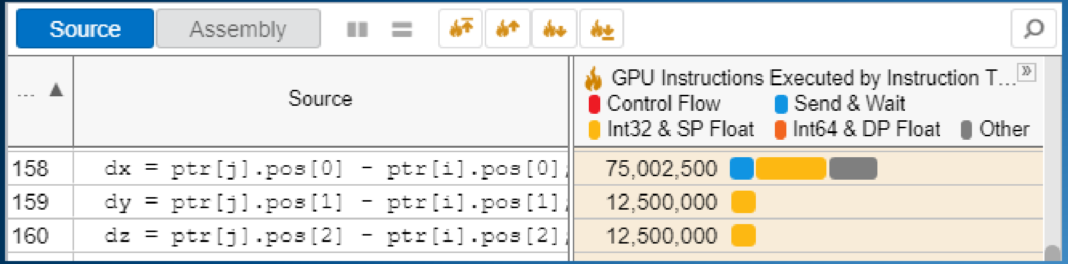
Analyze Data Faster

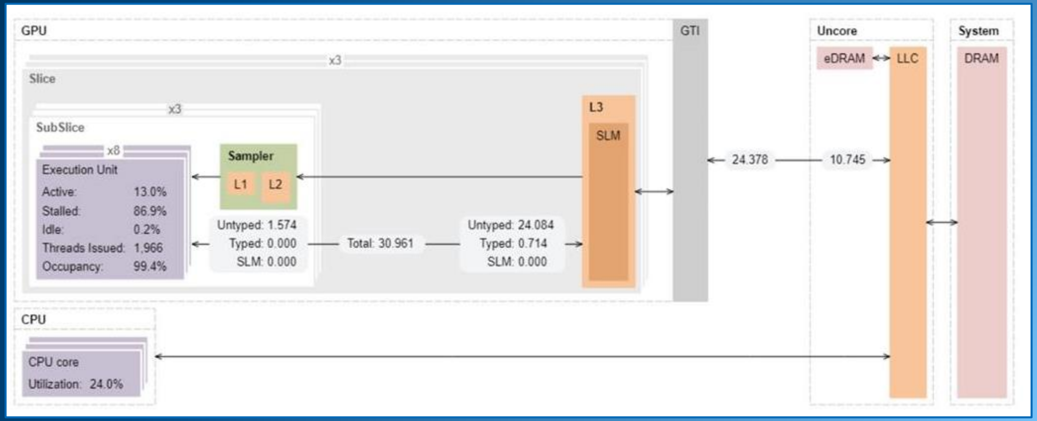
- See data on your source, in architecture diagrams, as a histogram, on a timeline…
- Filter and organize data to find answers

Work Your Way

- User interface or command line
- Profile locally and remotely
- Install as an application
- Install as a server accessible with a web browser







Rich Set of Profiling Capabilities for Multiple Markets

Intel® VTune™ Profiler

Single Thread

Optimize single-threaded performance.

Multithreaded

Effectively use all available cores.

System

See a system-level view of application performance.

Media & OpenCL™ Applications

Deliver high-performance image and video processing pipelines.

HPC & CLoud

Access specialized, in-depth analyses for HPC and cloud computing.

Memory & Storage Management

Diagnose memory, storage, and data plane bottlenecks.

Analyze & Filter Data

Mine data for answers.

Environment

Fits your environment and workflow.

Two Great Ways to Collect Data

Intel® VTune™ Profiler

Software Collector	Hardware Collector
Uses OS interrupts	Uses the on-chip Performance Monitoring Unit (PMU)
Collects from a single process tree	Collect system wide or from a single process tree.
~10ms default resolution	~1ms default resolution (finer granularity - finds small functions)
Either an Intel® or a compatible processor	Requires a genuine Intel® processor for collection
Call stacks show calling sequence	Optionally collect call stacks
Works in virtual environments	Works in a VM only when supported by the VM (e.g., vSphere*, KVM)
No driver required	Uses Intel driver or perf if driver not installed

No special recompiles - C, C++, DPC++, C#, Fortran, Java, Python, Assembly

Find Answers Fast

Intel® VTune™ Profiler

Adjust Data Grouping

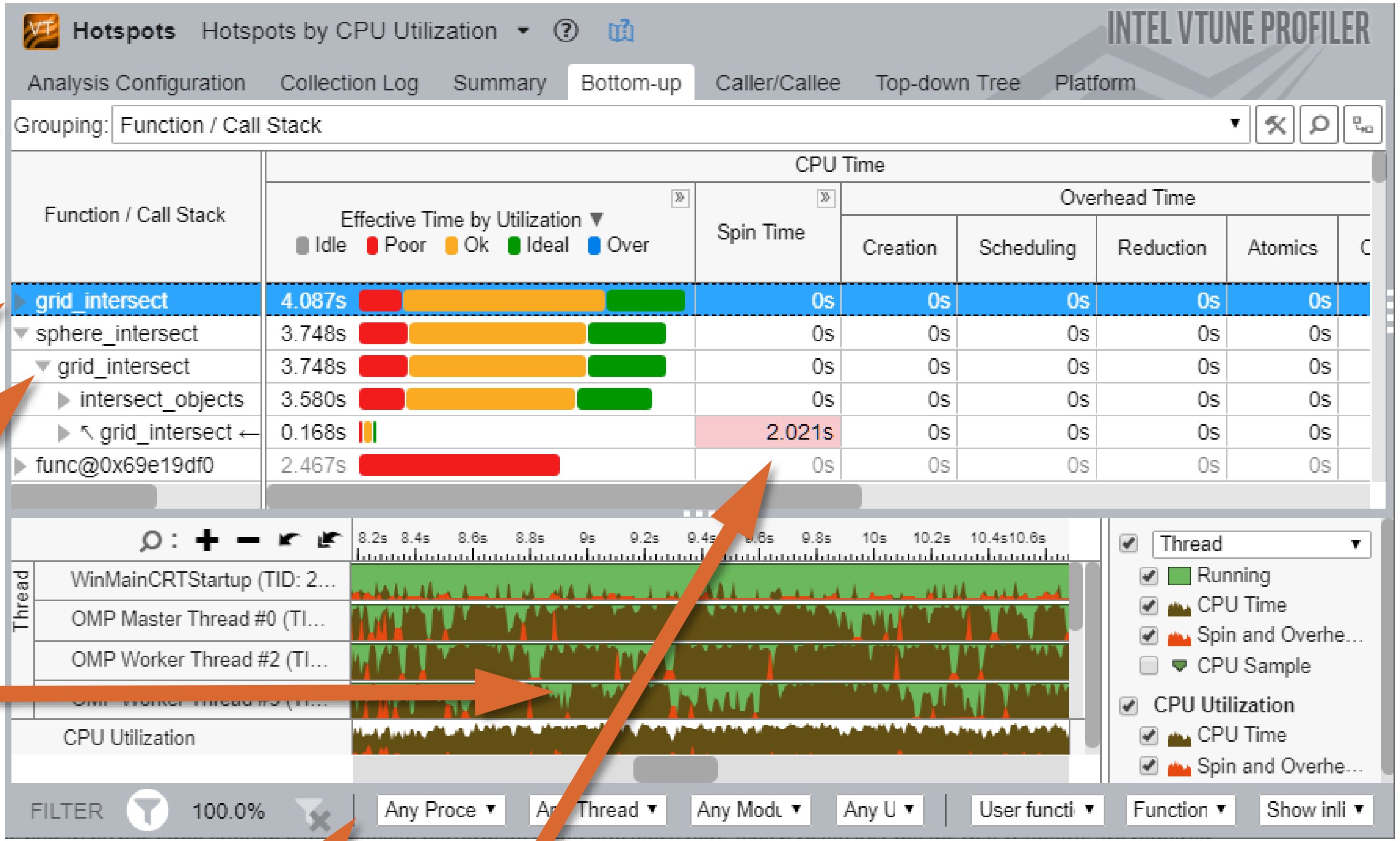
Function / Call Stack
Source Function / Function / Call Stack
Sync Object / Function / Call Stack
Sync Object / Thread / Function / Call Stack
... (Partial list shown)

Double Click Function to View Source

Click [▶] for Call Stack

Filter by Timeline Selection (or by Grid Selection)

Zoom In And Filter On Selection
Filter In by Selection
Remove All Filters



Filter by Process & Other Controls

Tuning Opportunities Shown in Pink. Hover for Tips

See Profile Data On Source / Asm

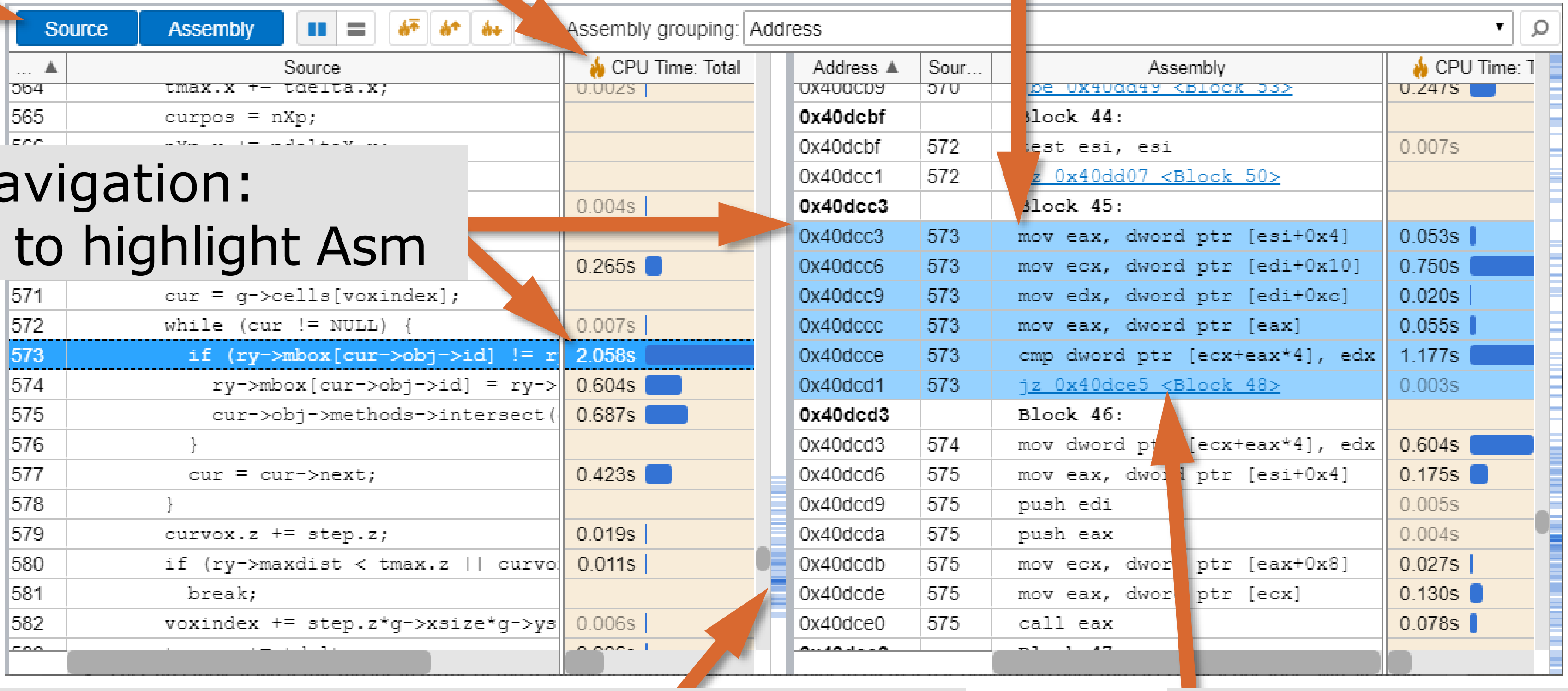
Double Click from Grid or Timeline

View Source / Asm or both

CPU Time

Right click for instruction reference manual

Quick Asm navigation:
Select source to highlight Asm

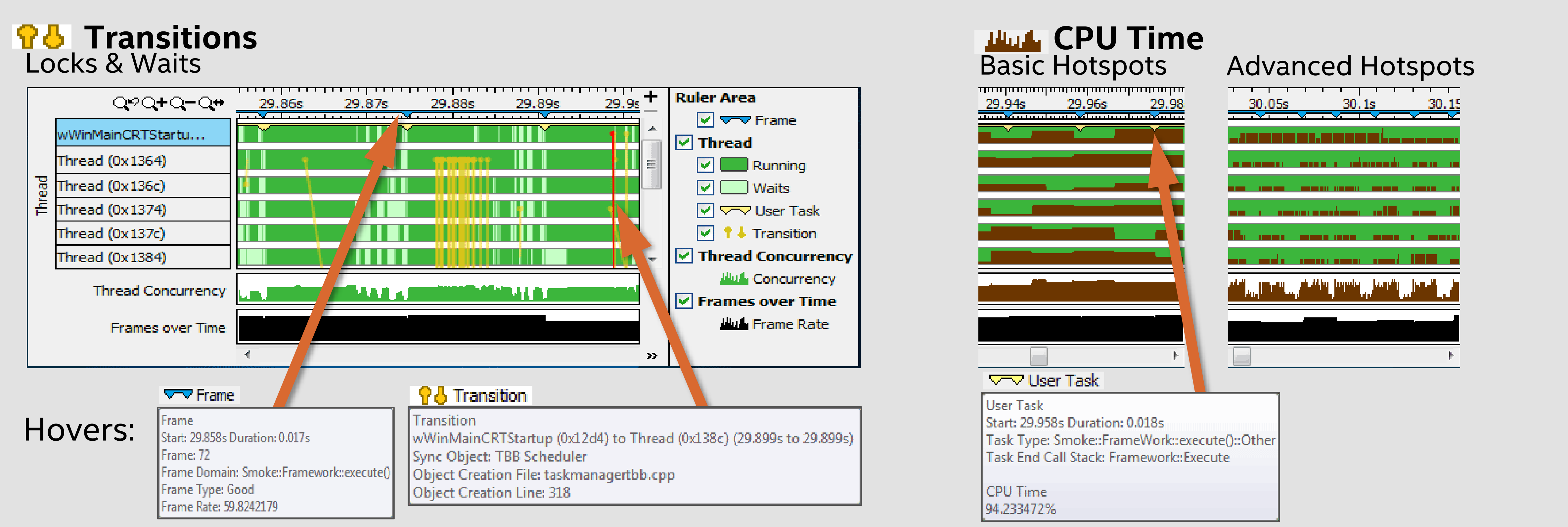


Scroll Bar "Heat Map" is an overview of hot spots

Click jump to scroll Asm

Timeline Visualizes Thread Behavior

Intel® VTune™ Profiler



- Optional: Use API to mark frames and user tasks Frame User Task
- Optional: Add a mark during collection

Command Line Interface

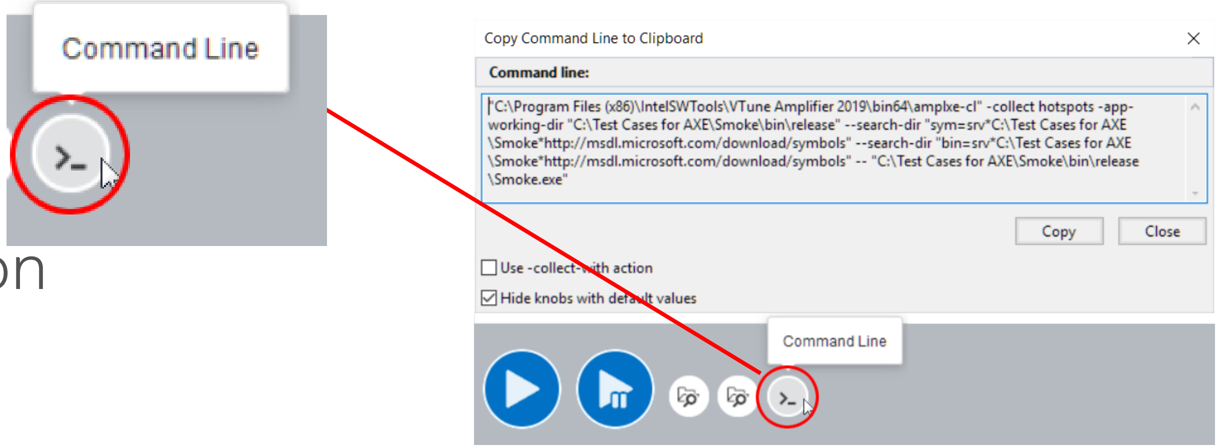
Automate analysis

- Set up the environment variables:
  - **Windows:** `<install-dir>\env\vars.bat`
  - **Linux:** `<install-dir>/env/vars.sh`

Help: `vtune -help`

Use UI to setup

1) Configure analysis in UI
2) Press “Command Line…” button
3) Copy & paste command



Great for regression analysis – send results file to developer
Command line results can also be opened in the UI

Default Intel® VTune™ Profiler Install Directories

In Intel® oneAPI Base Toolkit:

- Windows: [Program Files]\Intel\oneAPI\vtune\<*version*>
- Linux: /opt/intel/oneapi/vtune/<*version*>

Standalone:

- Windows: [Program Files]\IntelSWTools\VTune Profiler <*version*>
- Linux: /opt/intel/vtune_profiler_*version*

On Apple* macOS* systems:

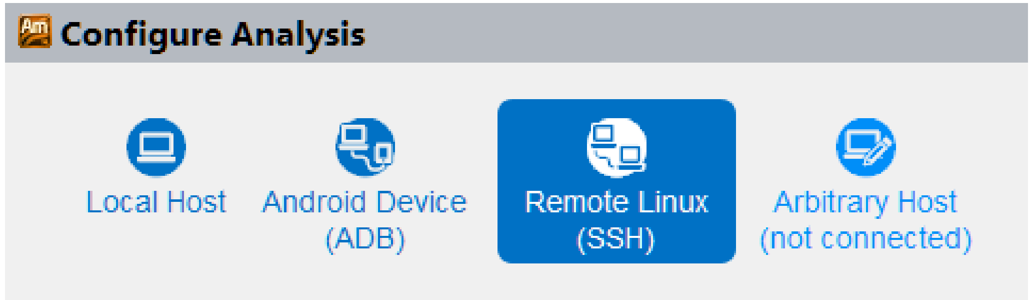
- /Applications/Intel VTune Profiler <*version*>.app

Interactive Remote Data Collection

Performance analysis of remote systems just got a lot easier

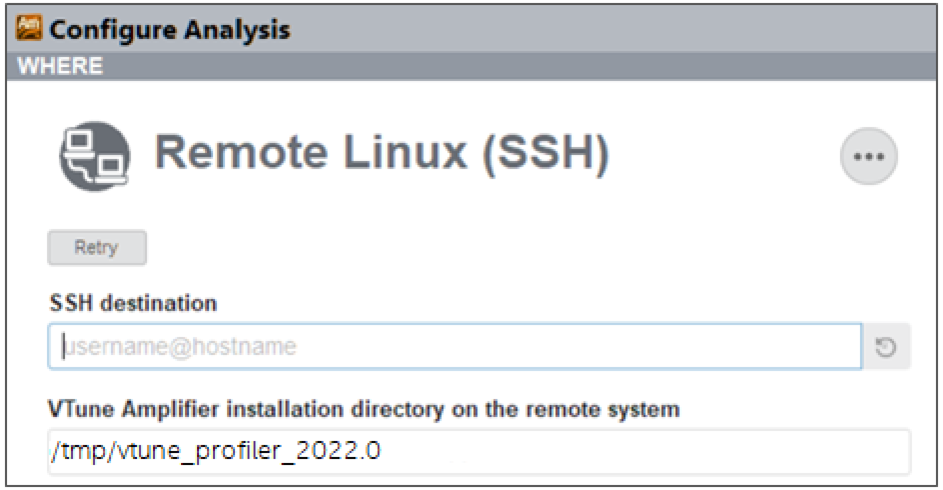
Interactive analysis

1) Configure SSH to a remote Linux* target
2) Choose and run analysis with the UI



Command line analysis

1) Run command line remotely on Windows* or Linux* target
2) Copy results back to host and open in UI



Conveniently use your local UI to analyze remote systems

Intel® VTune™ Profiler

Faster, Scalable Code Faster

Get the Data You Need

- Hotspot (Statistical call tree), Call counts (Statistical)
- Thread Profiling – Concurrency and Lock & Waits Analysis
- Cache miss, Bandwidth analysis…[1]
- GPU Offload and OpenCL™ Kernel Tracing

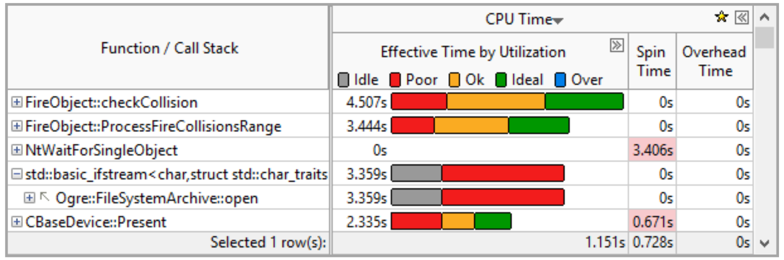
Find Answers Fast

- View Results on the Source / Assembly
- OpenMP Scalability Analysis, Graphical Frame Analysis
- Filter Out Extraneous Data – Organize Data with Viewpoints
- Visualize Thread & Task Activity on the Timeline

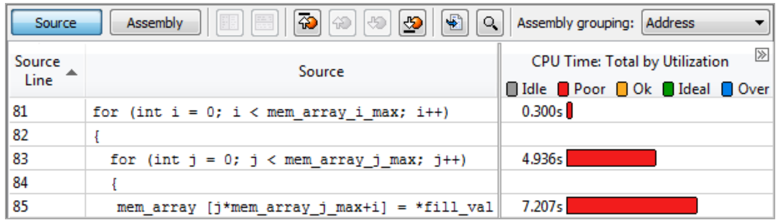
Easy to Use

- No Special Compiles – C, C++, C#, Fortran, Java, Python, ASM
- Visual Studio* Integration or Stand Alone
- Local & Remote Data Collection, Command Line
- Analyze Windows* & Linux* data on macOS

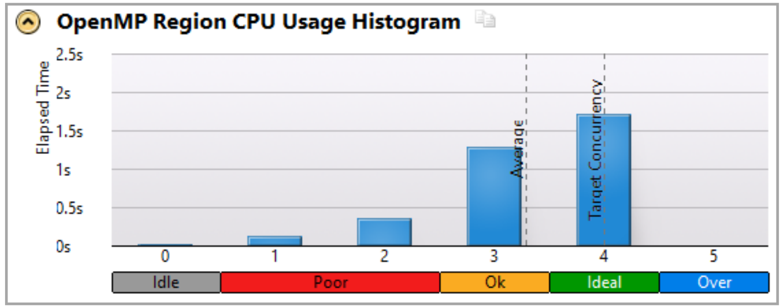
Quickly Find Tuning Opportunities



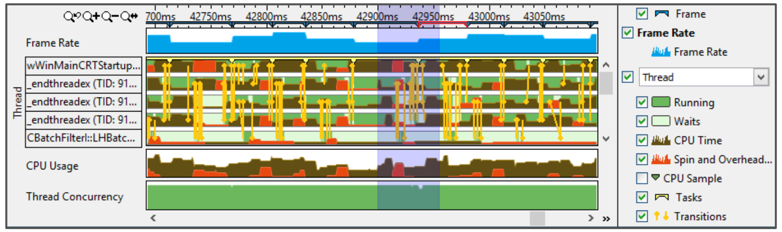
See Results On The Source Code



Tune OpenMP Scalability



Visualize & Filter Data



[1] Events vary by processor. [2] No data collection on OS X*

intel.

Intel® VTune™ Profiler

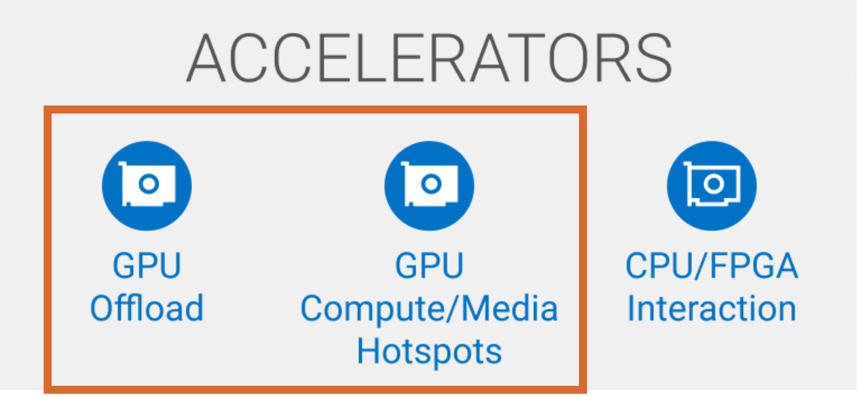
GPU Profiling

Two GPU Analysis types

Intel® VTune™ Profiler

GPU Offload: **Is the offload efficient?**

- Find inefficiencies in offload
- Identify if you are CPU or GPU bound
- Find the kernel to optimize first
- Correlate CPU and GPU activity
- Analyze DMA packet execution



GPU Compute/Media Hotspots: **Is the GPU kernel efficient?**

- Identify what limits the performance of the kernel
- GPU source/instruction level profiling
- Find memory latency or inefficient kernel algorithms

GPU Offload Profiling

Intel® VTune™ Profiler

- Simply follow the sections on the Summary page
- Tuning methodology on top of HW metrics

GPU Usage: 0.6%

Use this section to understand whether the GPU was utilized properly and which of the engines were utilized. Identify the amount of gaps in the GPU utilization that potentially could be loaded with some work. This metric is calculated for the engines that had at least one piece of work scheduled to them.

GPU Usage

GPU Usage breakdown by GPU engines and work types.

GPU Engine / Packet Type	GPU Time	(%)
Render and GPGPU	1.146s	0.6%
Unknown	0.888s	0.5%
GHAL3D	0.249s	0.1%
OpenCL	0.009s	0.0%

EU Array Stalled/Idle: 94.4% of Elapsed time

Analyze the average value of EU Array Stalled/Idle metric and identify why EUs were waiting for resources instead of doing computations. This metric is critical for compute-bound applications. Explore typical reasons for this kind of inefficiency listed below.

GPU L3 Bandwidth Bound: 0.5% of peak value

DRAM Bandwidth Bound: 0.0% of Elapsed time

Occupancy: 25.8% of peak value

Identify too large or too small computing tasks with low occupancy that make the EU array idle while waiting for the scheduler. Note that frequent SLM accesses and barriers may affect the maximum possible occupancy.

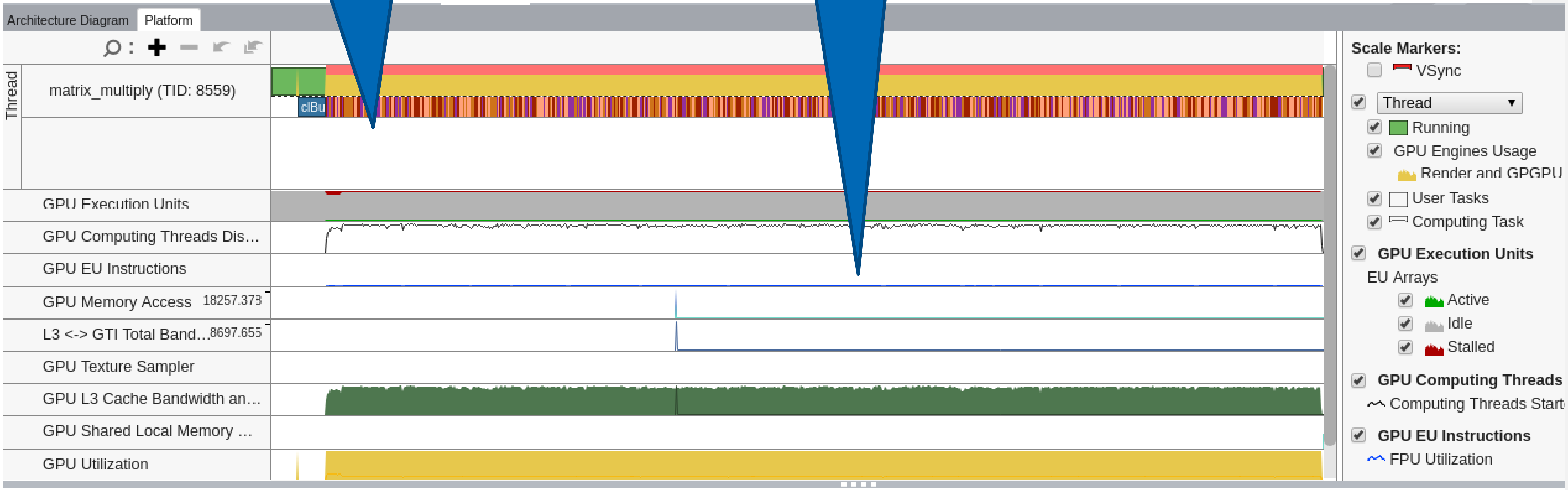
Hottest GPU Computing Tasks with Low Occupancy

Sampler Busy: 40.6% of peak value

Timeline Correlates GPU and CPU Activity

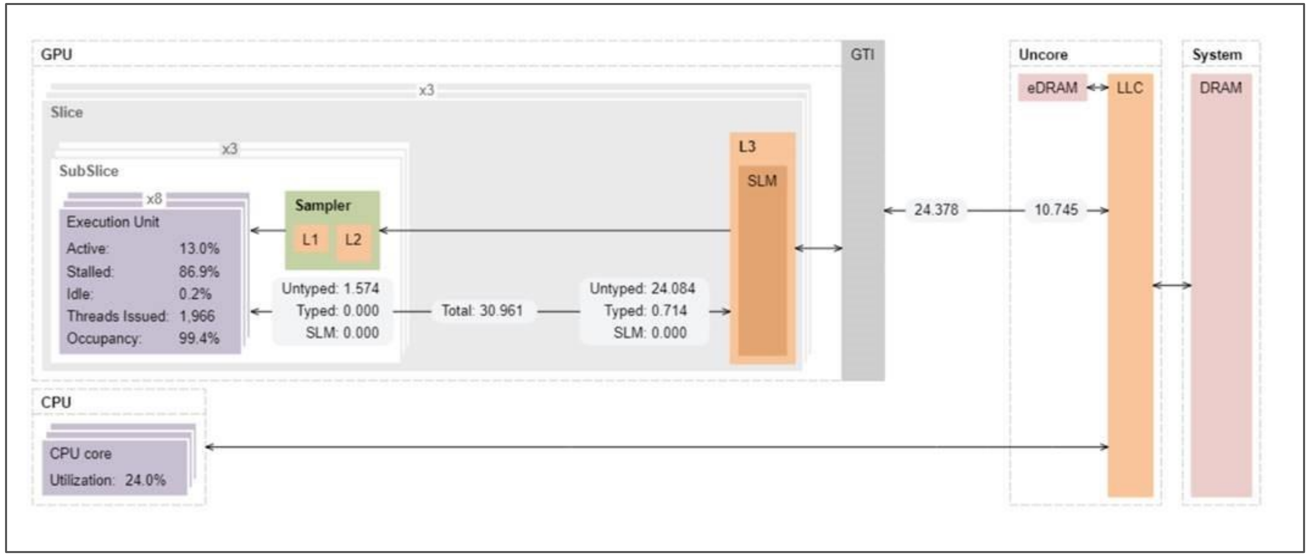
Identify too much or too little kernel activity

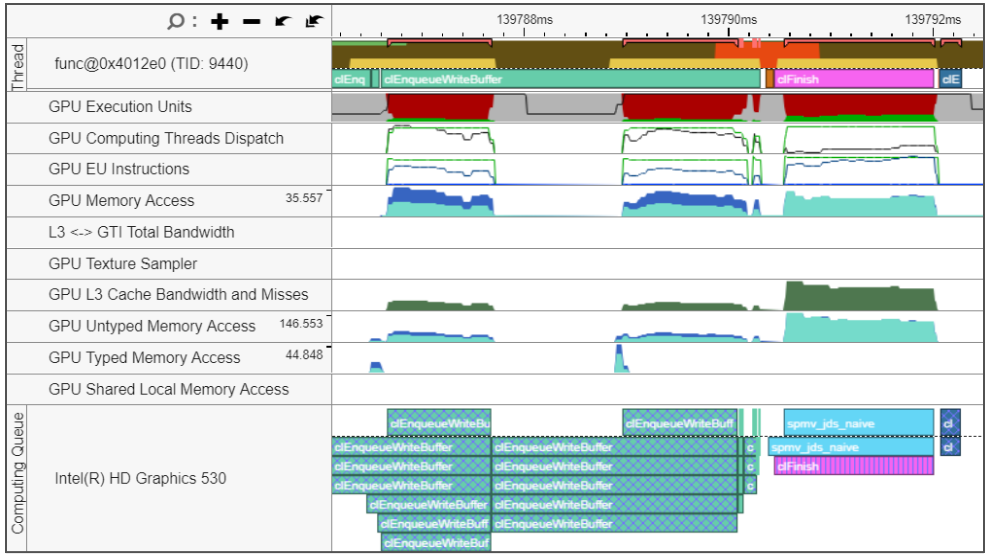
Correlate GPU activity with kernels and threads



GPU Hotspots: Aggregated and Overtime Views

Computing Task	Work Size		Computing Task			
	Global	Local	Total Time ▼	Average Time	Instance Count	SIMD Width
▶ clEnqueueWriteBuffer			0.005s	0.000s	14	
▶ spmv_jds_naive	146944	256	0.003s	0.001s	2	16
▶ clEnqueueReadBuffer			0.000s	0.000s	2	
▶ [Outside any task]			0s	0s	0	



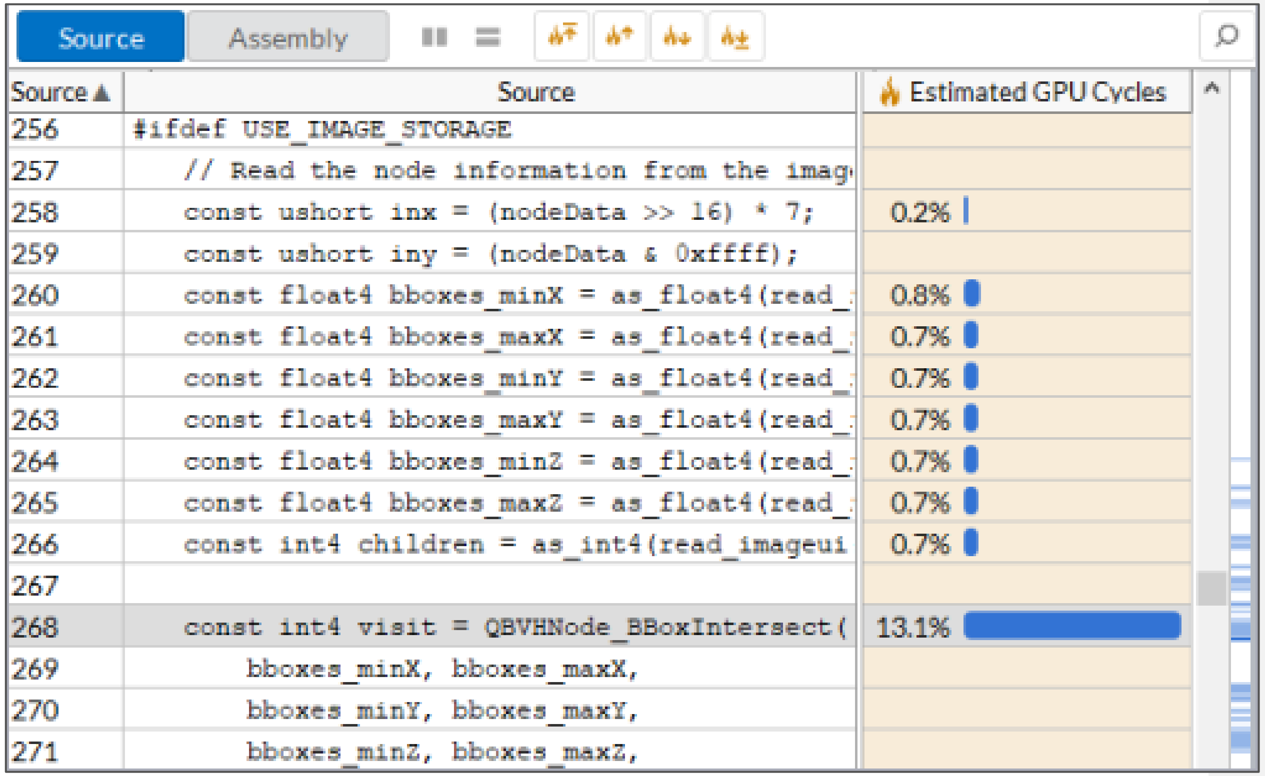


GPU Compute/Media Hotspots

Tune Inefficient Kernel Algorithms

Analyze GPU Kernel Execution

- Find memory latency or inefficient kernel algorithms
- See the hotspot on the OpenCL™ or DPC++ source & assembly code
- Analyze DMA packet execution
  - Packet Queue Depth histogram
  - Packet Duration histogram
- GPU-side call stacks



intel.

Intel® VTune™ Profiler

Memory Analysis

What's Using All The Memory?

Memory Consumption Analysis

See What Is Allocating Memory

- Lists top memory consuming functions and objects
- View source to understand cause
- Filter by time using the memory consumption timeline

Standard & Custom Allocators

- Recognizes libc malloc/free, memkind and jemalloc libraries
- Use custom allocators after markup with ITT Notify API

Languages

- Python*
- Linux*: Native C, C++, Fortran

Top Memory-Consuming Objects

This section lists the most memory-consuming objects in your application. Optimizing these objects results in improving an overall application memory consumption.

Memory Object	Memory Consumption
dictobject.c:632 (768 B)	768 B
filedoalloc.c:120 (4 KB)	4 KB
iofopen.c:76 (568 B)	568 B
msort.c:224 (1 KB)	1 KB
dictobject.c:632 (3 KB)	3 KB
[Others]	217 TB

Native language support is not currently available for Windows*

Optimize Memory Access

Memory Access Analysis - Intel® VTune™ Profiler

Tune data structures for performance

- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

Optimize NUMA latency & scalability

- True & false sharing optimization
- Auto detect max system bandwidth
- Easier tuning of inter-socket bandwidth

Easier install, Latest processors

- No special drivers required on Linux*
- Intel® Xeon Phi™ processor MCDRAM (high bandwidth memory) analysis

Top Memory Objects by Latency

This section lists memory objects that introduced the highest latency to the overall application execution.

Memory Object	Total Latency	Loads	Stores	LLC Miss Count
alloc_test.cpp:157 (30 MB)	65.6%	4,239,327,176	4,475,334,256	0
alloc_test.cpp:135 (305 MB)	6.8%	411,212,336	441,613,248	0
alloc_test.cpp:109 (305 MB)	6.3%	439,213,176	449,613,488	0
alloc_test!l_data_init.436.0.6 (576 B)	5.2%	742,422,272	676,820,304	0
[vmlinux]	4.6%	173,605,208	116,003,480	0
[Others]	11.5%	1,533,646,008	1,674,450,232	0

*N/A is applied to non-summable metrics.

Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	Stores	LLC Miss Count ▼	
		Local DRAM Access Count	Remote DRAM Access Count
doTriadompparallel_for@2	40,307,609,1...	2,439,273,176	2,430,472,912
triad!c (152 MB)	19,200,576	1,821,654,648	1,864,855,944
triad!b (152 MB)	10,400,312	615,218,456	560,816,824
[Unknown]	7,200,216	2,400,072	3,200,096
triad!doTriad (2 MB)	15,200,456	0	0
[Stack]	2,120,063,600	0	1,600,048
triad!a (152 MB)	38,135,544,0...	0	0
update_blocked_averages	6,400,192	2,400,072	0

Memory Access Analysis

Intel® VTune™ Profiler

Tune data structures for better performance

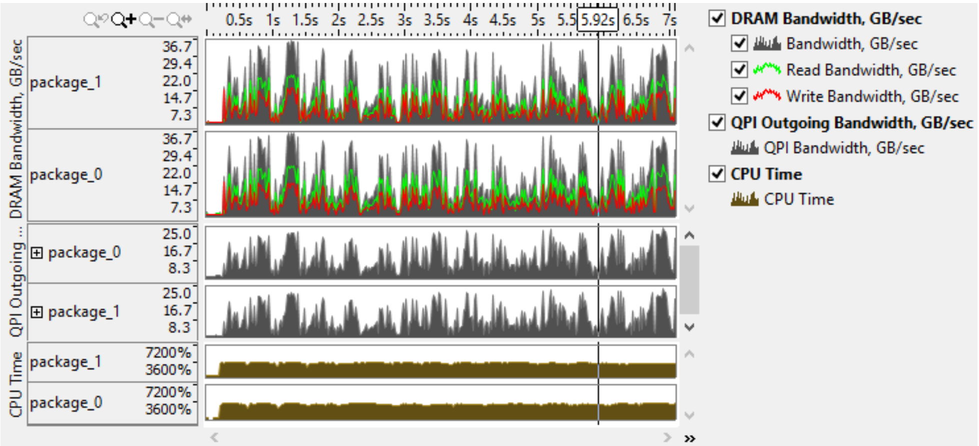
- Attribute cache misses to data structures

Grouping: Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack

Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack	Memory Bound	Loads	Stores	LLC Miss Count	Average Latency (cycles)
⊟ DRAM, GB/sec	0.657	125,874,377,622	16,061,040...	130,507,830	40
⊟ High	0.750	28,236,084,708	5,014,875, ...	75,304,518	91
⊞ stream.c:180 (76 MB)		900,002,700	654,009,810	18,301,098	495
⊞ stream.c:179 (76 MB)		1,050,003,150	667,210,008	33,301,998	487
⊞ stream.c:181 (76 MB)		1,434,004,302	907,213,608	20,101,206	412
Selected 1 row(s):	1.000	126,000,378	21,600,324	300,018	61

Better Bandwidth Analysis for Non-Uniform Memory

- See Read & Write contributions to Total Bandwidth
- Easier tuning of multi-socket bandwidth



Seeing total bandwidth can suggest data blocking opportunities to change a bandwidth bound app into a compute bound app.

Intel® VTune™ Profiler

Demo

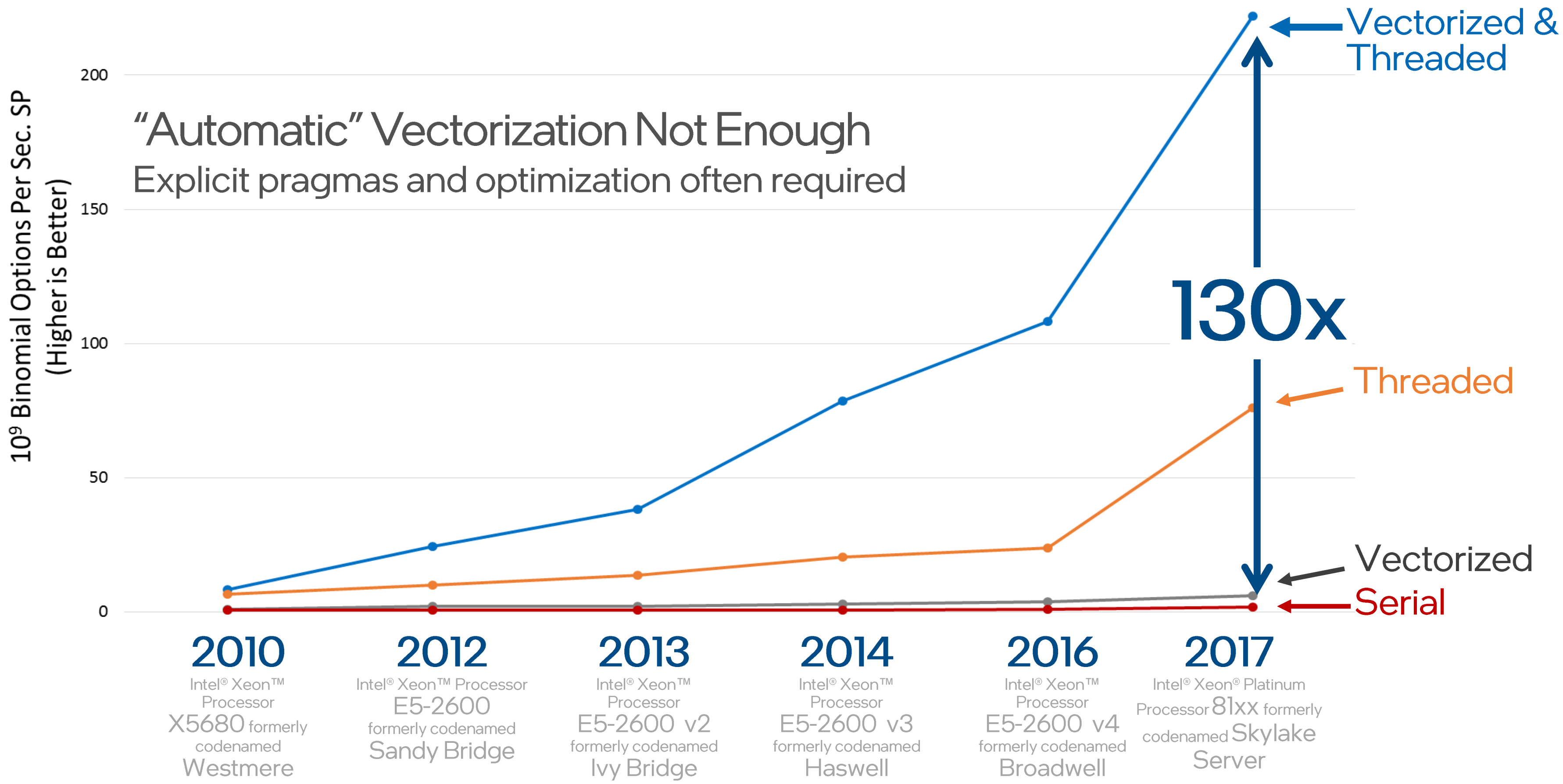
Design your code for high-performance with

Intel® Advisor

intel®

Intel® Advisor: Vectorize & Thread or Performance Dies

Threaded + Vectorized Can Be Much Faster that Either One Alone

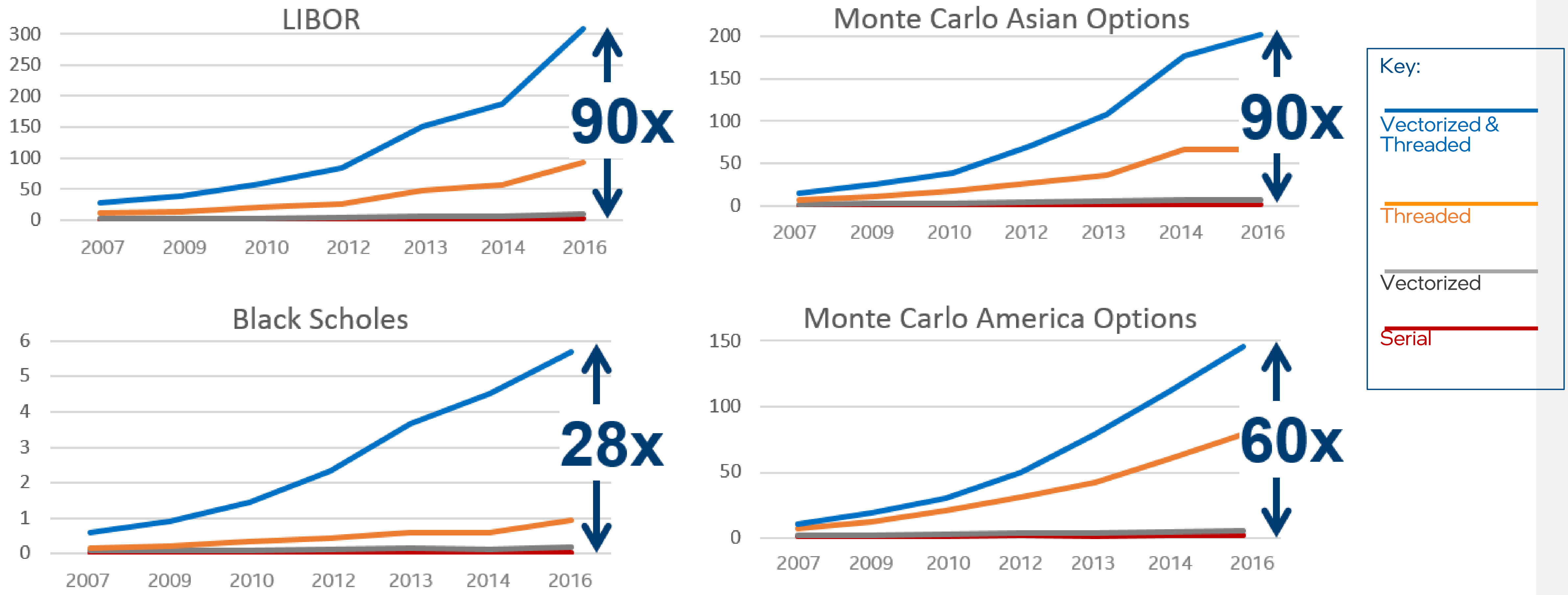


The Difference is Growing with Each New Generation of Hardware

Testing Date: Performance results are **based on testing by Intel employees as of 2017** and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: See Vectorize & Thread or Performance Dies Configurations for 2010-2016 Benchmarks in Backup.

Intel® Advisor: Vectorization & Threading is Critical on Modern Hardware



Testing Date: Performance results are **based on testing by Intel employees as of 2017** and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: See Vectorize & Thread or Performance Dies Configurations for 2010-2016 Benchmarks in Backup.

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. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex. Your costs and results may vary.

“Automatic” Vectorization Often Not Enough

A good compiler can still benefit greatly from vectorization optimization

- Compiler will not always vectorize
  - Check for Loop Carried Dependencies using Intel® Advisor
  - All clear? Force vectorization.
    C++ use: pragma simd, Fortran use: SIMD directive
- Not all vectorization is efficient vectorization
  - Stride of 1 is more cache efficient than stride of 2 and greater. Analyze with Intel® Advisor.
  - Consider data layout changes
    Intel® SIMD Data Layout Templates can help

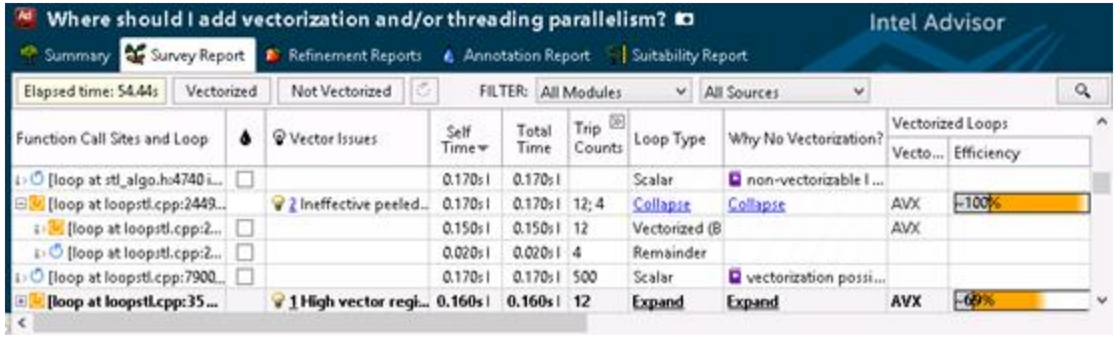
Benchmarks on prior slides did not all “auto vectorize.” Compiler directives were used to force vectorization and get more performance.

Arrays of structures are great for intuitively organizing data, but are much less efficient than structures of arrays. Use the Intel® SIMD Data Layout Templates (Intel® SDLT) to map data into a more efficient layout for vectorization.

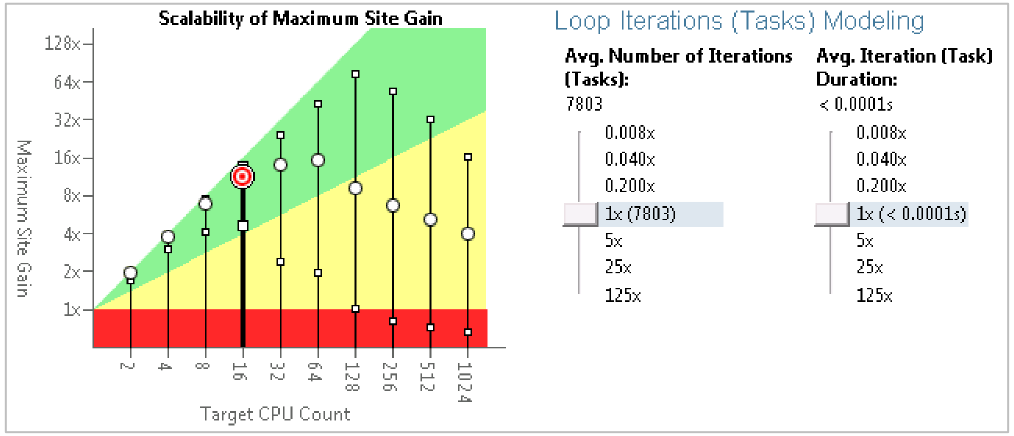
Faster Code Faster with Data Driven Design

Intel® Advisor – Vectorization Optimization and Thread Prototyping

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



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



Less Effort, Less Risk and More Impact

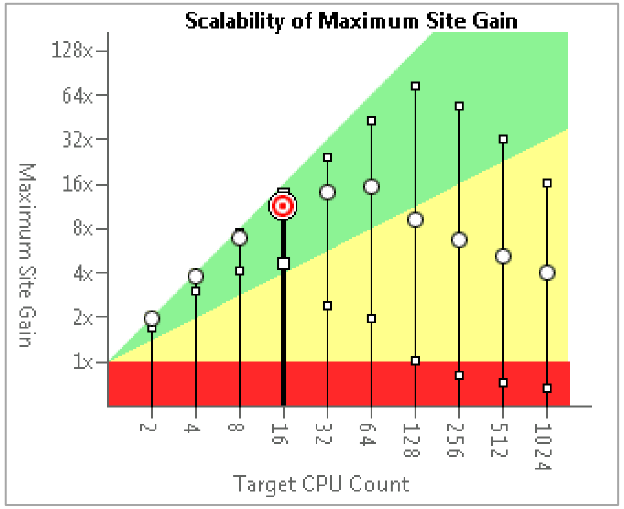
Part of Intel® Parallel Studio for Windows* and Linux*

http://intel.ly/advisor-xe

Get Faster Code Faster! Intel® Advisor

Thread Prototyping

- Have you:
  - Threaded an app, but seen little benefit?
  - Hit a "scalability barrier"?
  - Delayed release due to sync. errors?
- Data Driven Threading Design:
  - Quickly prototype multiple options
  - Project scaling on larger systems
  - Find synchronization errors before implementing threading
  - Design without disrupting development



Add Parallelism with Less Effort, Less Risk and More Impact

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

Simon Hammond

Senior Technical Staff

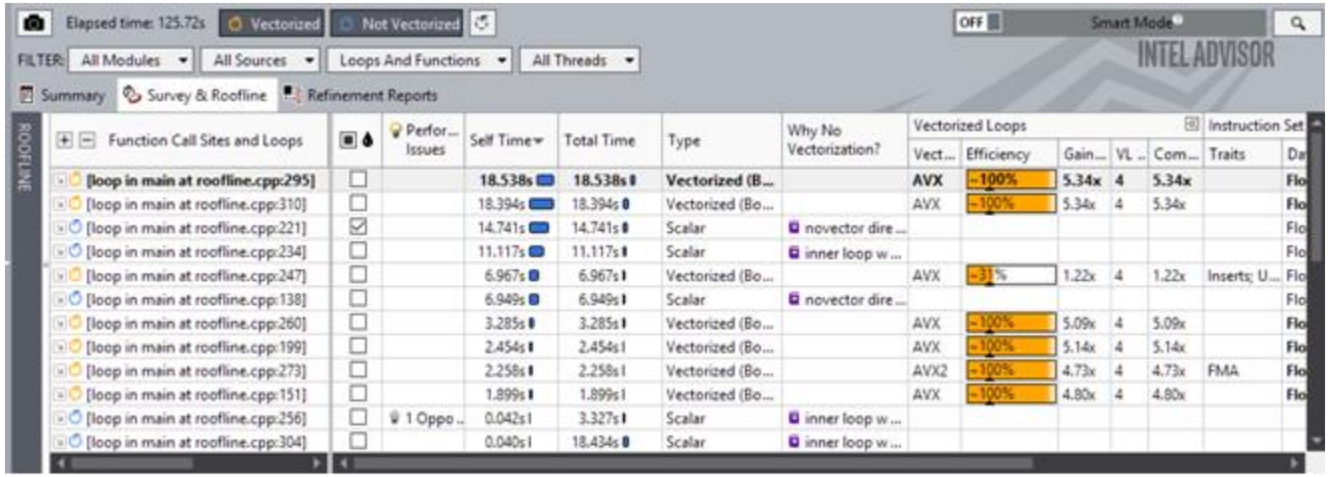
Sandia National Laboratories

Get Faster Code Faster! Intel® Advisor

Vectorization Optimization

- Have you:
  - Recompiled for AVX2 with little gain
  - Wondered where to vectorize?
  - Recoded intrinsics for new arch.?
  - Struggled with compiler reports?

- Data Driven Vectorization:
  - What vectorization will pay off most?
  - What's blocking vectorization? Why?
  - Are my loops vector friendly?
  - Will reorganizing data increase performance?
  - Is it safe to just use pragma simd?



"Intel® Advisor's Vectorization Advisor permitted me to focus my work where it really mattered. When you have only a limited amount of time to spend on optimization, it is invaluable."

Gilles Civario

Senior Software Architect

Irish Centre for High-End Computing

Vector Instructions are Dramatically Faster

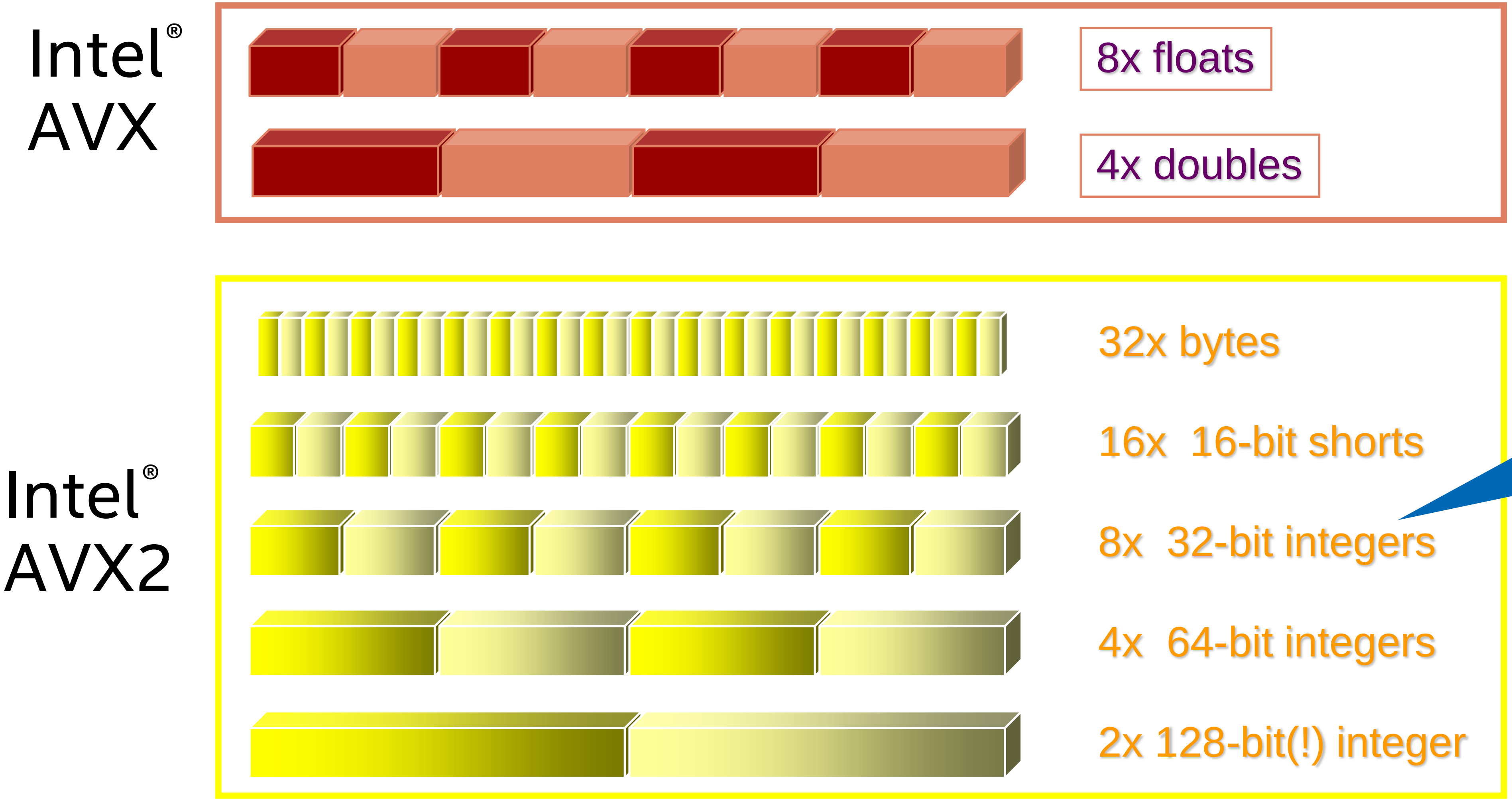
Multiple arithmetic operations with a single instruction

Adding 2 vectors

+	4.4	1.1	3.1	-8.5	-1.3	1.7	7.5	5.6	-3.2	3.6	4.8
	-0.3	-0.5	0.5	0	0.1	0.8	0.9	0.7	1	0.6	-0.5
=	4.1	0.6	3.6	-8.5	-1.2	2.5	8.4	6.3	-2.2	4.2	4.3

- These instructions are also referred to as Single Instruction Multiple Data (SIMD instructions)

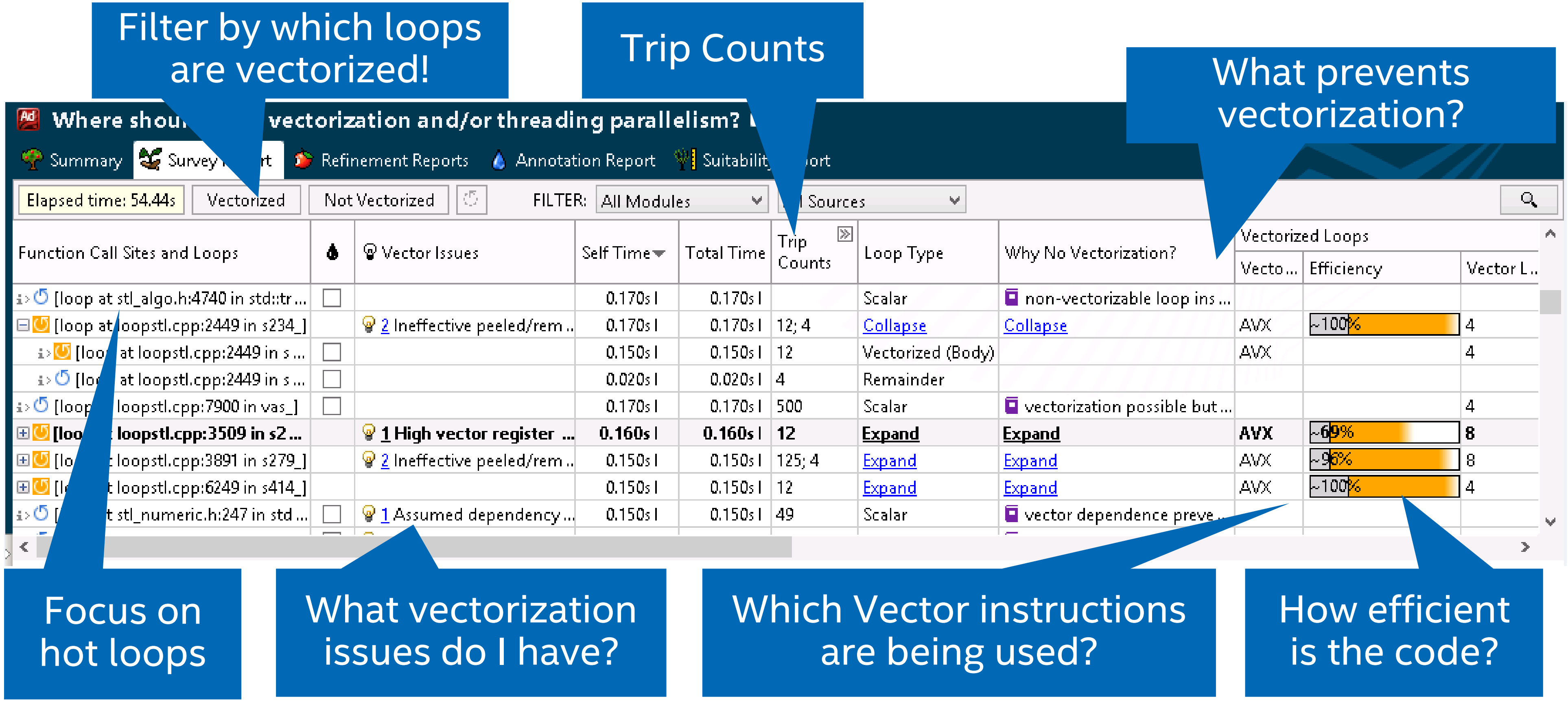
Intel® Advanced Vector Extensions (Intel® AVX)



Vector length – the number of elements that can processed

The Right Data At Your Fingertips

Get all the data you need for high impact vectorization



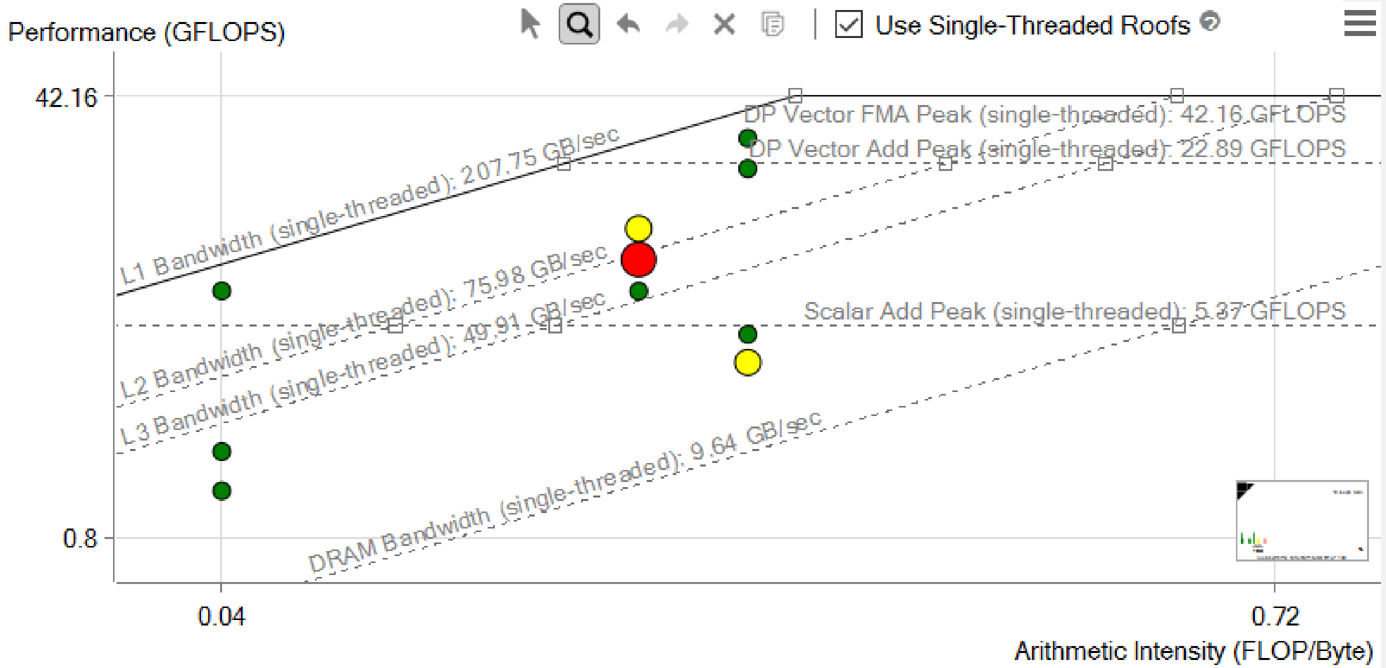
Get Faster Code Faster!

Intel® Advisor

Roofline Analysis

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?



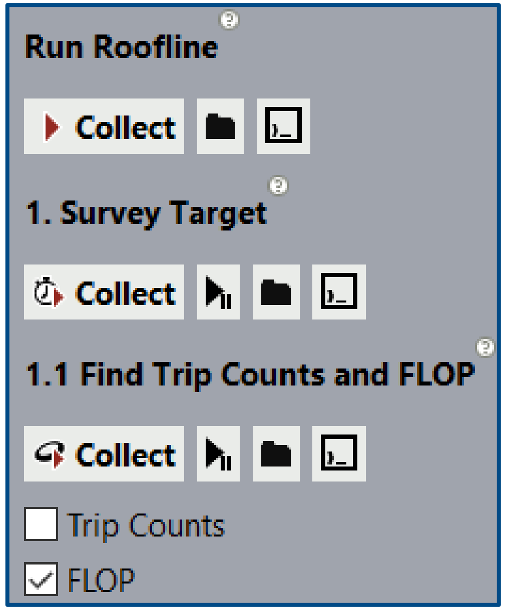
Roofline first proposed by University of California at Berkeley: [*Roofline: An Insightful Visual Performance Model for Multicore Architectures*](), 2009

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.

Classic vs. Cache-Aware Roofline

- Intel® Advisor uses the Cache-Aware Roofline model, which has a different definition of Arithmetic Intensity than the original (“Classic”) model.

Classical Roofline

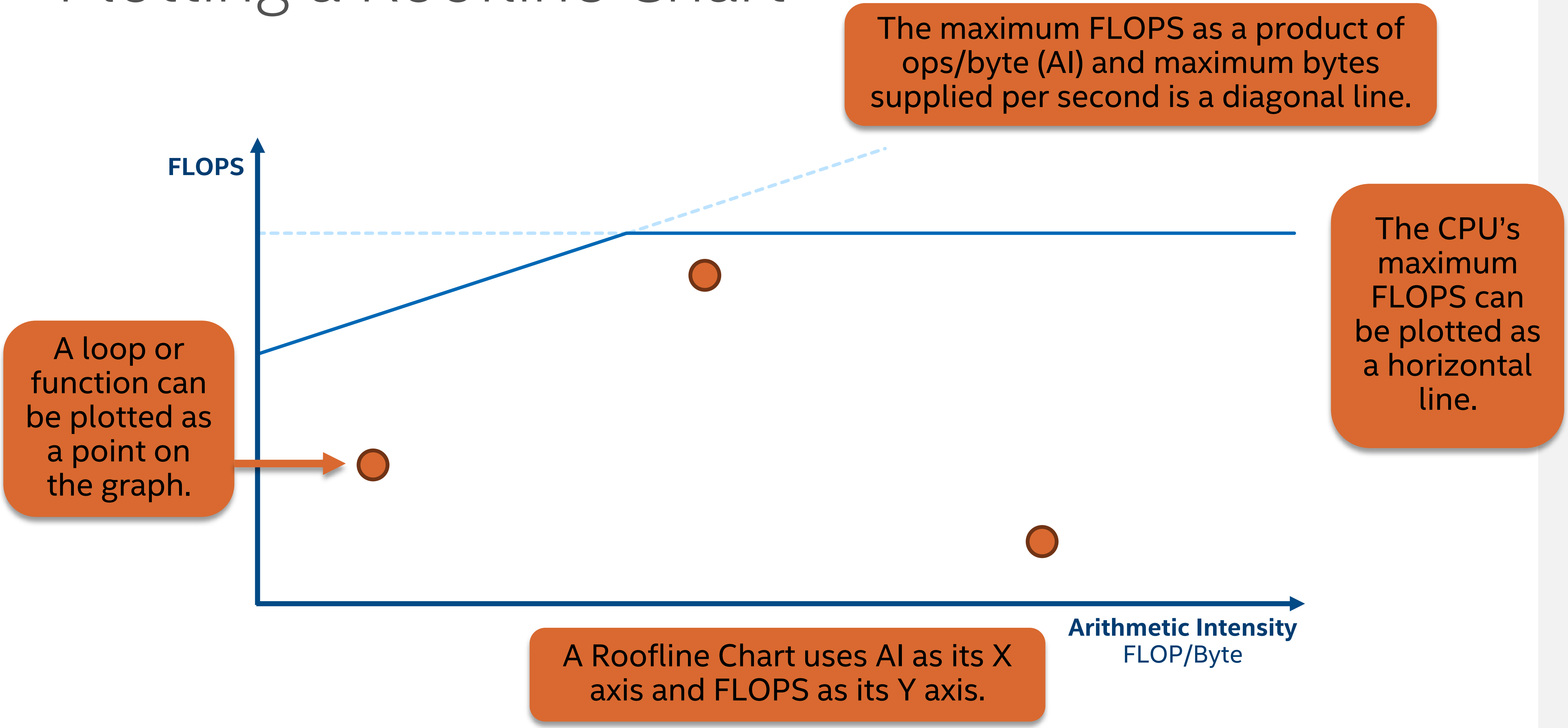
- Traffic measured from one level of memory (usually DRAM)
- AI may change with data set size
- AI changes as a result of memory optimizations

Cache-Aware Roofline

- Traffic measured from all levels of memory
- AI is tied to the algorithm and will not change with data set size
- Optimization does not change AI*, only the performance

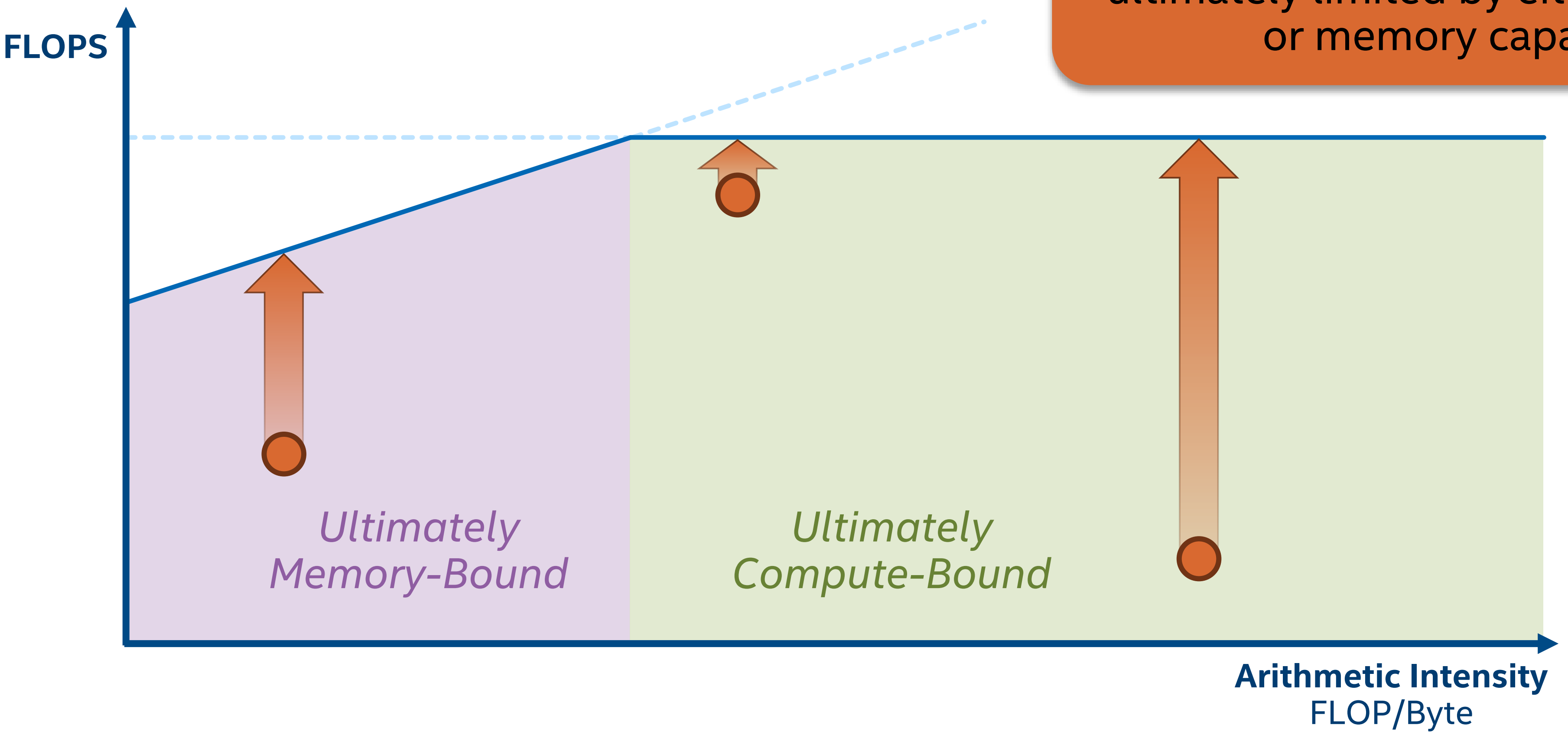
**Compiler optimizations may modify the algorithm, which may change the AI.*

Plotting a Roofline Chart

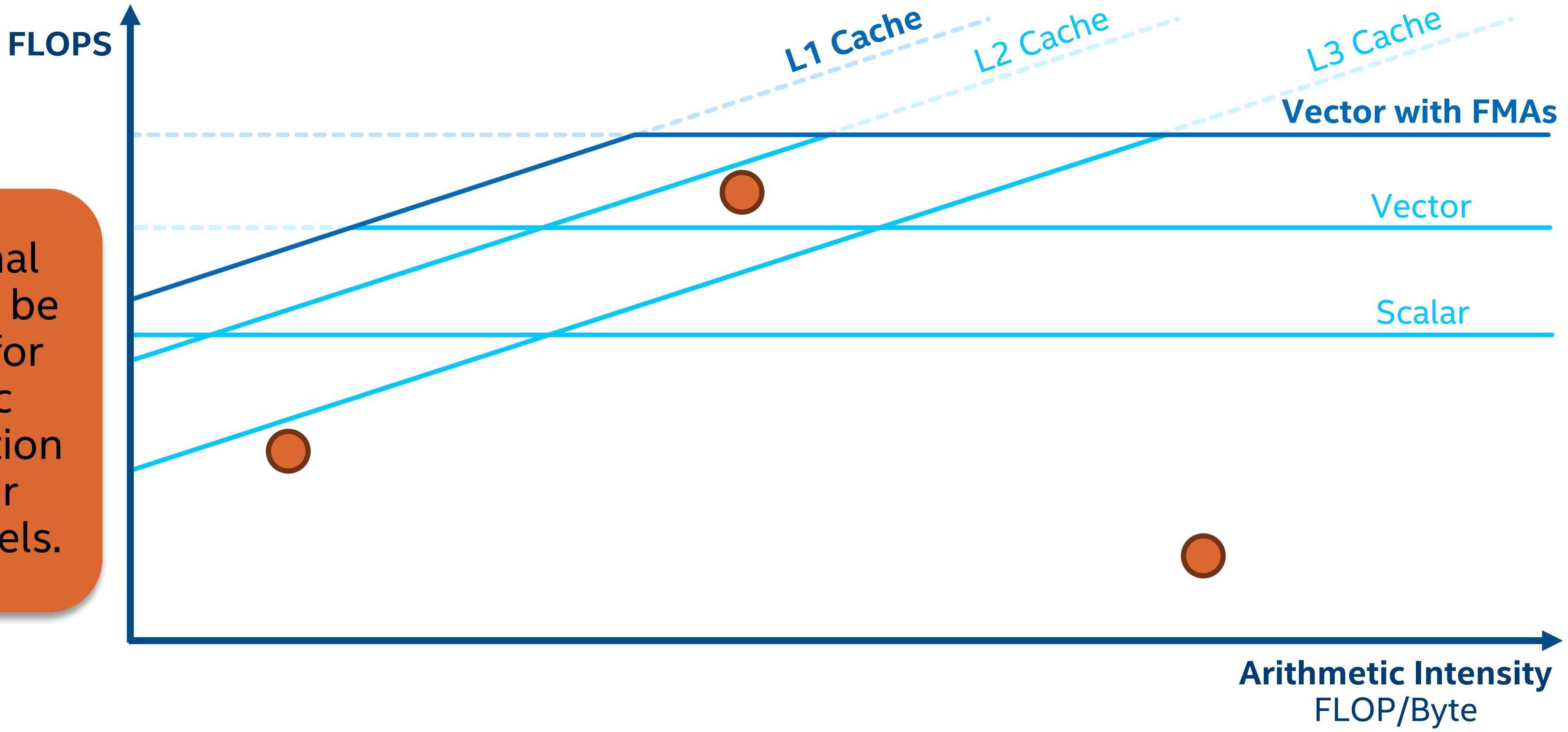


Ultimate Performance Limits

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



Sub-Roofs and Current Limits

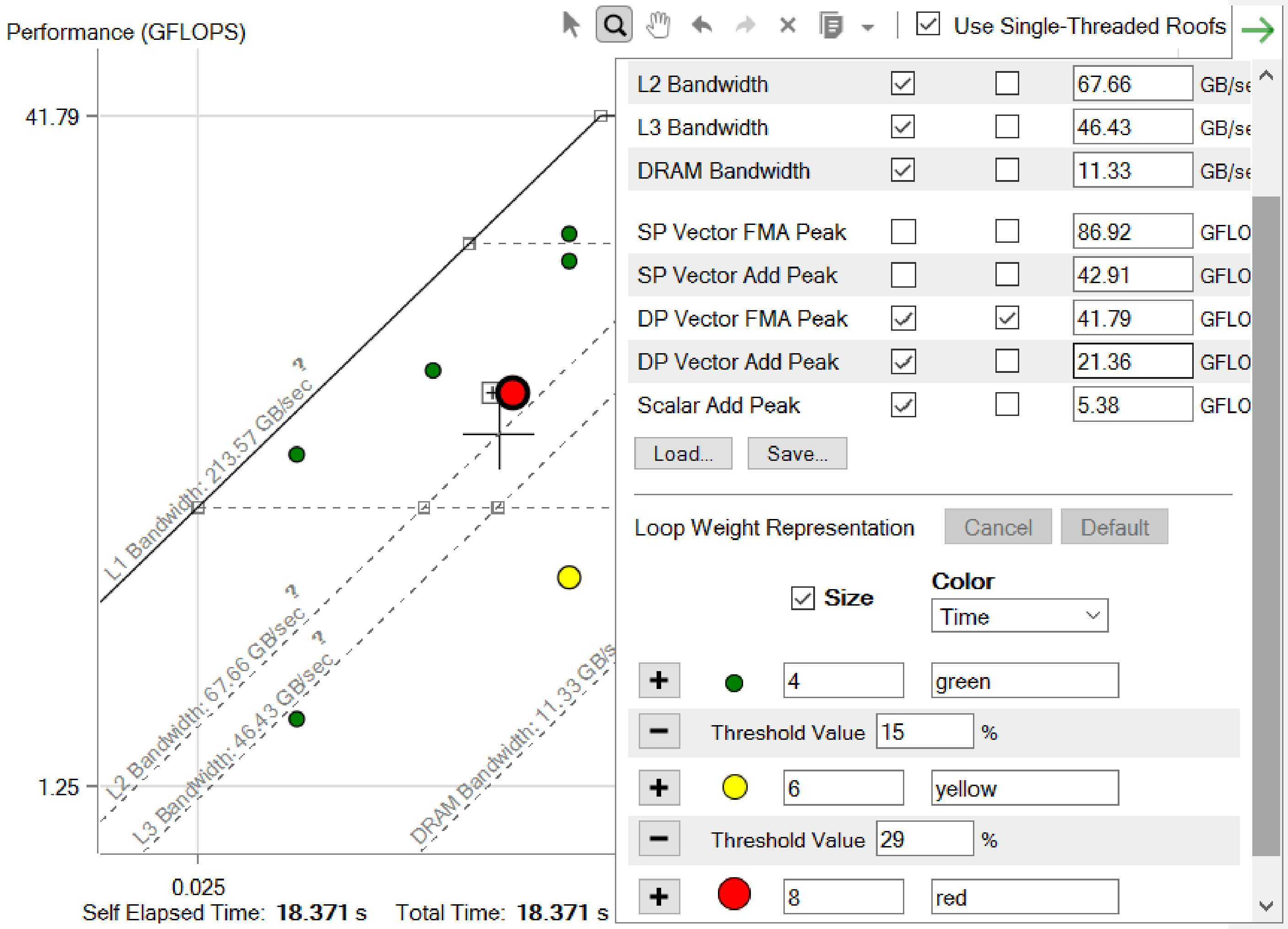


Additional roofs can be plotted for specific computation types or cache levels.

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

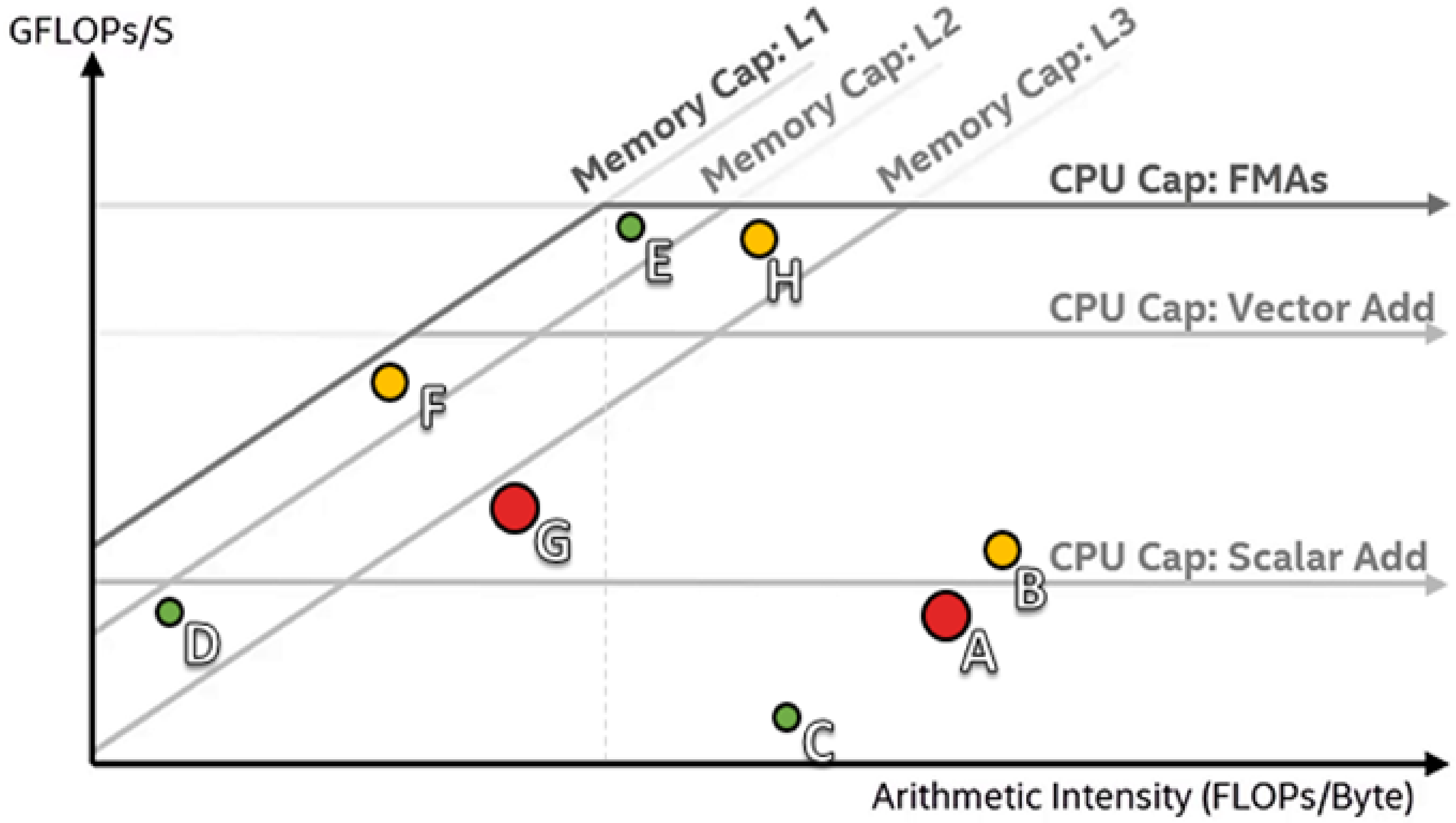
The Intel® Advisor Roofline Interface

- 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.



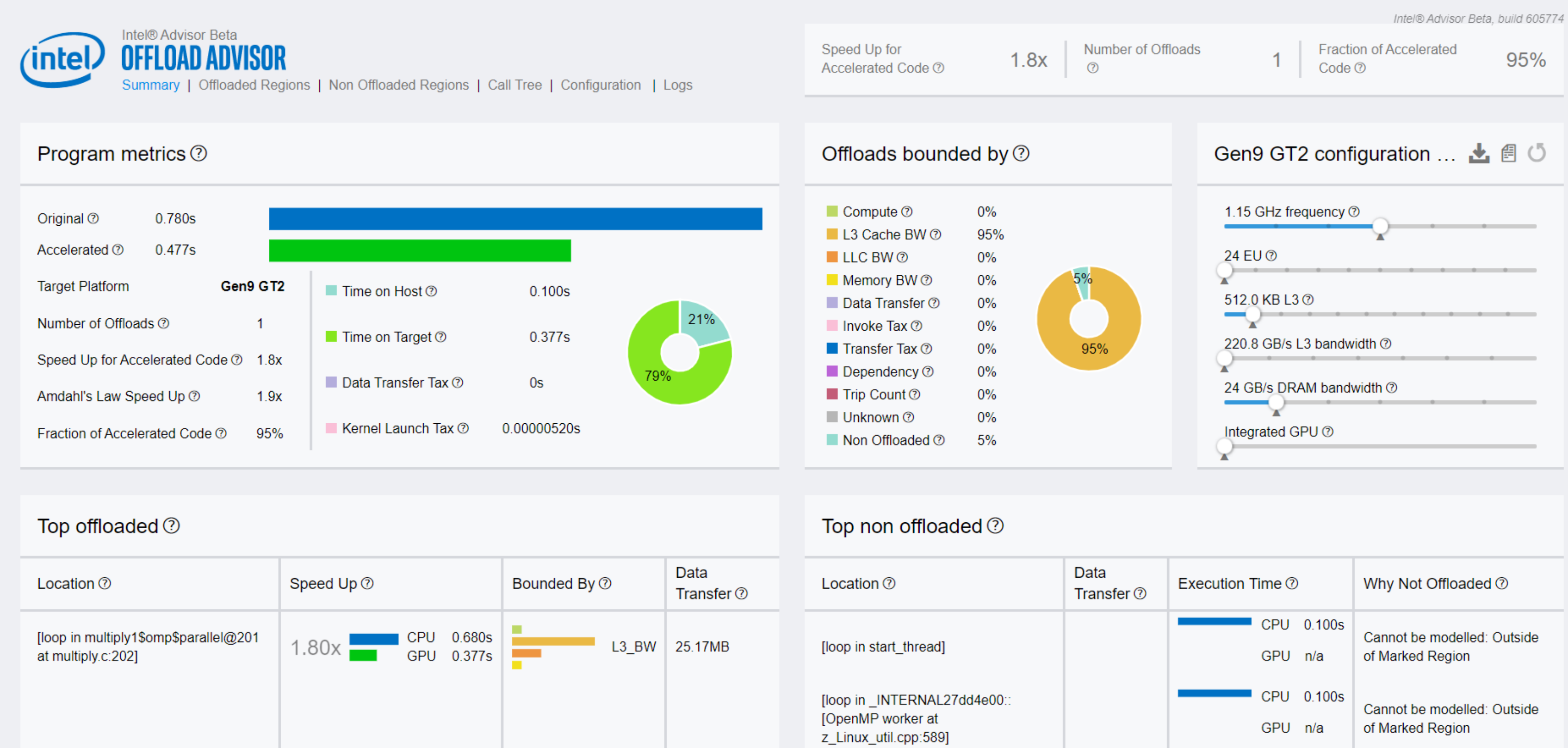
Intel® Advisor

Offload Advisor

Intel® Advisor - Offload Advisor

Starting from an optimized binary (running on CPU):

- Helps define which sections of the code should run on a given accelerator
- Provides performance projection on accelerators



Intel® Advisor - Offload Advisor

Find code that can be profitably offloaded

Speed Up for Accelerated Code ⓘ	1.8x	Number of Offloads ⓘ	1	Fraction of Accelerated Code ⓘ	95%

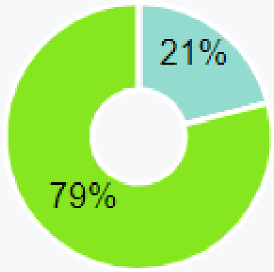
Speedup of accelerated code 1.8 x

Program metrics ⓘ

Original ⓘ	0.780s
Accelerated ⓘ	0.477s

Target Platform	**Gen9 GT2**
Number of Offloads ⓘ	1
Speed Up for Accelerated Code ⓘ	1.8x
Amdahl's Law Speed Up ⓘ	1.9x
Fraction of Accelerated Code ⓘ	95%

Time on Host ⓘ	0.100s
Time on Target ⓘ	0.377s
Data Transfer Tax ⓘ	0s
Kernel Launch Tax ⓘ	0.00000520s



How to Run Intel® Advisor – Offload Advisor

- source <advisor_install_dir>/advixe-vars.sh
- `advixe-python $APM/collect.py advisor_project --config gen9 -- /home/test/matrix`
- `advixe-python $APM/analyze.py advisor_project --config gen9 --out-dir /home/test/analyze`
- View the report.html generated (or generate a command-line report)

Analyze for a specific GPU config

Compare Acceleration on Different GPUs

Gen9 – Not profitable to offload kernel

Speed Up for Accelerated Code	1.0x	Number of Offloads	0	Fraction of Accelerated Code	0%

Program metrics

Original	0.15s
Accelerated	0.15s

Target Platform	Gen9 GT2	Time on Host	0.15s
Number of Offloads	0	Time on Accelerator	0s
Speed Up for Accelerated Code	1.0x	Data Transfer Tax	s
Amdahl's Law Speed Up	1.0x	Invocation Tax	s
Fraction of Accelerated Code	0%	Code Transfer Tax	s

100%

CPU	0.14s	Not profitable: Computation Time is high despite the full use of Target Device capabilities
GPU	0.167	

Gen11 – 1.6x speedup

Speed Up for Accelerated Code	1.6x	Number of Offloads	1	Fraction of Accelerated Code	98%

Program metrics

Original	0.15s
Accelerated	0.10s

Target Platform	Gen11 GT2	Time on Host	<0.01s
Number of Offloads	1	Time on Accelerator	0.09s
Speed Up for Accelerated Code	1.6x	Data Transfer Tax	0s
Amdahl's Law Speed Up	1.7x	Invocation Tax	<0.01s
Fraction of Accelerated Code	98%	Code Transfer Tax	<0.01s

10%
90%

1.60x	CPU	0.14s	Compute	1.15MB
	GPU	0.09s		

Intel® Advisor

Demo

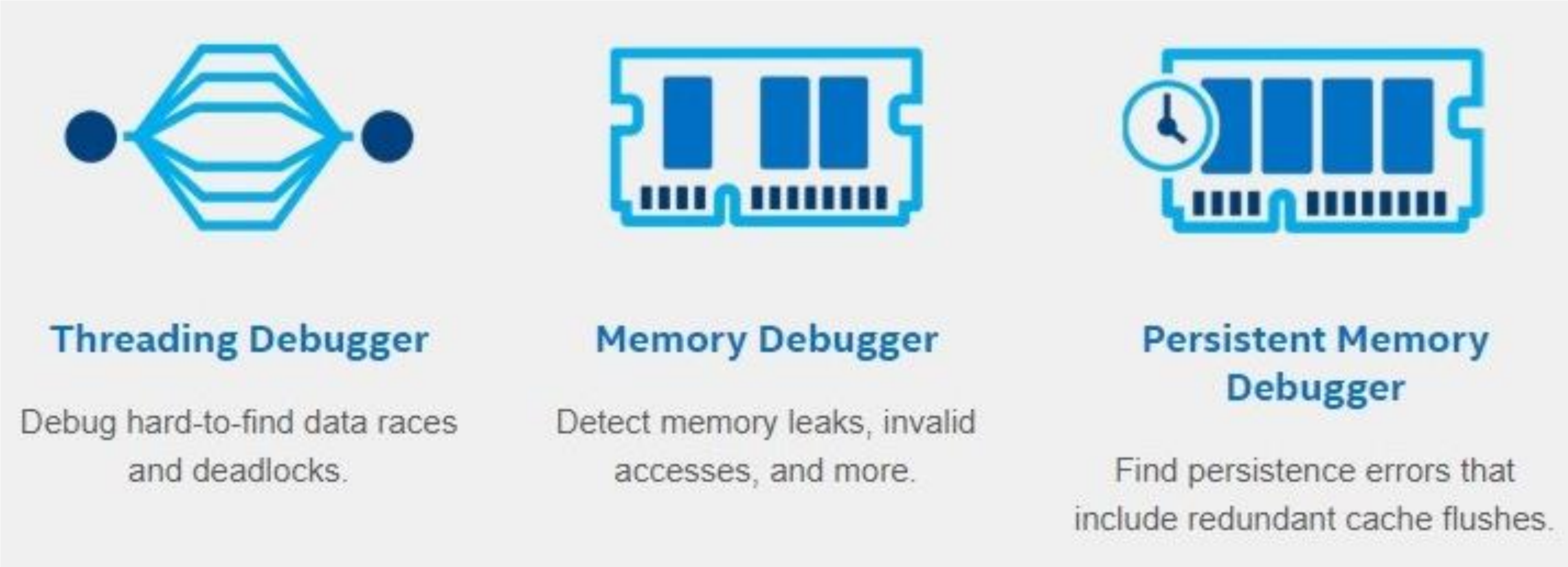
Deliver reliable applications with

Intel® Inspector

intel®

Threading, Memory and Persistence Debugger

Intel® Inspector



Race Conditions Are Difficult to Diagnose

They only occur occasionally and are difficult to reproduce

Correct Answer

Thread 1	Thread 2		Shared Counter
			0
Read count		🡐	0
Increment			0
Write count		🡒	1
	Read count	🡐	1
	Increment		1
	Write count	🡒	**2**

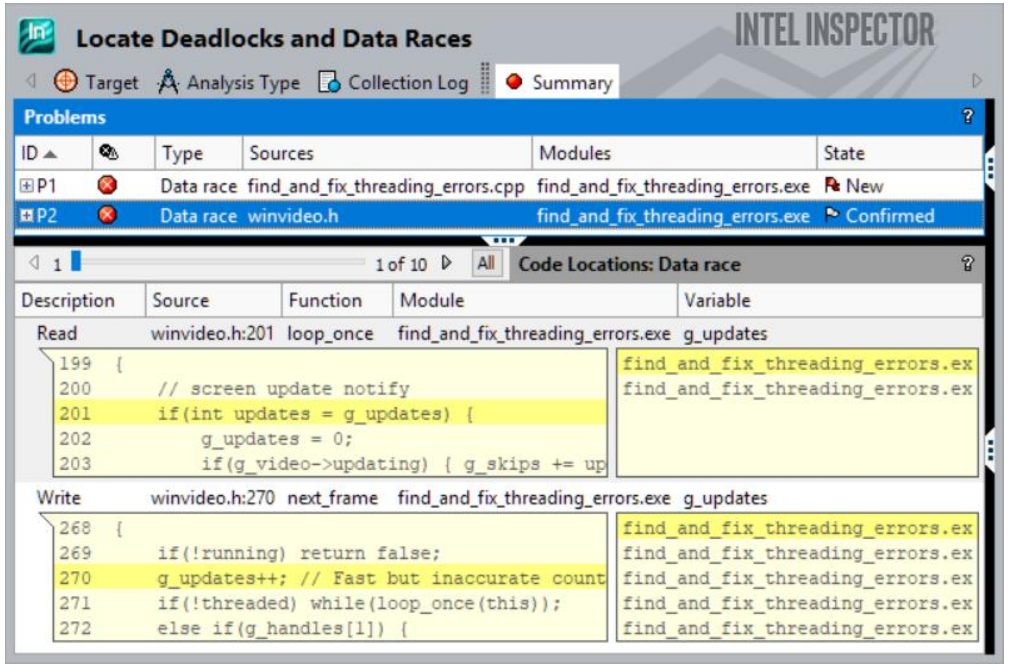
Incorrect Answer

Thread 1	Thread 2		Shared Counter
			0
Read count		🡐	0
	Read count	🡐	0
Increment			0
	Increment		0
Write count		🡒	1
	Write count	🡒	**1**

Intel® Inspector

Find & Debug Memory and Threading Errors

- Correctness Tools Increase ROI by 12%-21%[1]
  - Errors found earlier are less expensive to fix
  - Races & deadlocks not easily reproduced
  - Memory errors are hard to find without a tool
- Faster Diagnosis with Debugger Breakpoints
  - Breakpoint set just before the problem occurs
  - Examine variables and threads with the debugger
- Debug Persistent Memory Errors
  - Missing cache flushes / store fences and more
- New in 2021 release:
  - Preview: Memory and threading errors analysis for DPC++ and OpenMP offloaded codes, executed on CPU target.



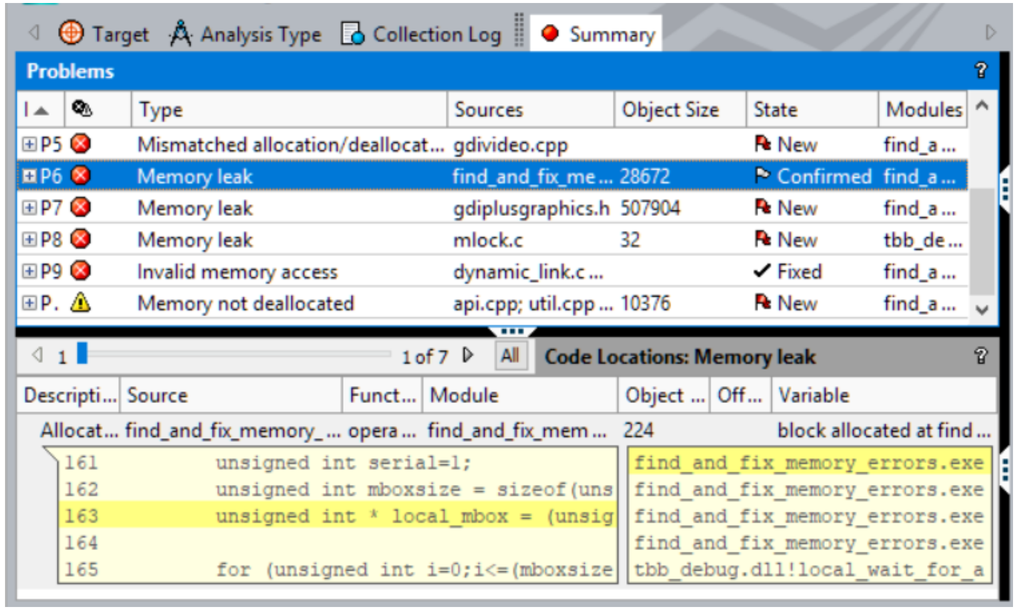
[1]**Cost Factors – Square Project Analysis -** *CERT: U.S. Computer Emergency Readiness Team, and Carnegie Mellon CyLab NIST: National Institute of Standards & Technology: Square Project Results*

intel.

Debug Memory & Threading Errors

Intel® Inspector

- Find and eliminate errors
  - Memory leaks, invalid access…
  - Races & deadlocks
  - C, C++ and Fortran (or a mix)
- Simple, Reliable, Accurate
  - No special recompiles
    Use any build, any compiler[1]
  - Analyzes dynamically generated or linked code
  - Inspects 3rd party libraries without source
  - Productive user interface + debugger integration
  - Command line for automated regression analysis



Clicking an error instantly displays source code snippets and the call stack

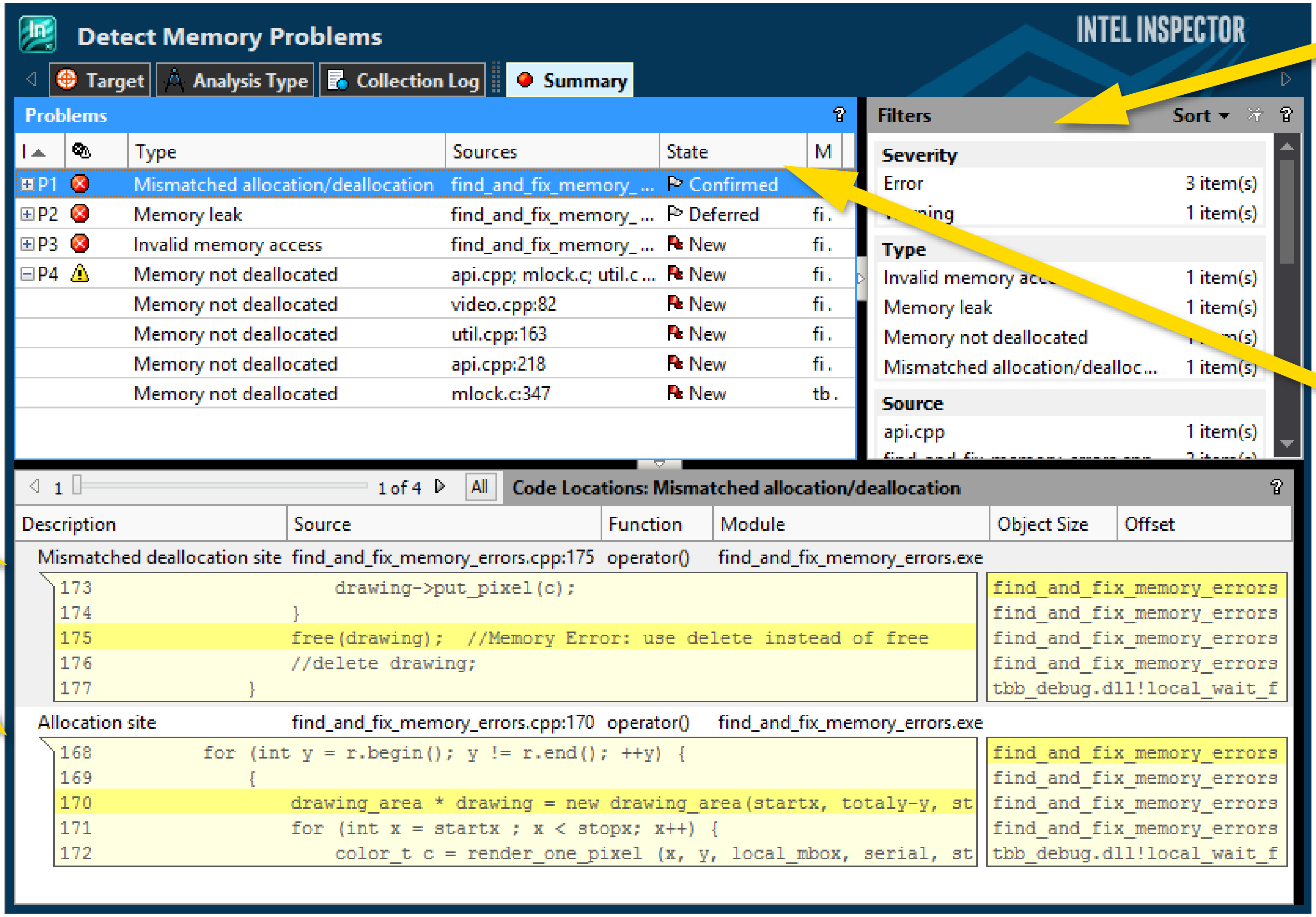
Fits your existing process

[1]Compilers that follows common OS standards.

Productive User Interface Saves Time

Intel® Inspector

Select a problem set



Filters let you focus on a module, or error type, or just the new errors or…

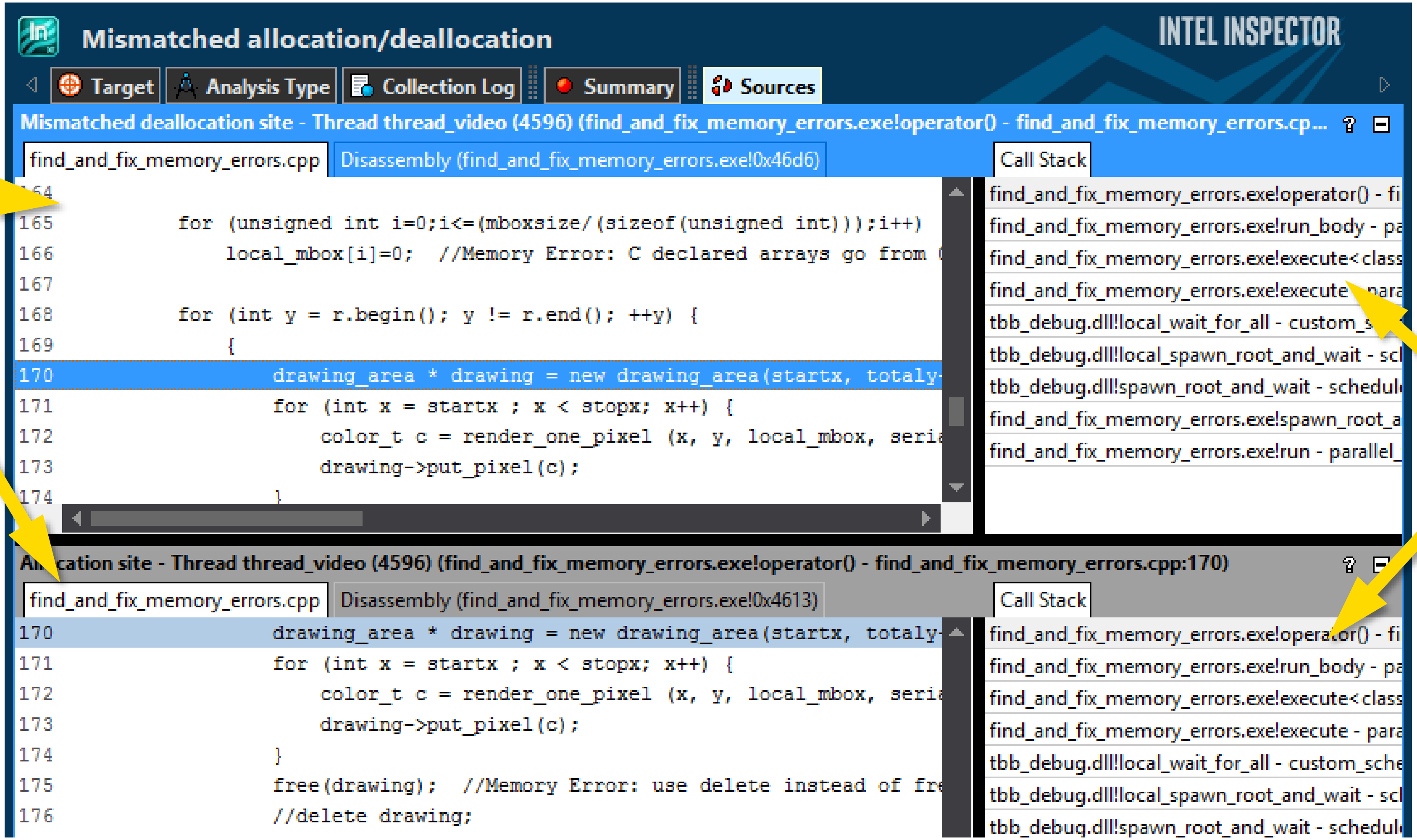
Problem States: New, Not Fixed, Fixed, Confirmed, Not a problem, Deferred, Regression

Code snippets displayed for selected problem

Double Click for Source & Call Stack

Intel® Inspector

Source code locations displayed for selected problem

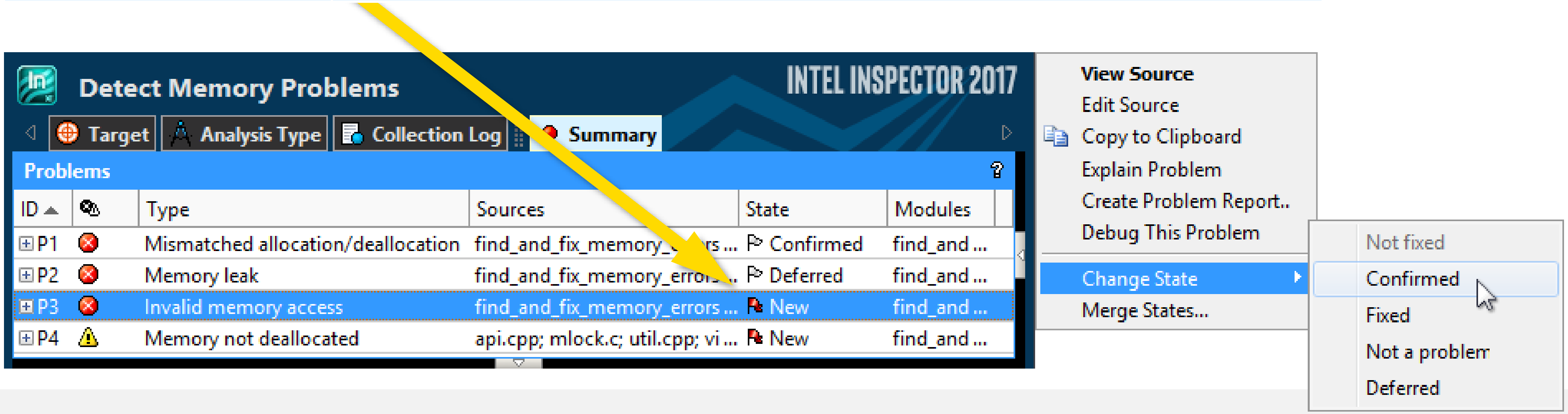


Call Stack

Easy Problem Management

Quickly see new problems and regressions

State	Description
New	Detected by this run
Not Fixed	Previously seen error detected by this run
Not a Problem	Set by user (tool will not change)
Confirmed	Set by user (tool will not change)
Fixed	Set by user (tool will change)
Regression	Error detected with previous state of “Fixed”

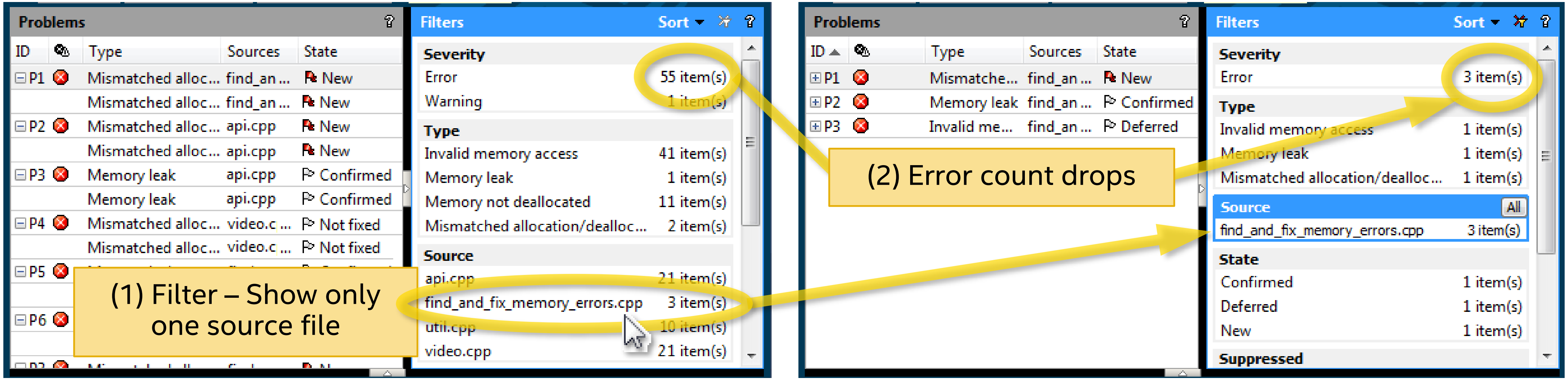


Filtering - Focus on What's Important

Example: See only the errors in one source file

Before – All Errors

After – Only errors from one source file



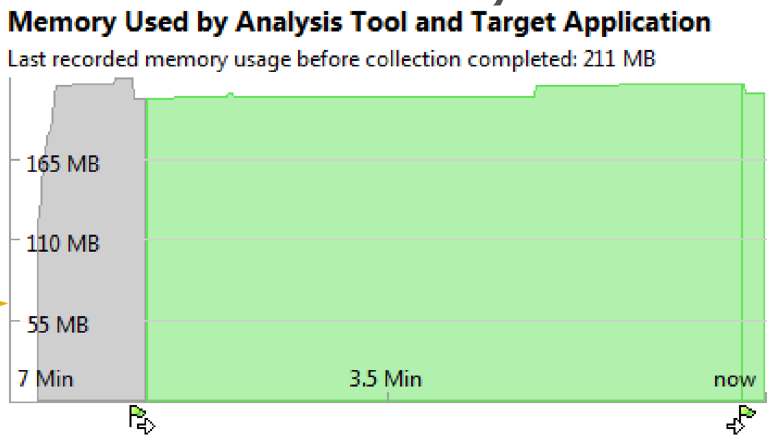
Tip: Set the "Investigated" filter to "Not investigated" while investigating problems.
This removes from view the problems you are done with, leaving only the ones left to investigate.

Incrementally Diagnose Memory Growth

Intel® Inspector

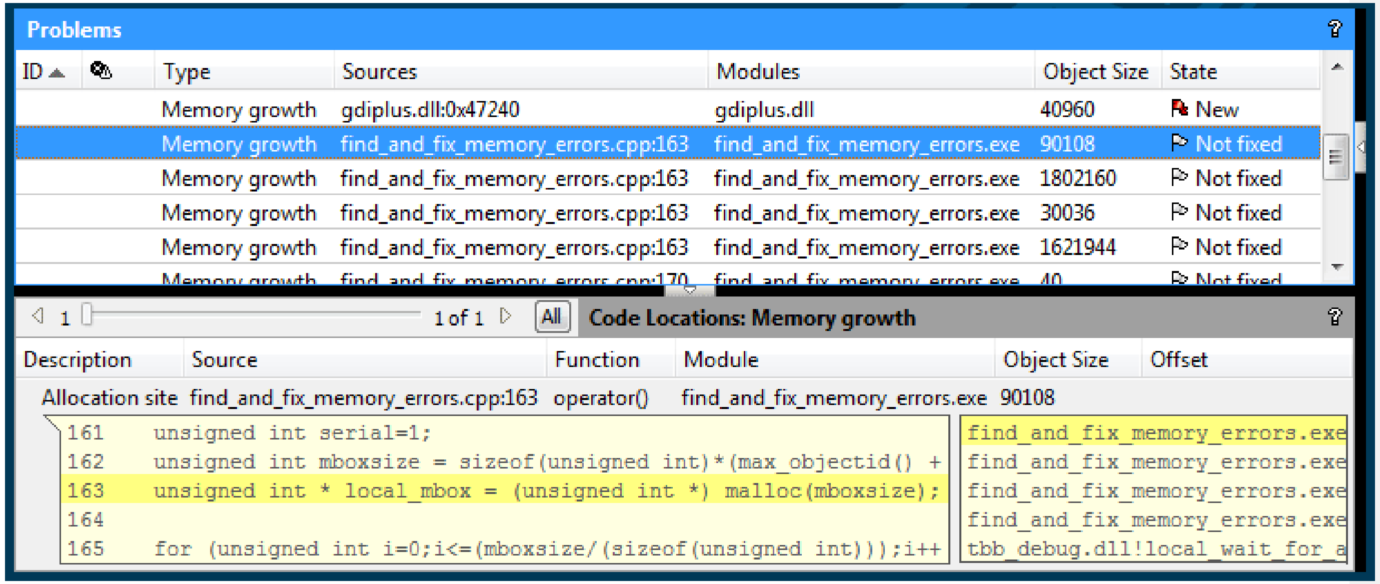
As your app is running…

Memory usage graph plots memory growth



Select a cause of memory growth

See the code snippet & call stack



Speed diagnosis of difficult to find heap errors

Automate Regression Analysis

Command Line Interface

- inspxe-cl is the command line:
  - **Windows:** `C:\Program Files\Intel\Inspector XE \bin[32|64]\inspxe-cl.exe`
  - **Linux:** `/opt/intel/inspector_xe/bin[32|64]/inspxe-cl`
- Help:

```
inspxe-cl –help
```

- Set up command line with GUI
- Command examples:

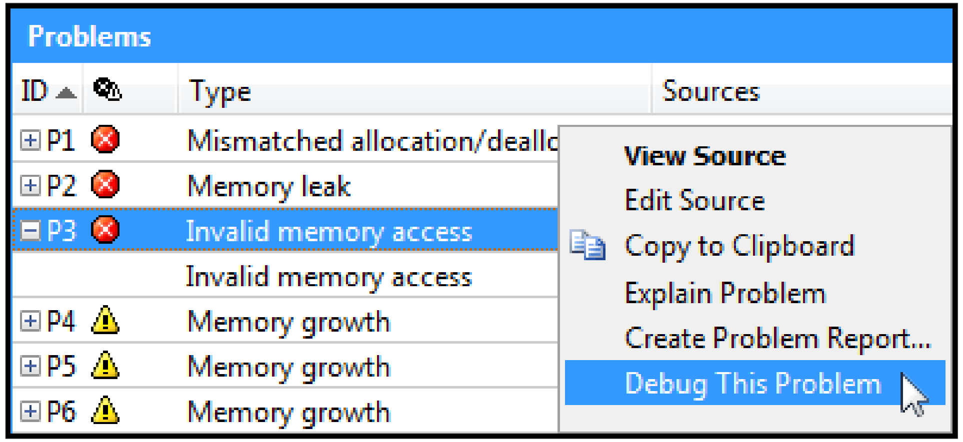
```
1.inspxe-cl -collect-list
2. inspxe-cl –collect ti2 -- MyApp.exe
3. inspxe-cl –report problems
```

Send results file to developer to analyze with the UI

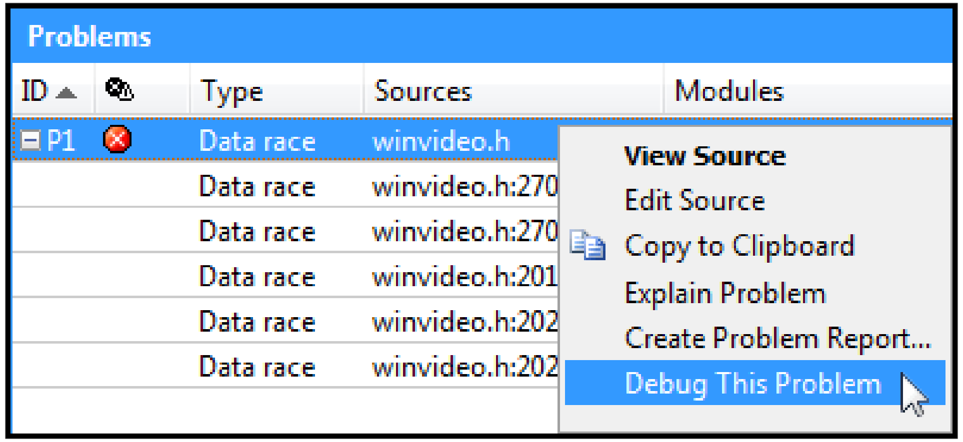
Break At Just The Right Time

Intel® Inspector - Memory & Thread Debugger

Memory Errors



Threading Errors



- Break into the debugger just before the error occurs.
- Examine the variables and threads.
- Diagnose the problem.

Save time. Find and diagnose errors with less effort.

Productive Memory & Threading Debugger

Intel® Inspector

	Memory Analysis	Threading Analysis
View Context of Problem Stack Multiple Contributing Source Locations	 ✓ ✓	 ✓ ✓
Collapse multiple “sightings” to one error (e.g., memory allocated in a loop, then leaked is 1 error)	✓	✓
Suppression, Filtering, and Workflow Management	✓	✓
Visual Studio* Integration (Windows*)	✓	✓
Command line for automated tests	✓	✓
Timeline visualization	✓	✓
Memory Growth during a transaction	✓	
Trigger Debugger Breakpoint	✓	✓

Easier & Faster Debugging of Memory & Threading Errors