

Intel® oneAPI Performance Analysis Tools

Dmitry Tarakanov
Technical Consulting Engineer



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.

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

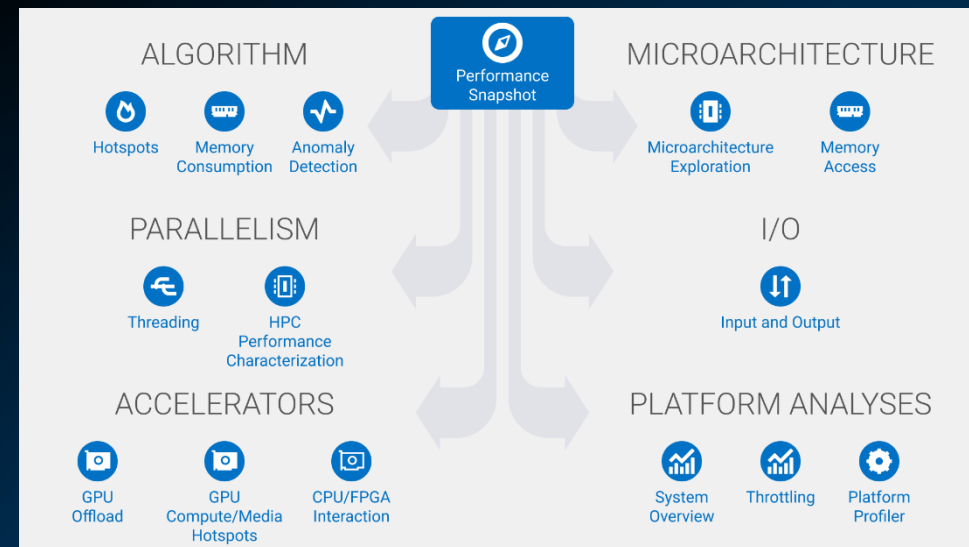
- A suite of profiling for CPU, GPU, FPGA, threading, memory, cache, storage, offload, power...
- DPC++, C, C++, Fortran, Python*, Go*, Java*, or a mix
- Linux, Windows, FreeBSD, Android, Yocto and more

Analyze Data Faster

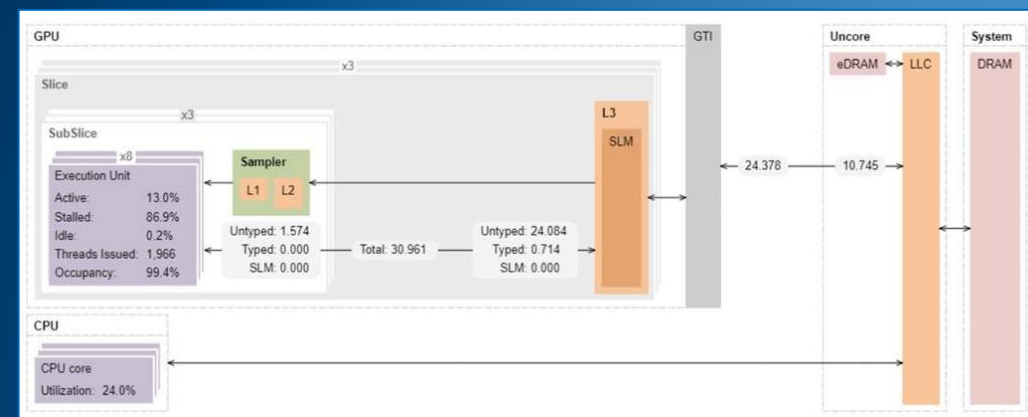
- See data on your source, in architecture diagrams, as a histogram, on a timeline...
- Filter and organize data to find answers

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



| Source | | Assembly | | GPU Instructions Executed by Instruction T... | |
|--------|------------------------------------|----------|--|--|--|
| ... | | ... | | Control Flow Send & Wait Int32 & SP Float Int64 & DP Float Other | |
| 158 | dx = ptr[j].pos[0] - ptr[i].pos[0] | | | 75,002,500 | |
| 159 | dy = ptr[j].pos[1] - ptr[i].pos[1] | | | 12,500,000 | |
| 160 | dz = ptr[j].pos[2] - ptr[i].pos[2] | | | 12,500,000 | |



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.



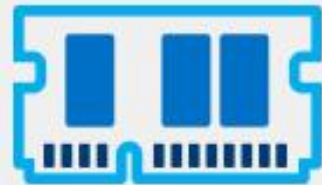
Media & OpenCL™ Applications

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

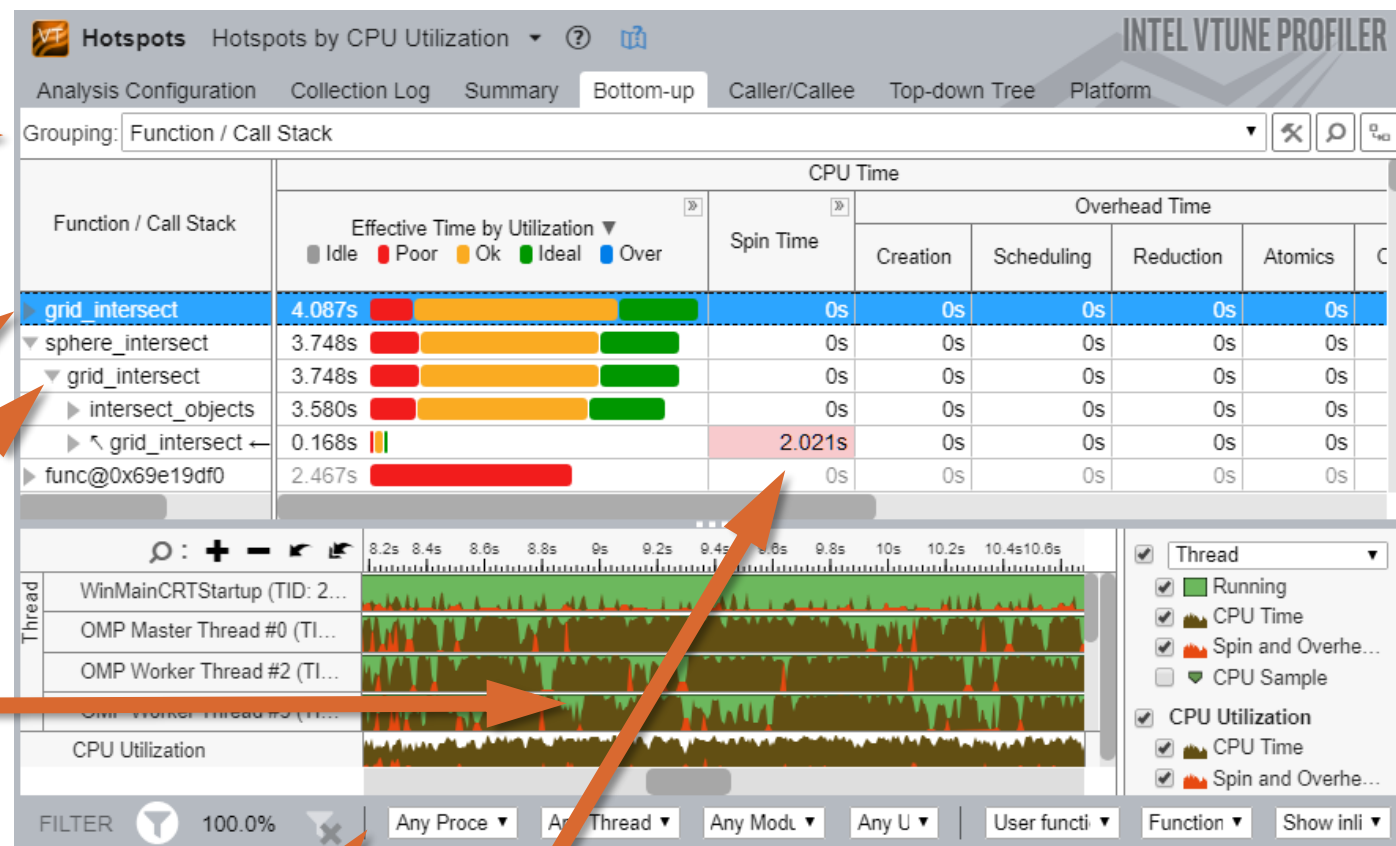
Function / Call Stack
Source Function / Function / Call Stack
Sync Object / Function / Call Stack
Sync Object / Thread / Function / Call Stack
... (Partial list shown)

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

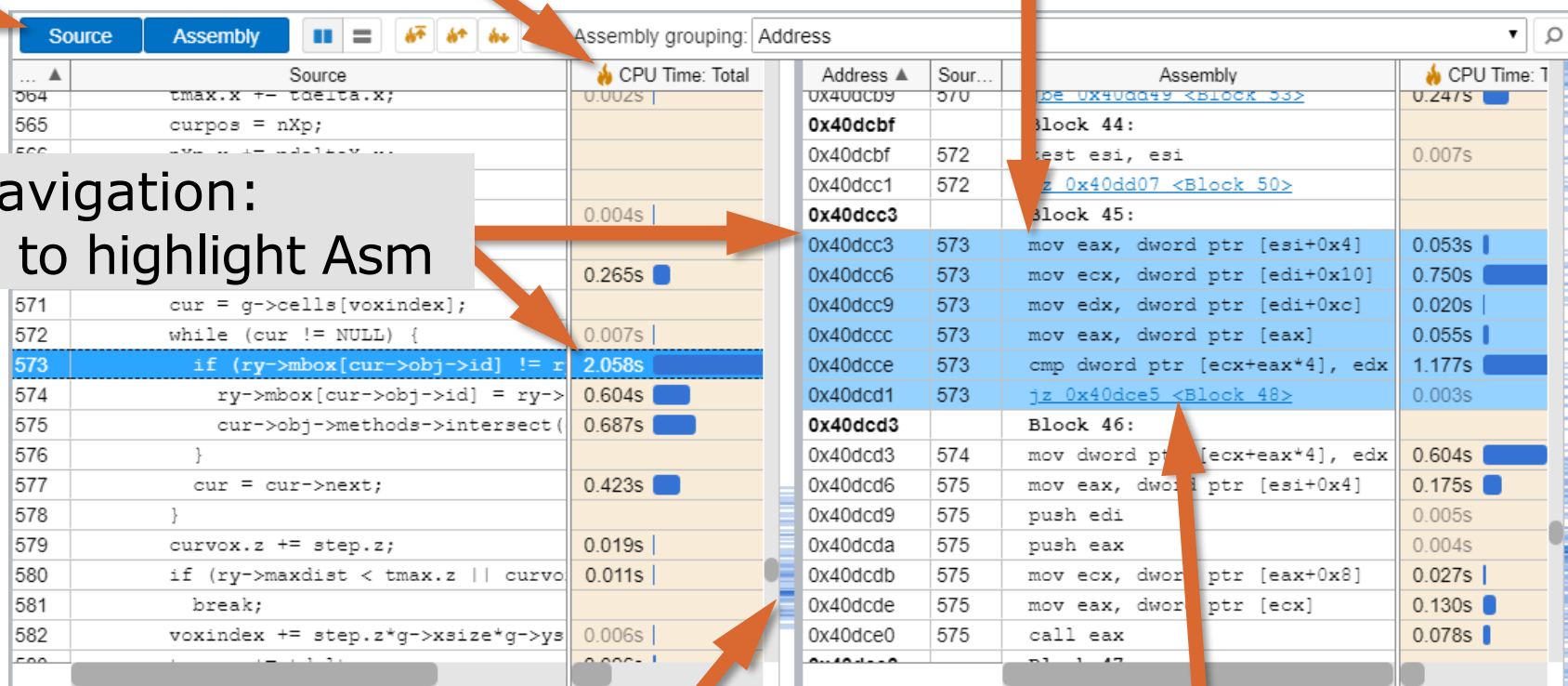
Double Click from Grid or Timeline

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

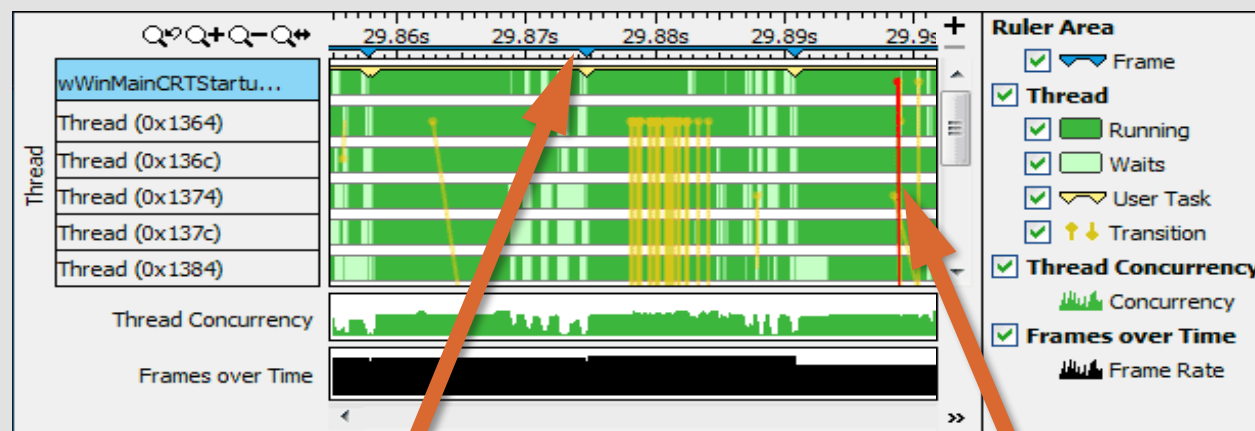
Click jump to scroll Asm

Timeline Visualizes Thread Behavior

Intel® VTune™ Profiler

🔑🔑 Transitions

Locks & Waits



Hovers:

🔑 Frame

Frame
Start: 29.858s Duration: 0.017s
Frame: 72
Frame Domain: Smoke::Framework::execute()
Frame Type: Good
Frame Rate: 59.8242179

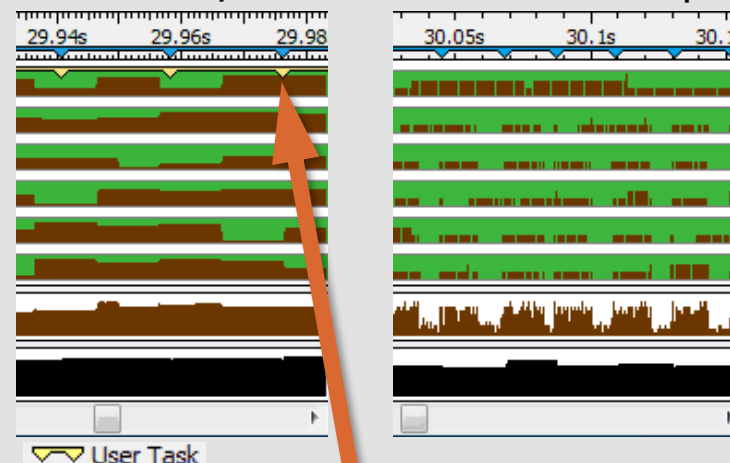
🔑 Transition

Transition
wWinMainCRTStartup (0x12d4) to Thread (0x138c) (29.899s to 29.899s)
Sync Object: TBB Scheduler
Object Creation File: taskmanagertbb.cpp
Object Creation Line: 318

🏠 CPU Time

Basic Hotspots

Advanced Hotspots



🔑 User Task

User Task
Start: 29.958s Duration: 0.018s
Task Type: Smoke::FrameWork::execute()::Other
Task End Call Stack: Framework::Execute

CPU Time
94.233472%

- Optional: Use API to mark frames and user tasks
- Optional: Add a mark during collection 📍

🔑 Frame

🔑 User Task

Command Line Interface

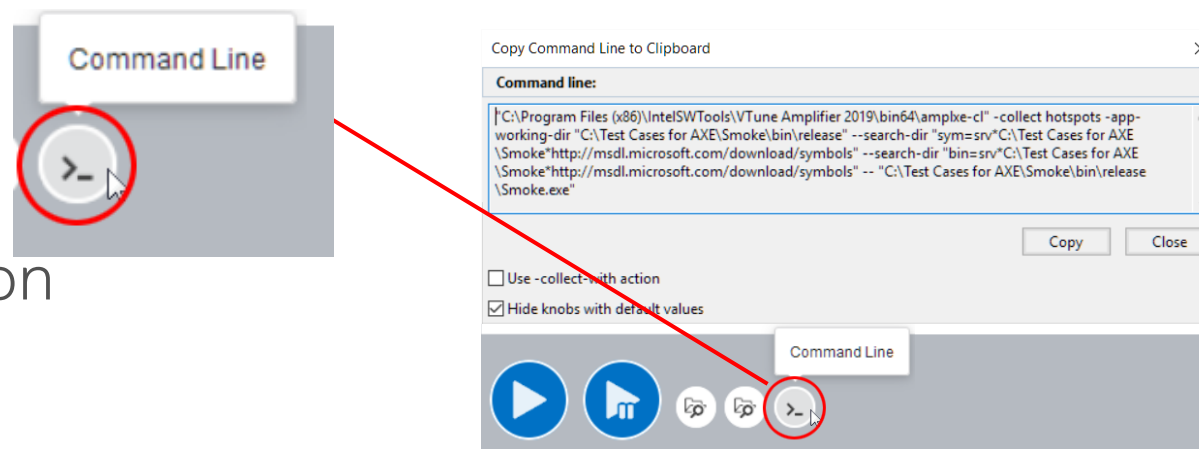
Automate analysis

- 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:

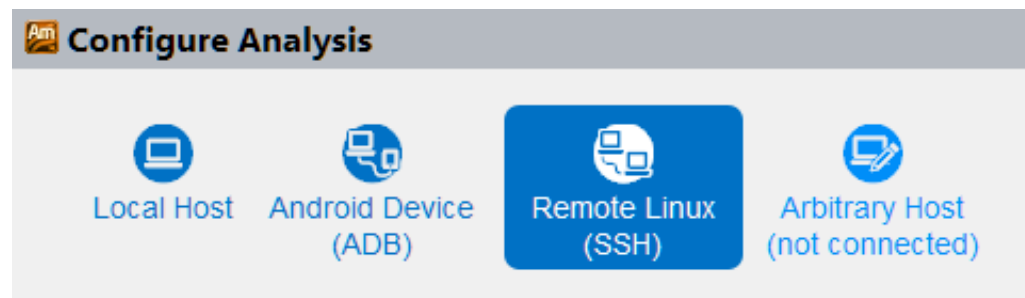
- /Applications/Intel VTune Profiler *<version>*.app

Interactive Remote Data Collection

Performance analysis of remote systems just got a lot easier

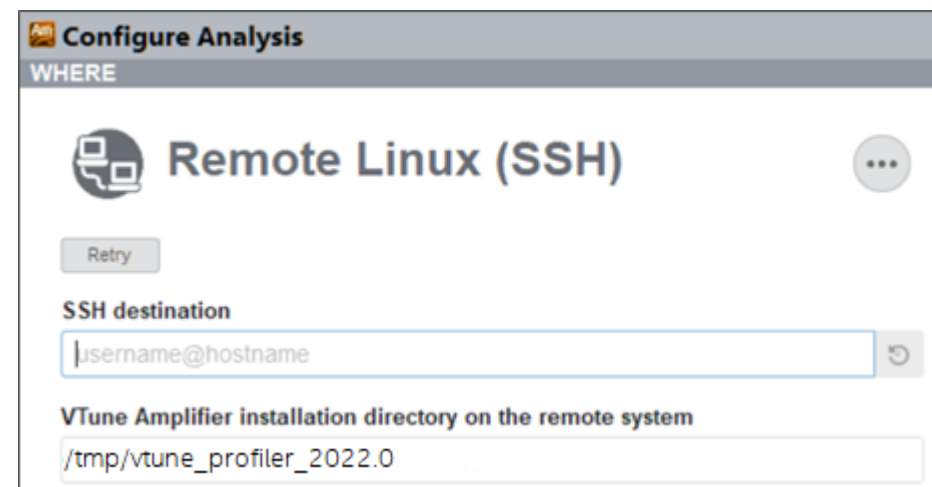
Interactive analysis

- 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

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

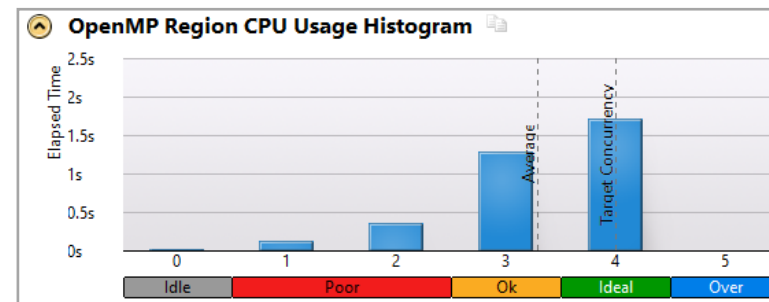
Quickly Find Tuning Opportunities

| Function / Call Stack | CPU Time | | | | | Spin Time | Overhead Time |
|--|-------------------------------|------|----|-------|------|-----------|---------------|
| | Effective Time by Utilization | | | | | | |
| | Idle | Poor | Ok | Ideal | Over | | |
| FireObject::checkCollision | 4.507s | | | | | 0s | 0s |
| FireObject::ProcessFireCollisionsRange | 3.444s | | | | | 0s | 0s |
| NtWaitForSingleObject | 0s | | | | | 3.406s | 0s |
| std::basic_ifstream<char,struct std::char_traits | 3.359s | | | | | 0s | 0s |
| Ogre::FileSystemArchive::open | 3.359s | | | | | 0s | 0s |
| CBaseDevice::Present | 2.335s | | | | | 0.671s | 0s |
| Selected 1 row(s): | | | | | | 1.151s | 0.728s |

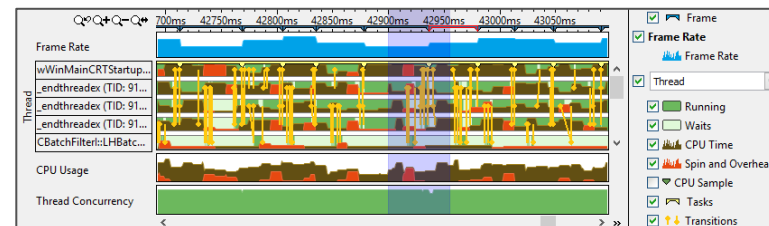
See Results On The Source Code

| Source | | CPU Time: Total by Utilization | |
|--------|---|--------------------------------|------|
| | | Idle | Poor |
| 81 | for (int i = 0; i < mem_array_i_max; i++) | 0.300s | |
| 82 | { | | |
| 83 | for (int j = 0; j < mem_array_j_max; j++) | 4.936s | |
| 84 | { | | |
| 85 | mem_array [j*mem_array_j_max+i] = *fill_val | 7.207s | |

Tune OpenMP Scalability



Visualize & Filter Data



Intel® VTune™ Profiler

GPU Profiling

Two GPU Analysis types

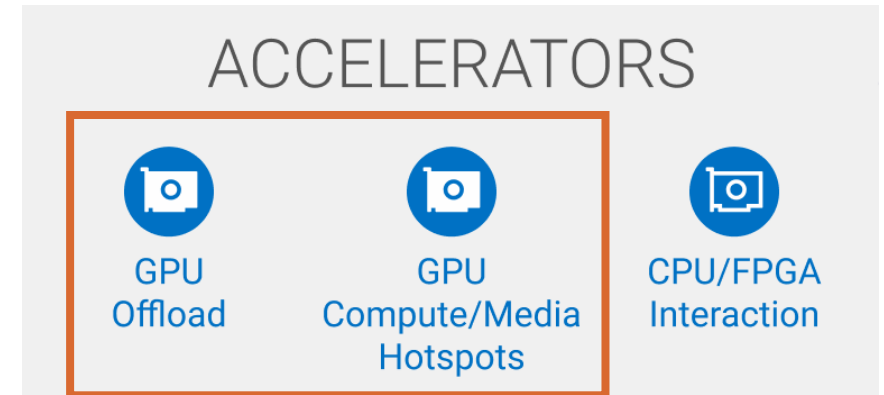
Intel® VTune™ Profiler

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

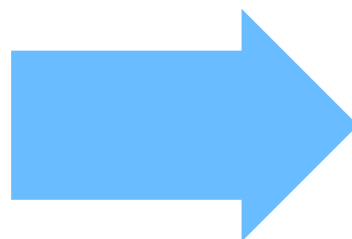
GPU Usage[?]: 0.6% 📉

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.

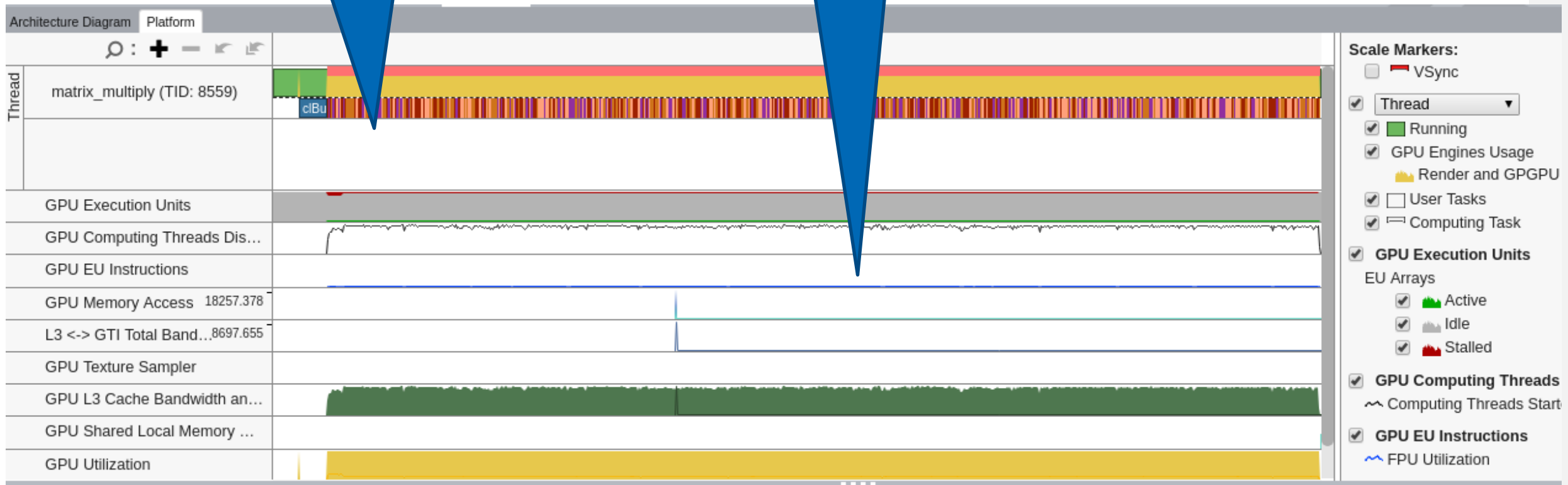
➤ Hottest GPU Computing Tasks with Low Occupancy

➤ Sampler Busy[?]: 40.6% of peak value

Timeline Correlates GPU and CPU Activity

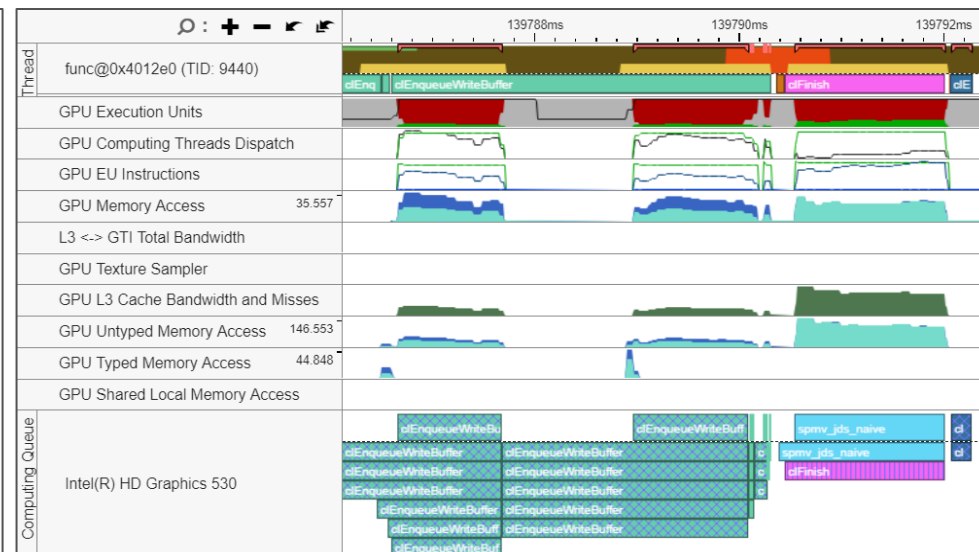
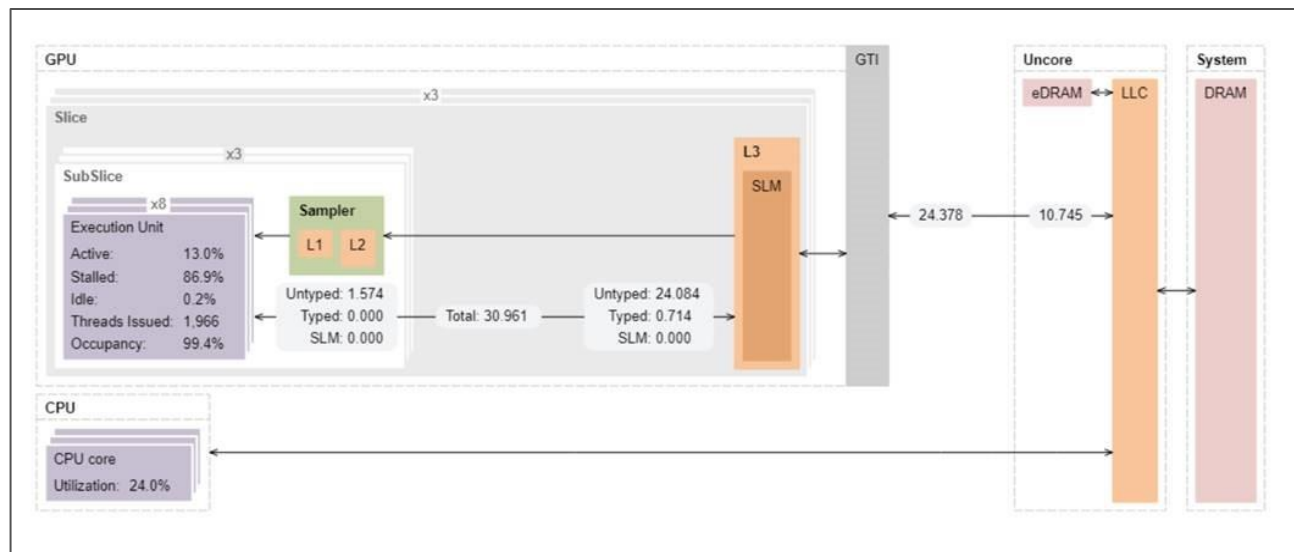
Identify too much or too little kernel activity

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

| Source | | Assembly | | = 🔥 🔥 🔥 🔥 | | | | | | | |
|----------|---|----------------------|--|-----------|--|--|--|--|--|--|--|
| Source ▲ | Source | Estimated GPU Cycles | | | | | | | | | |
| 256 | #ifdef USE_IMAGE_STORAGE | | | | | | | | | | |
| 257 | // Read the node information from the image | | | | | | | | | | |
| 258 | const ushort inx = (nodeData >> 16) * 7; | 0.2% | | | | | | | | | |
| 259 | const ushort iny = (nodeData & 0xffff); | | | | | | | | | | |
| 260 | const float4 bboxes_minX = as_float4(read_ | 0.8% | | | | | | | | | |
| 261 | const float4 bboxes_maxX = as_float4(read_ | 0.7% | | | | | | | | | |
| 262 | const float4 bboxes_minY = as_float4(read_ | 0.7% | | | | | | | | | |
| 263 | const float4 bboxes_maxY = as_float4(read_ | 0.7% | | | | | | | | | |
| 264 | const float4 bboxes_minZ = as_float4(read_ | 0.7% | | | | | | | | | |
| 265 | const float4 bboxes_maxZ = as_float4(read_ | 0.7% | | | | | | | | | |
| 266 | const int4 children = as_int4(read_imageui | 0.7% | | | | | | | | | |
| 267 | | | | | | | | | | | |
| 268 | const int4 visit = QBVHNode_BBoxIntersect(| 13.1% | | | | | | | | | |
| 269 | bboxes_minX, bboxes_maxX, | | | | | | | | | | |
| 270 | bboxes_minY, bboxes_maxY, | | | | | | | | | | |
| 271 | bboxes_minZ, bboxes_maxZ, | | | | | | | | | | |

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

Native language support is not currently available for Windows*

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 |

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 ^② |
|--|---------------|---------------|---------------|-----------------------------|
| 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 | Stores | LLC Miss Count ▼ | |
|--|-----------------|-------------------------|--------------------------|
| | | Local DRAM Access Count | Remote DRAM Access Count |
| ▼ doTriad\$omp\$parallel_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

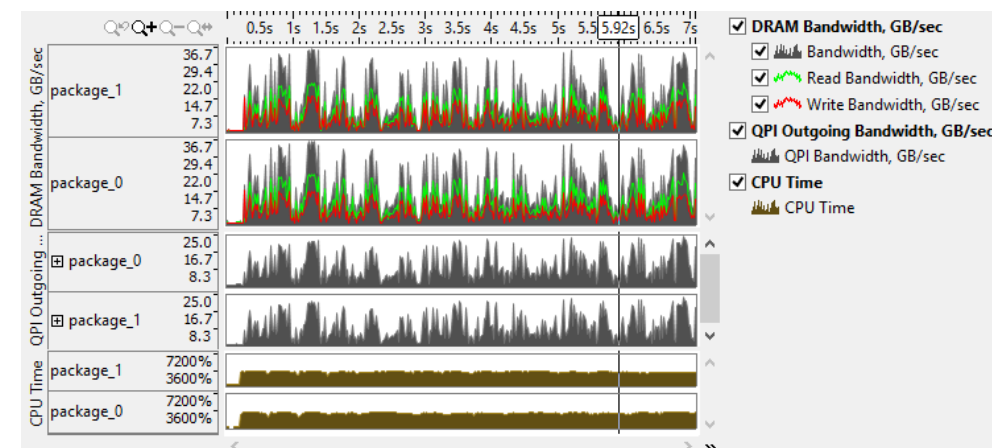
- Attribute cache misses to data structures

Better Bandwidth Analysis for Non-Uniform Memory

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

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 |



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

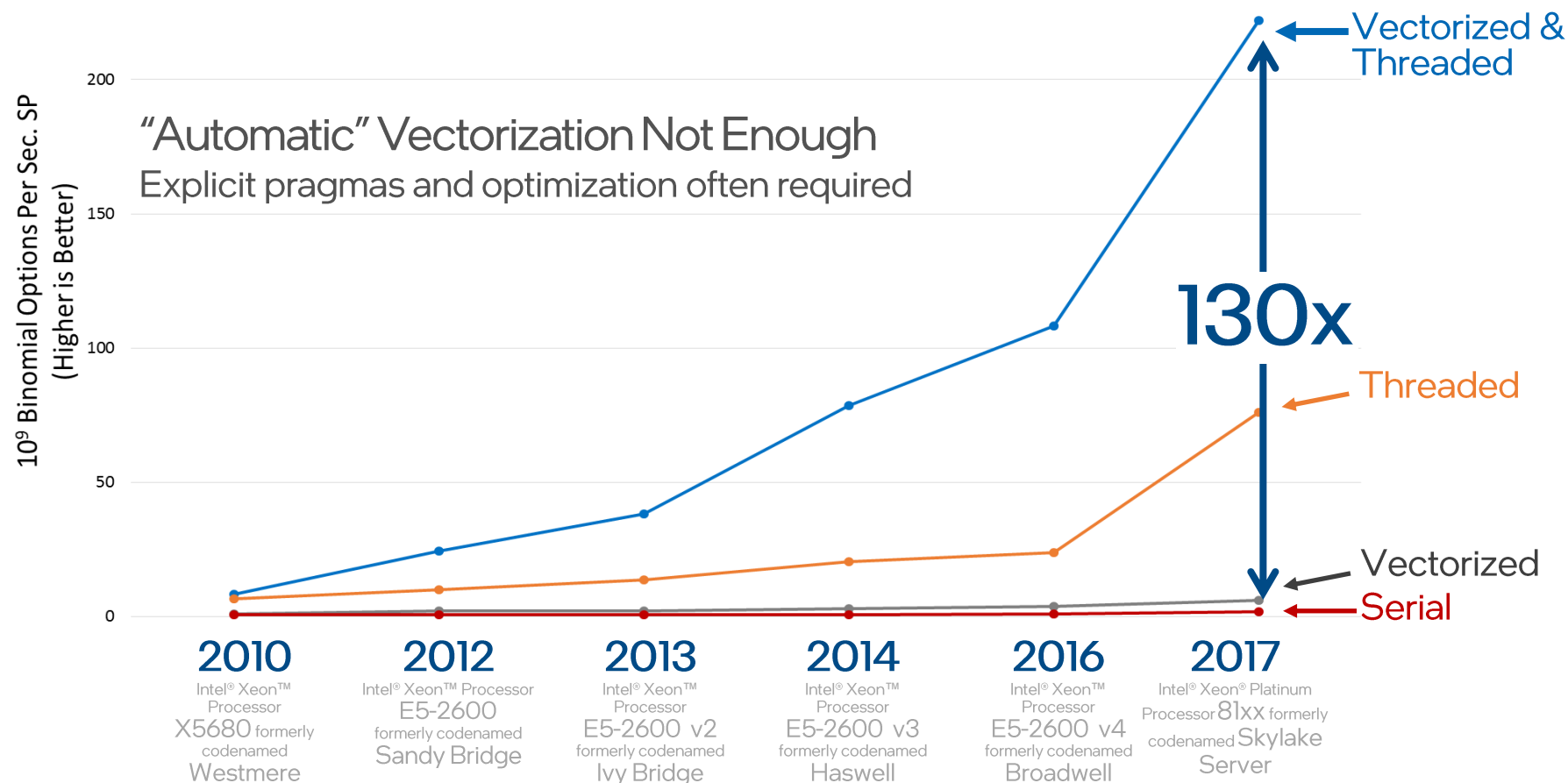
Intel® VTune™ Profiler Demo

Design your code for high-performance with
Intel[®] Advisor



Intel® Advisor: Vectorize & Thread or Performance Dies

Threaded + Vectorized Can Be Much Faster than Either One Alone



The Difference is Growing with Each New Generation of Hardware

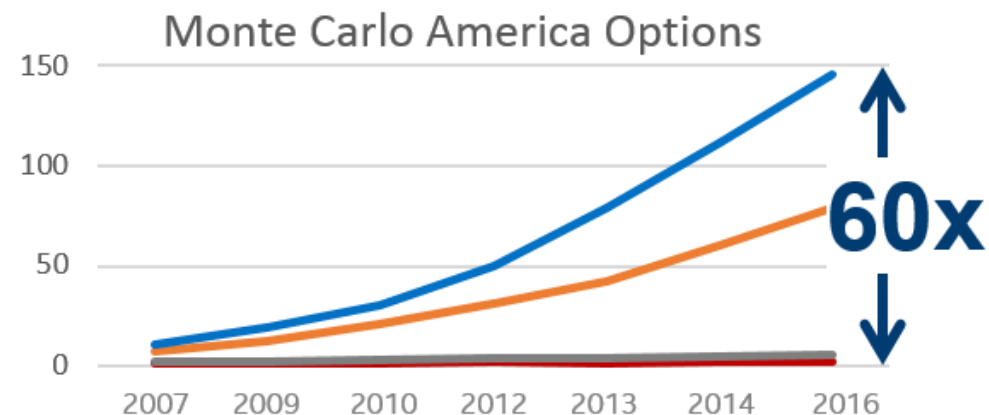
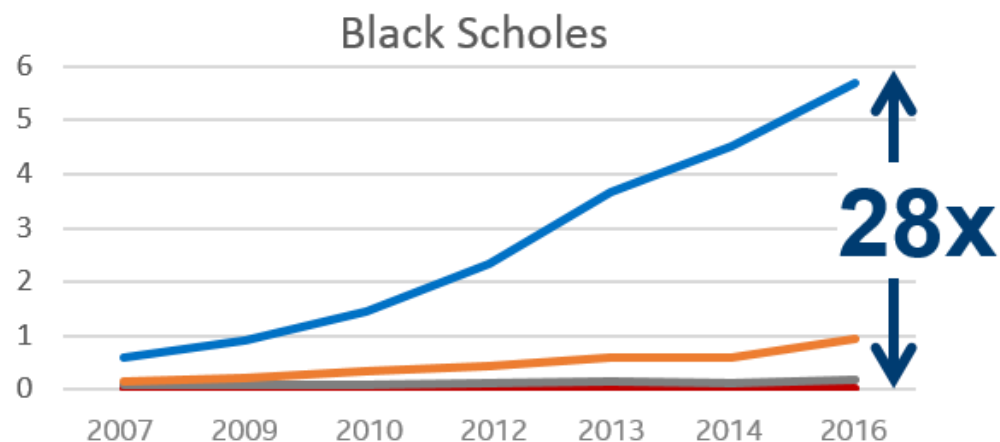
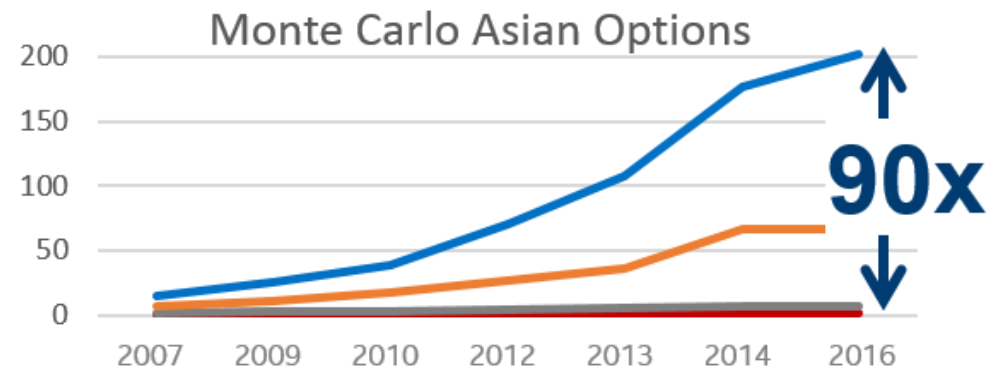
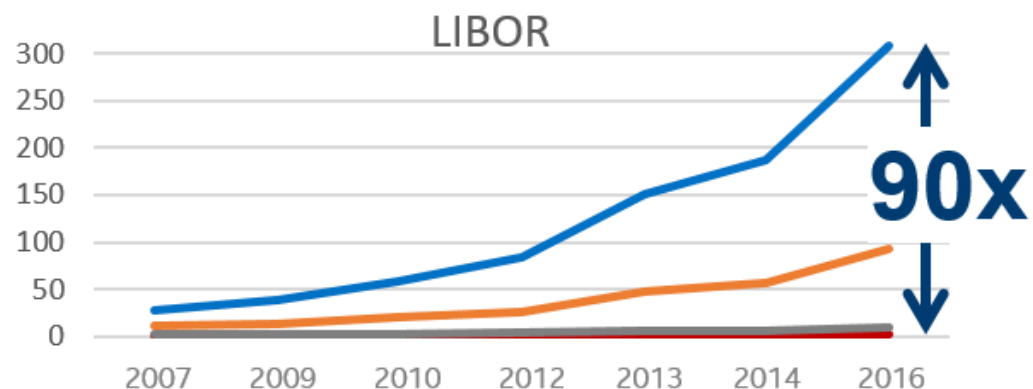
Testing Date: Performance results are based on testing by Intel employees as of 2017 and may not reflect all publicly available security updates.

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.

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.

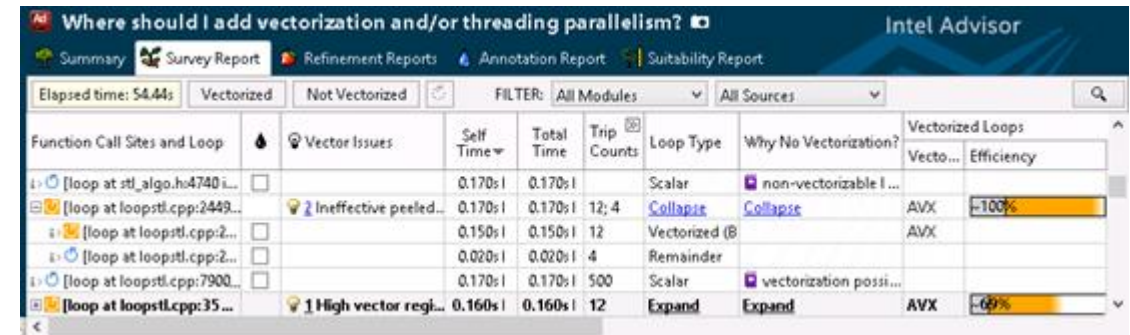
Arrays of structures are great for intuitively organizing data, but are much less efficient than structures of arrays. Use the [Intel® SIMD Data Layout Templates](#) (Intel® SDLT) to map data into a more efficient layout for vectorization.

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

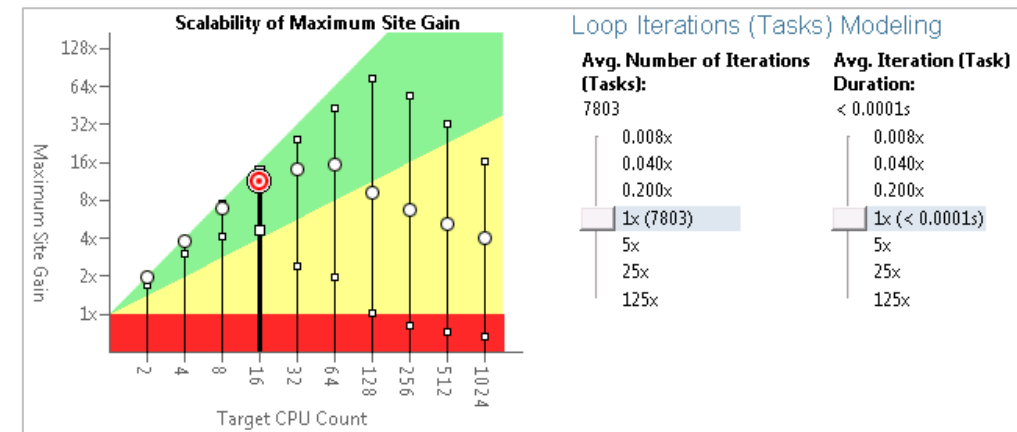


Intel Advisor interface showing a table of function call sites and loop vectorization status. The table includes columns for Function Call Sites and Loop, Vector Issues, Self Time, Total Time, Trip Counts, Loop Type, Why No Vectorization?, and Vectorized Loops. The table shows several loops, some of which are vectorized (e.g., AVX, AVX) and others that are not (e.g., non-vectorizable, vectorization possible).

| Function Call Sites and Loop | Vector Issues | Self Time | Total Time | Trip Counts | Loop Type | Why No Vectorization? | Vectorized Loops |
|------------------------------|-----------------------|-----------|------------|-------------|---------------|------------------------|------------------|
| [loop at stl_algo.h:4740] | | 0.170s | 0.170s | | Scalar | non-vectorizable l... | |
| [loop at loopstl.cpp:2449] | Ineffective peeled... | 0.170s | 0.170s | 12; 4 | Collapse | Collapse | AVX ~100% |
| [loop at loopstl.cpp:2... | | 0.150s | 0.150s | 12 | Vectorized (B | | AVX |
| [loop at loopstl.cpp:2... | | 0.020s | 0.020s | 4 | Remainder | | |
| [loop at loopstl.cpp:7900] | | 0.170s | 0.170s | 500 | Scalar | vectorization possi... | |
| [loop at loopstl.cpp:35... | High vector regi... | 0.160s | 0.160s | 12 | Expand | Expand | AVX ~69% |

■ 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*

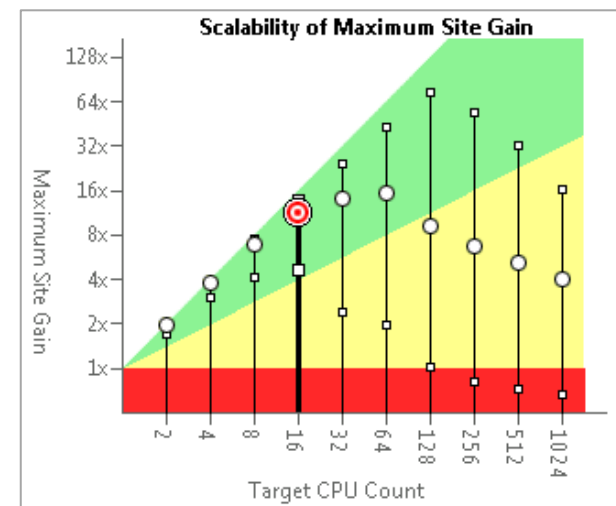
<http://intel.ly/advisor-xe>

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**



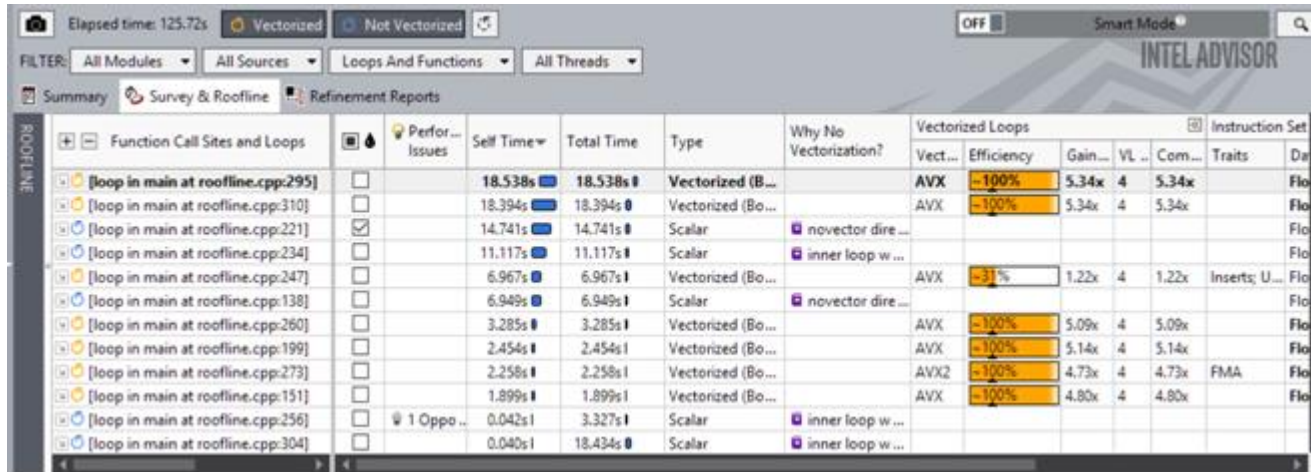
“Intel® Advisor has allowed us to quickly prototype ideas for parallelism, saving developer time and effort”

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?



The screenshot shows the Intel Advisor Vectorization Advisor interface. The top bar indicates 'Elapsed time: 125.72s' and 'Vectorized' status. The 'FILTER' section shows 'All Modules', 'All Sources', 'Loops And Functions', and 'All Threads'. The 'Summary' tab is selected, showing a list of function call sites and loops. The table columns include 'Function Call Sites and Loops', 'Perfor... Issues', 'Self Time', 'Total Time', 'Type', 'Why No Vectorization?', 'Vectorized Loops', and 'Instruction Set'. The table lists various loops and their vectorization status, with some loops being vectorized (e.g., AVX, AVX2) and others not being vectorized (e.g., Scalar, inner loop w...).

| Function Call Sites and Loops | Perfor... Issues | Self Time | Total Time | Type | Why No Vectorization? | Vectorized Loops | Instruction Set |
|------------------------------------|------------------|-----------|------------|-------------------|-----------------------|------------------|-----------------|
| [loop in main at roofline.cpp:295] | | 18.538s | 18.538s | Vectorized (B... | | AVX ~100% | 5.34x 4 5.34x |
| [loop in main at roofline.cpp:310] | | 18.394s | 18.394s | Vectorized (Bo... | | AVX ~100% | 5.34x 4 5.34x |
| [loop in main at roofline.cpp:221] | | 14.741s | 14.741s | Scalar | no vector dire... | | |
| [loop in main at roofline.cpp:234] | | 11.117s | 11.117s | Scalar | inner loop w... | | |
| [loop in main at roofline.cpp:247] | | 6.967s | 6.967s | Vectorized (Bo... | | AVX ~31% | 1.22x 4 1.22x |
| [loop in main at roofline.cpp:138] | | 6.949s | 6.949s | Scalar | no vector dire... | | |
| [loop in main at roofline.cpp:260] | | 3.285s | 3.285s | Vectorized (Bo... | | AVX ~100% | 5.09x 4 5.09x |
| [loop in main at roofline.cpp:199] | | 2.454s | 2.454s | Vectorized (Bo... | | AVX ~100% | 5.14x 4 5.14x |
| [loop in main at roofline.cpp:273] | | 2.258s | 2.258s | Vectorized (Bo... | | AVX2 ~100% | 4.73x 4 4.73x |
| [loop in main at roofline.cpp:151] | | 1.899s | 1.899s | Vectorized (Bo... | | AVX ~100% | 4.80x 4 4.80x |
| [loop in main at roofline.cpp:256] | 1 Oppo... | 0.042s | 3.327s | Scalar | inner loop w... | | |
| [loop in main at roofline.cpp:304] | | 0.040s | 18.434s | Scalar | inner loop w... | | |

"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

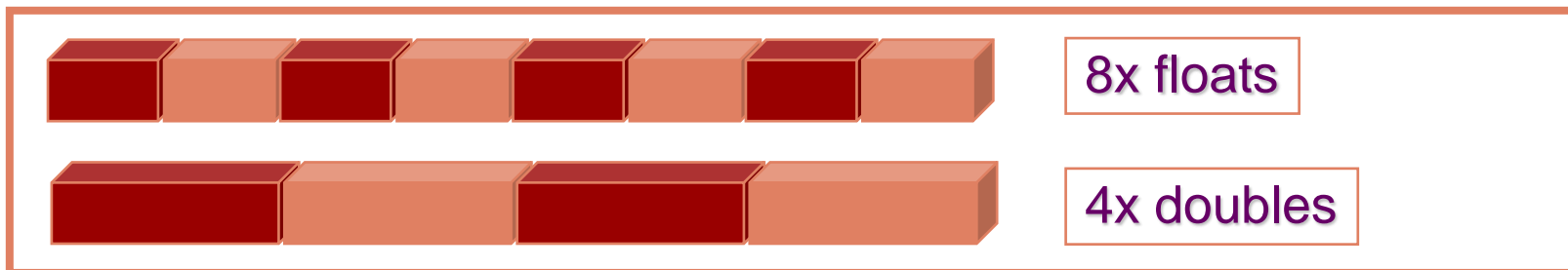
Adding 2
vectors

| | | | | | | | | | | | |
|---|------|------|-----|------|------|-----|-----|-----|------|-----|------|
| + | 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 |

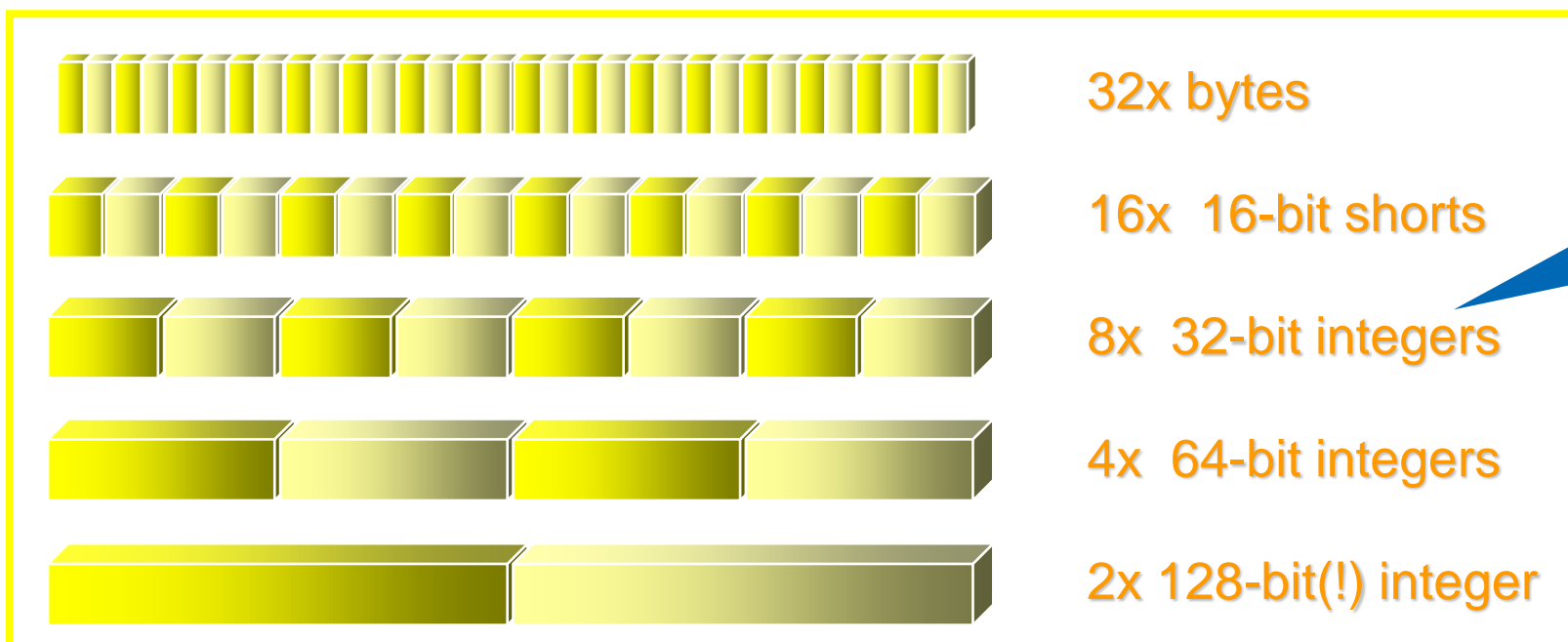
- These instructions are also referred to as Single Instruction Multiple Data (SIMD instructions)

Intel® Advanced Vector Extensions (Intel® AVX)

Intel®
AVX



Intel®
AVX2



Vector length –
the number of
elements that
can be processed

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?

| 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 std_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 std_numeric.h:247 in std ...] | 1 Assumed dependency ... | 0.150s | 0.150s | 49 | Scalar | vector dependence preve... | |

Get Faster Code Faster!

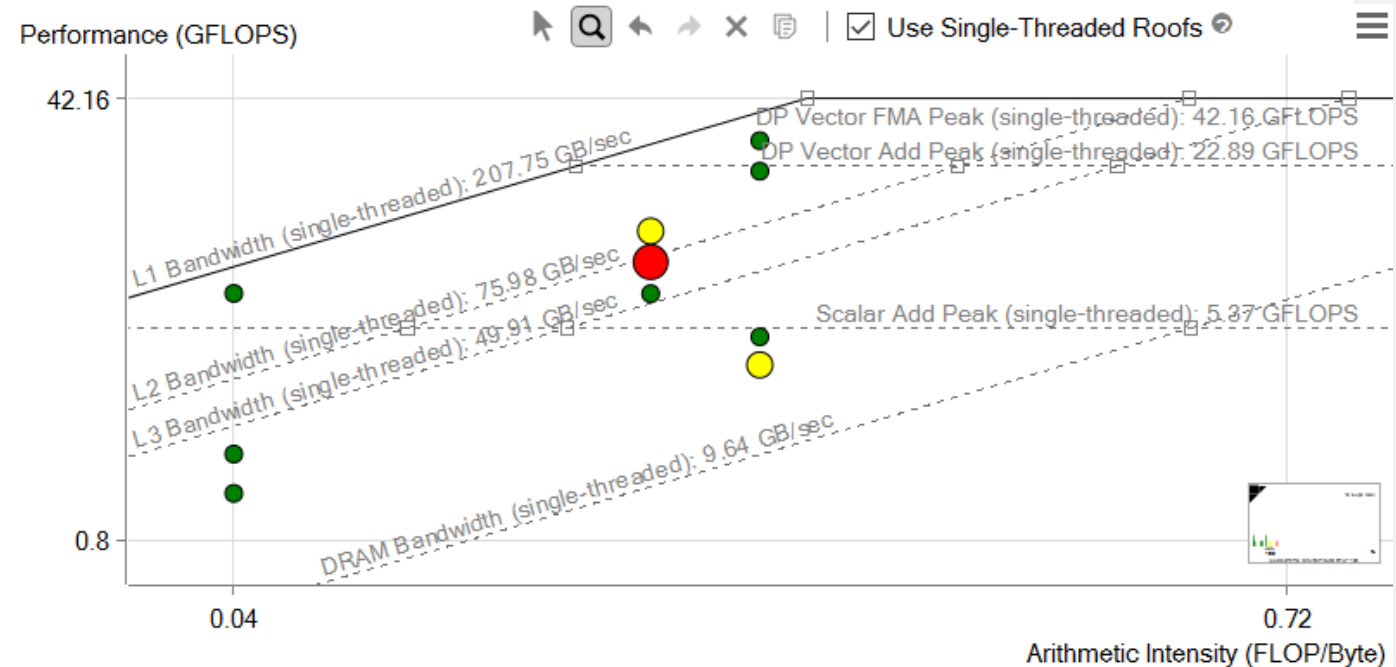
Intel® Advisor

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?



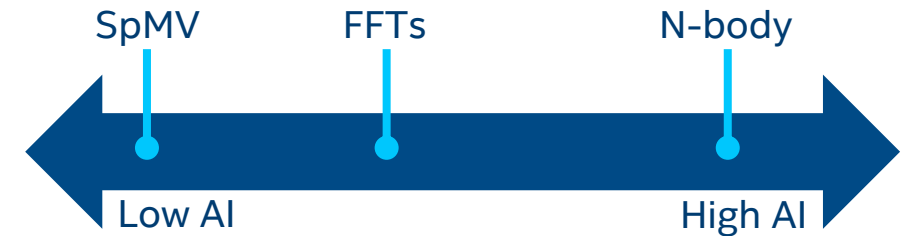
Roofline first proposed by University of California at Berkeley: [Roofline: An Insightful Visual Performance Model for Multicore Architectures](#), 2009

Cache-aware variant proposed by University of Lisbon: [Cache-Aware Roofline Model: Upgrading the Loft](#), 2013

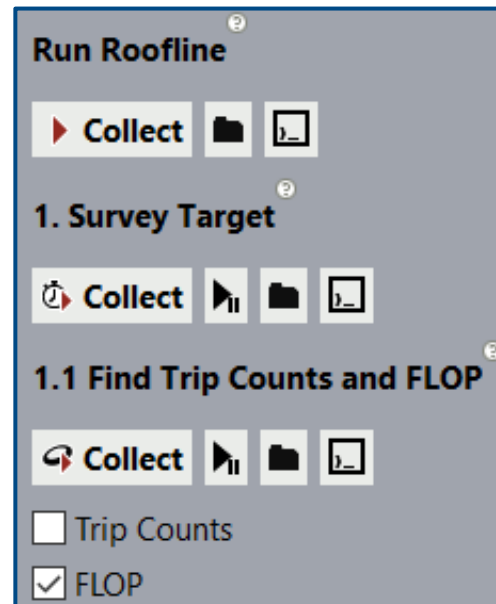
Roofline Metrics

- 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

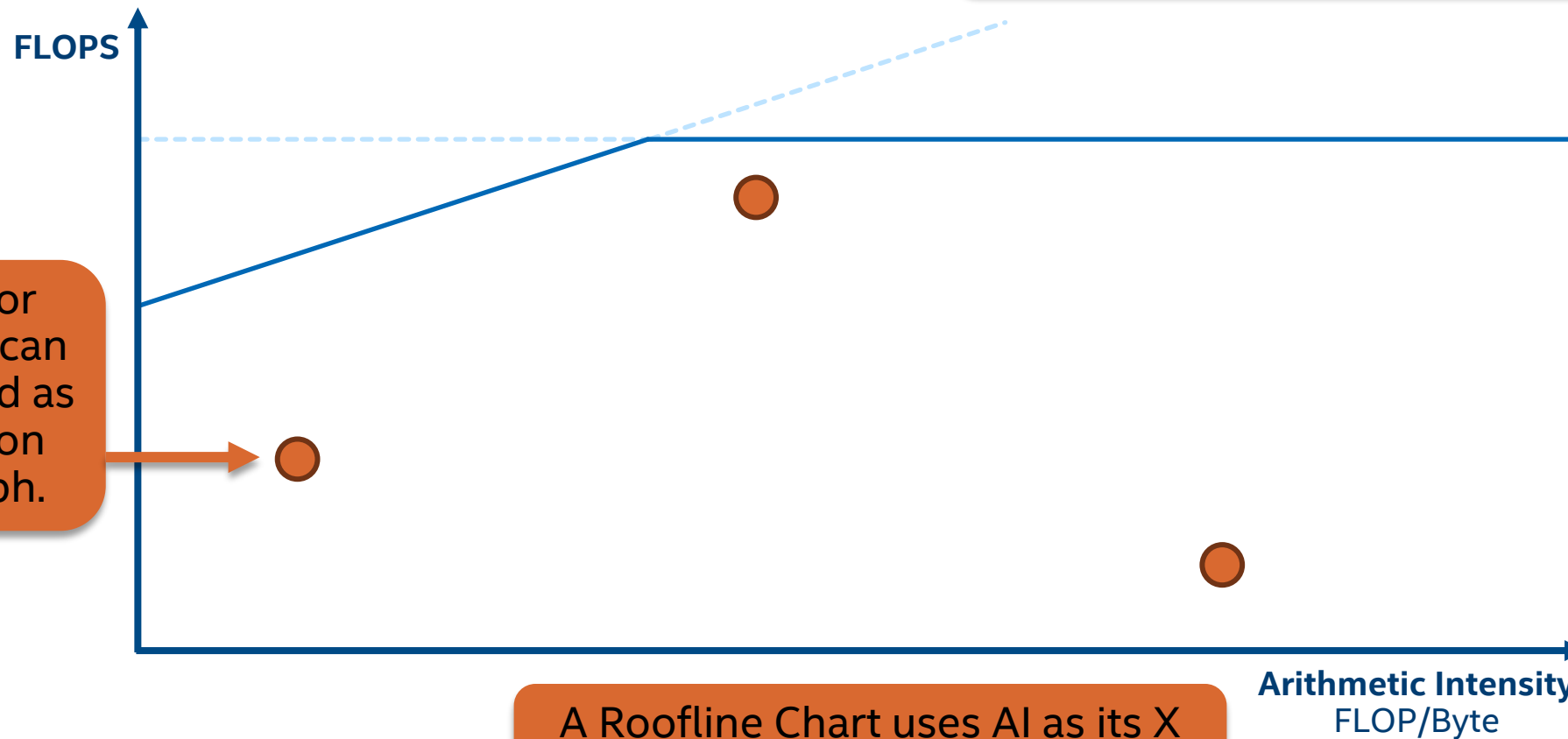
- Traffic measured from one level of memory (usually DRAM)
- AI may change with data set size
- AI changes as a result of memory optimizations

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



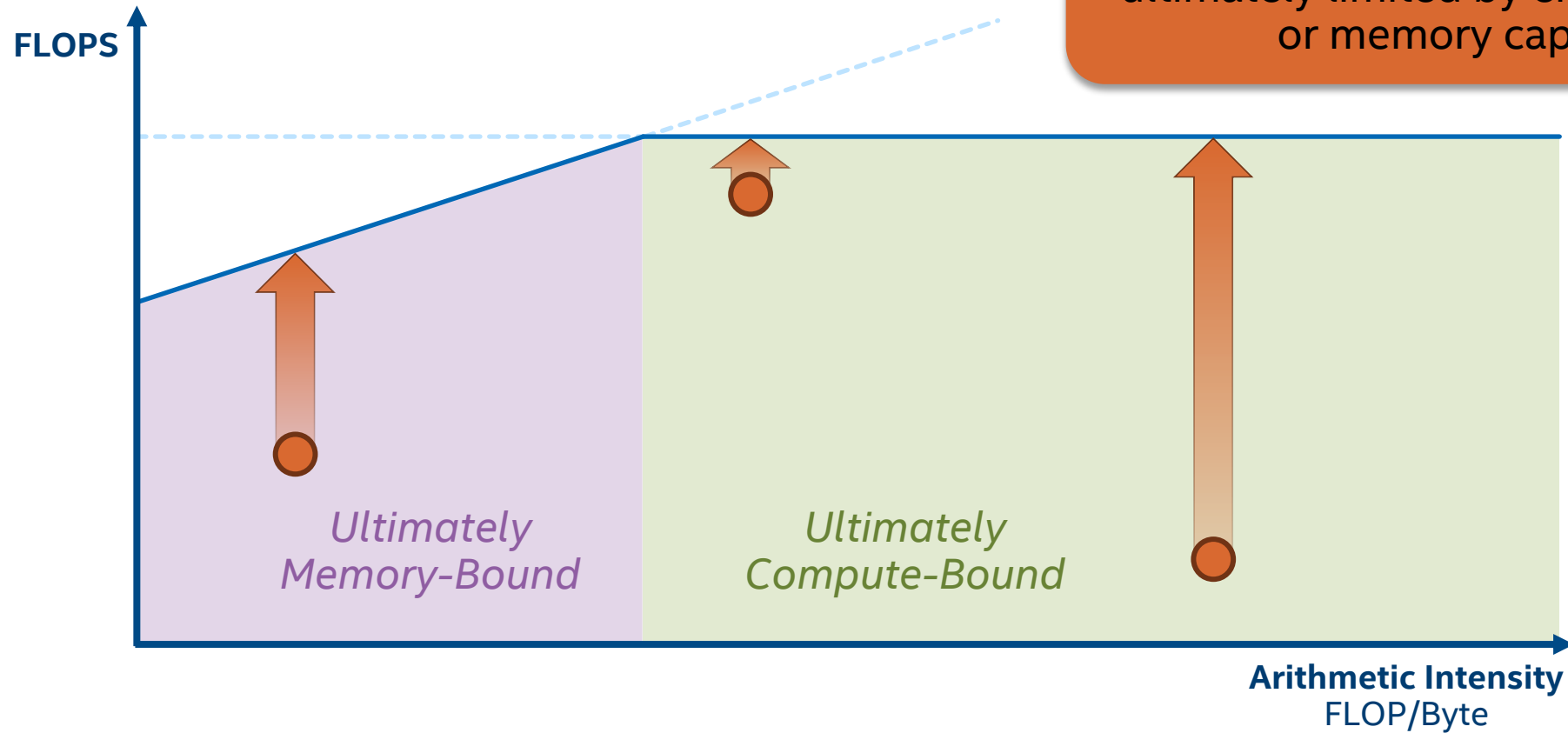
The maximum FLOPS as a product of ops/byte (AI) and maximum bytes supplied per second is a diagonal line.

The CPU's maximum FLOPS can be plotted as a horizontal line.

A loop or function can be plotted as a point on the graph.

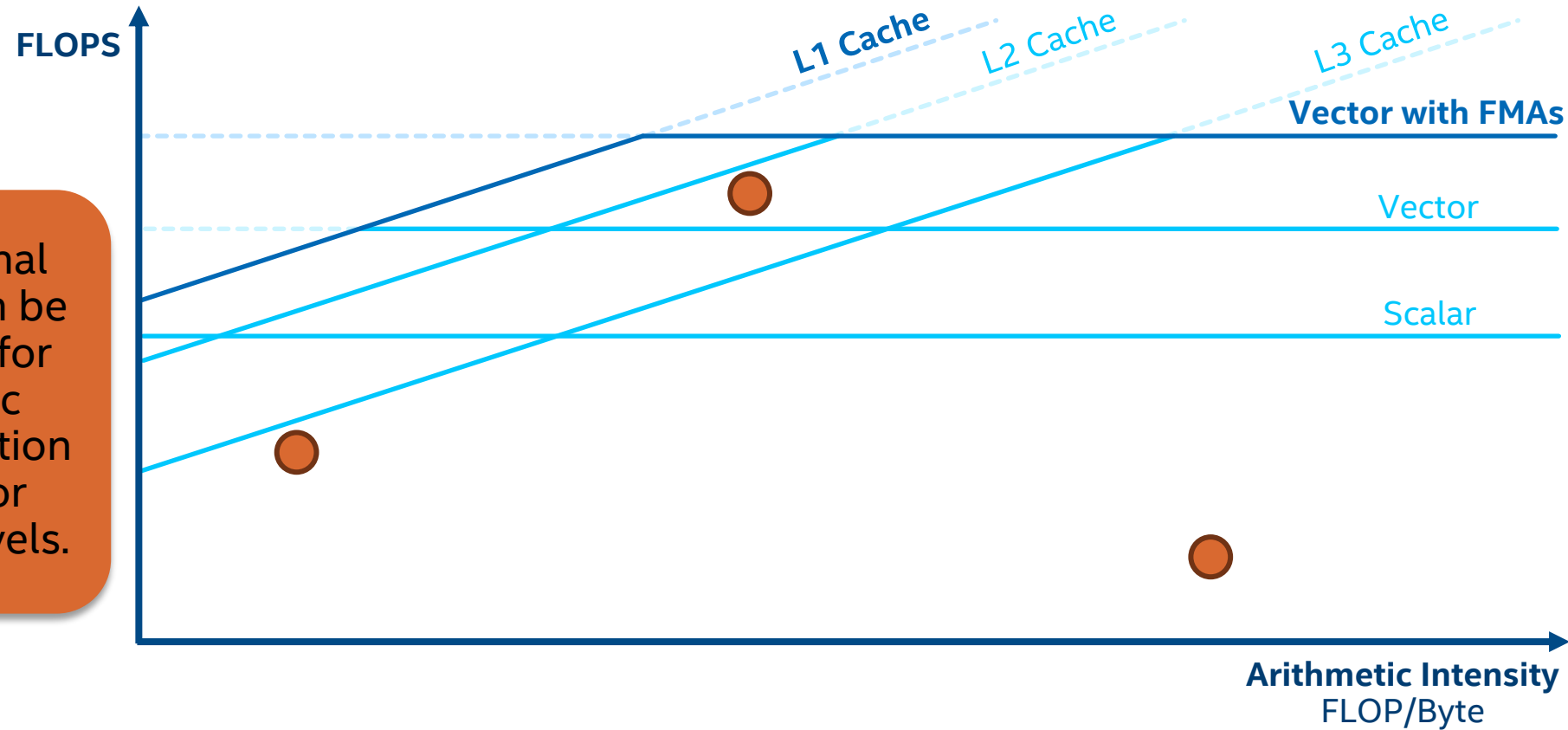
A Roofline Chart uses AI as its X axis and FLOPS as its Y axis.

Ultimate Performance Limits



Sub-Roofs and Current Limits

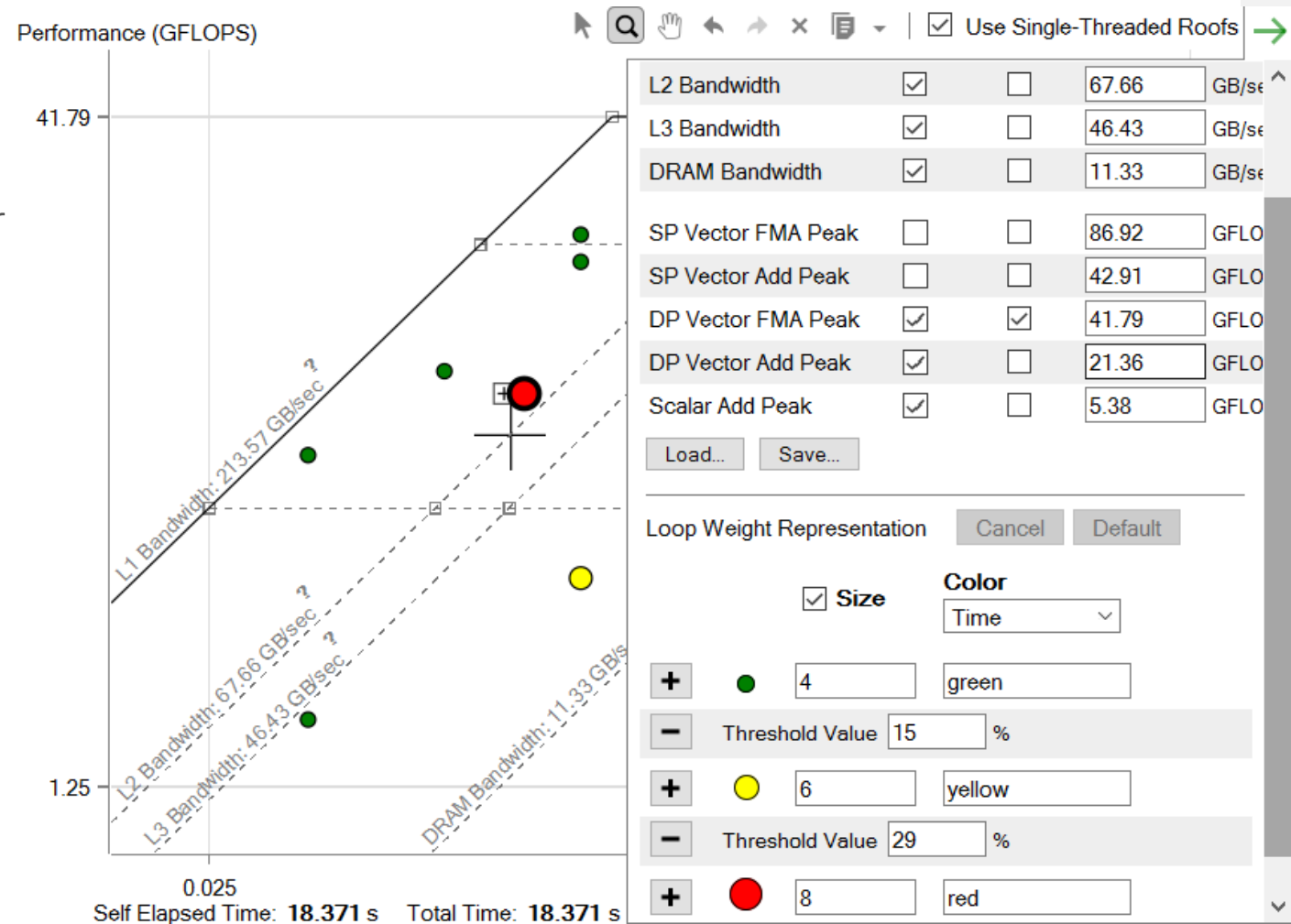
Additional roofs can be plotted for specific computation types or cache levels.



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

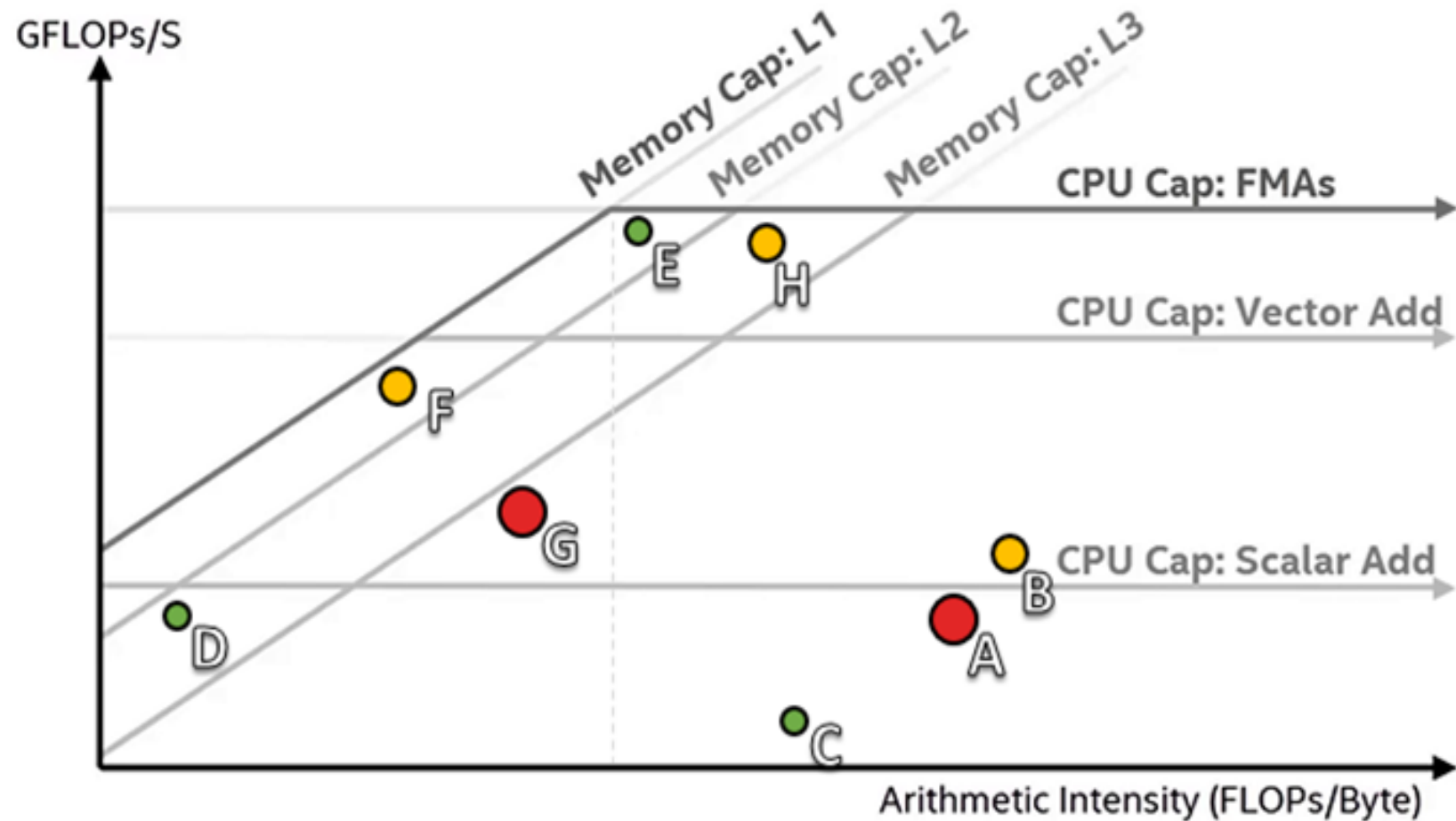
The Intel® Advisor Roofline Interface

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



Identifying Good Optimization Candidates

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



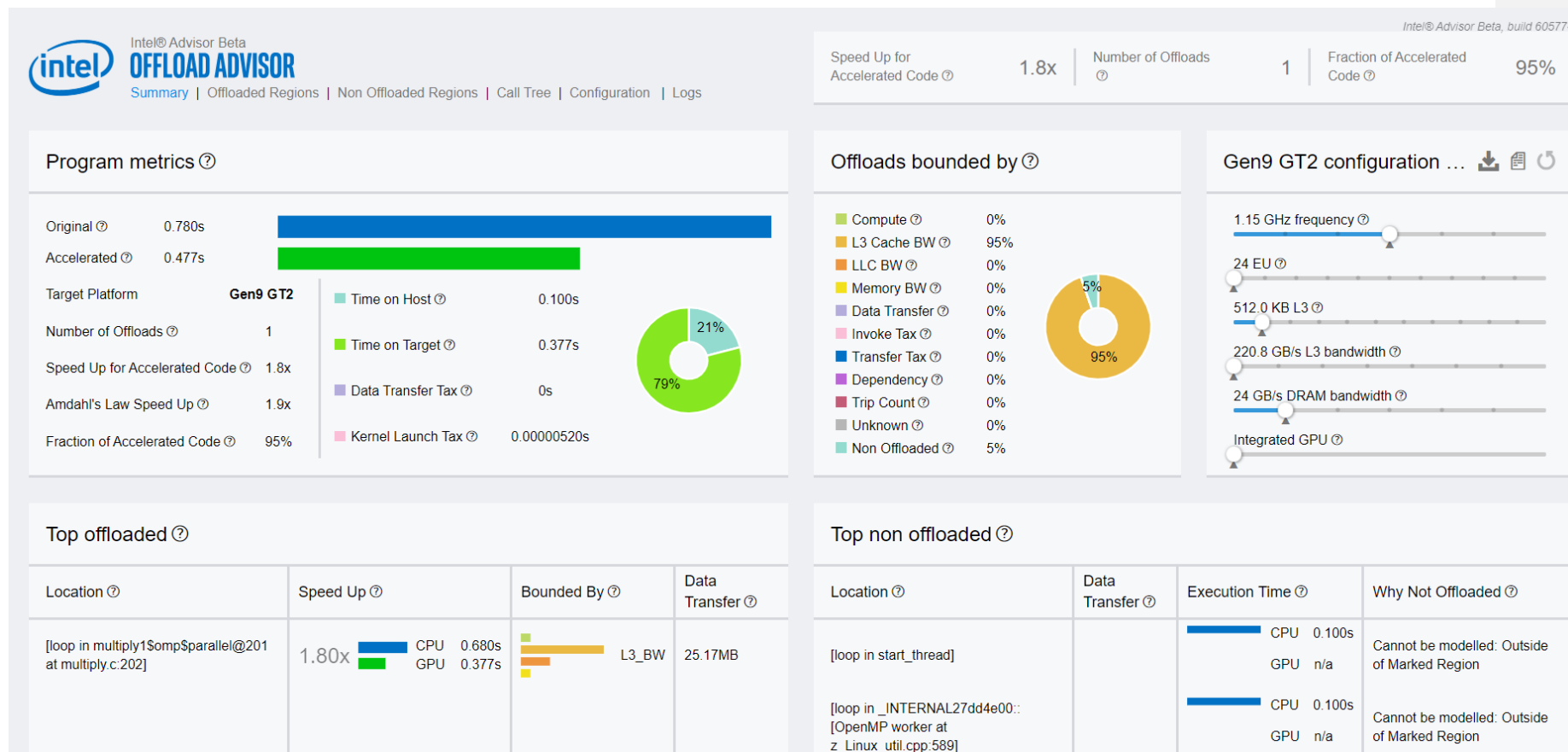
Intel® Advisor

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 ⓘ

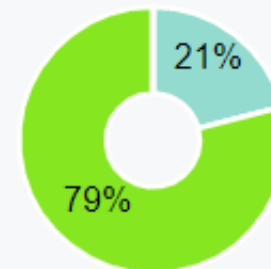
0.377s

Data Transfer Tax ⓘ

0s

Kernel Launch Tax ⓘ

0.00000520s



How to Run Intel® Advisor – Offload Advisor

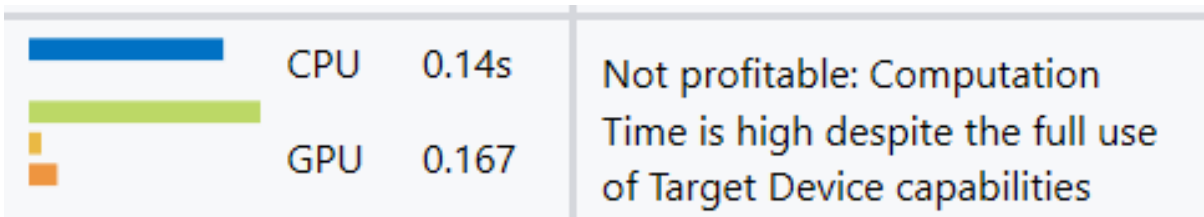
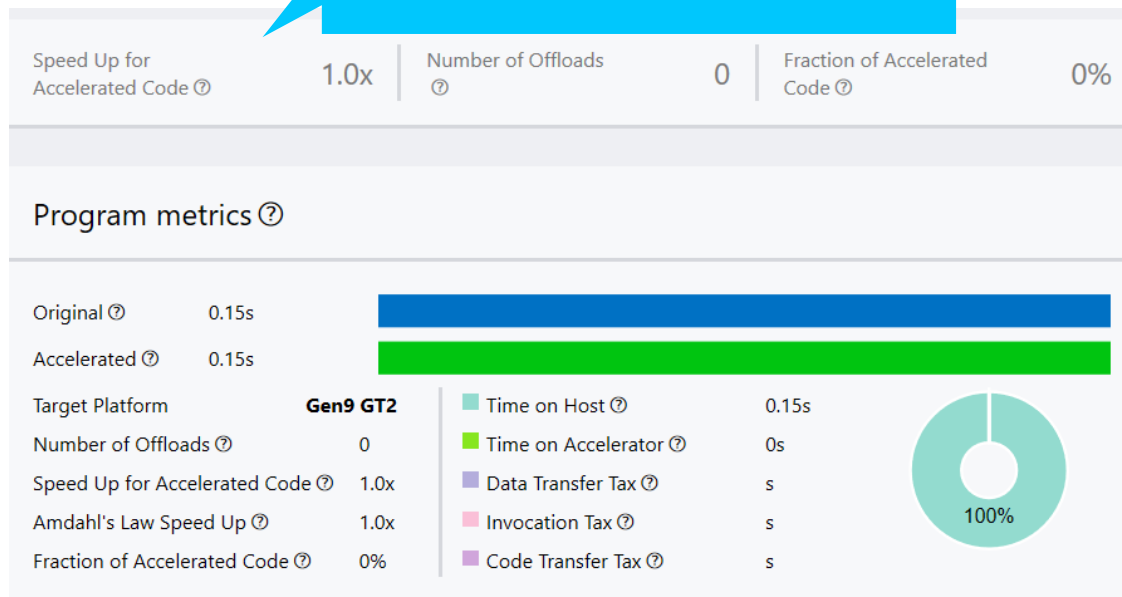
- `source <advisor_install_dir>/advixe-vars.sh`
- `advixe-python $APM/collect.py advisor_project --config gen9 --
/home/test/matrix`
- `advixe-python $APM/analyze.py advisor_project --config gen9 --out-dir
/home/test/analyze`
- View the report.html generated (or generate a command-line report)



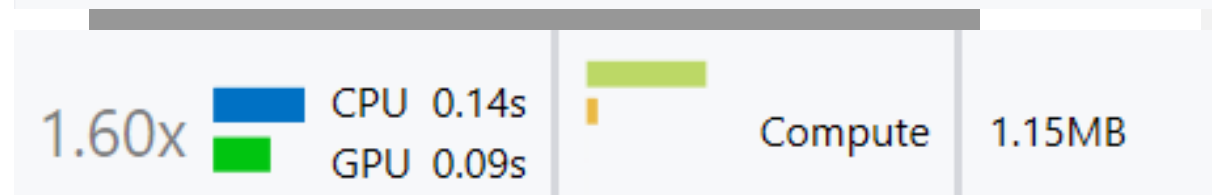
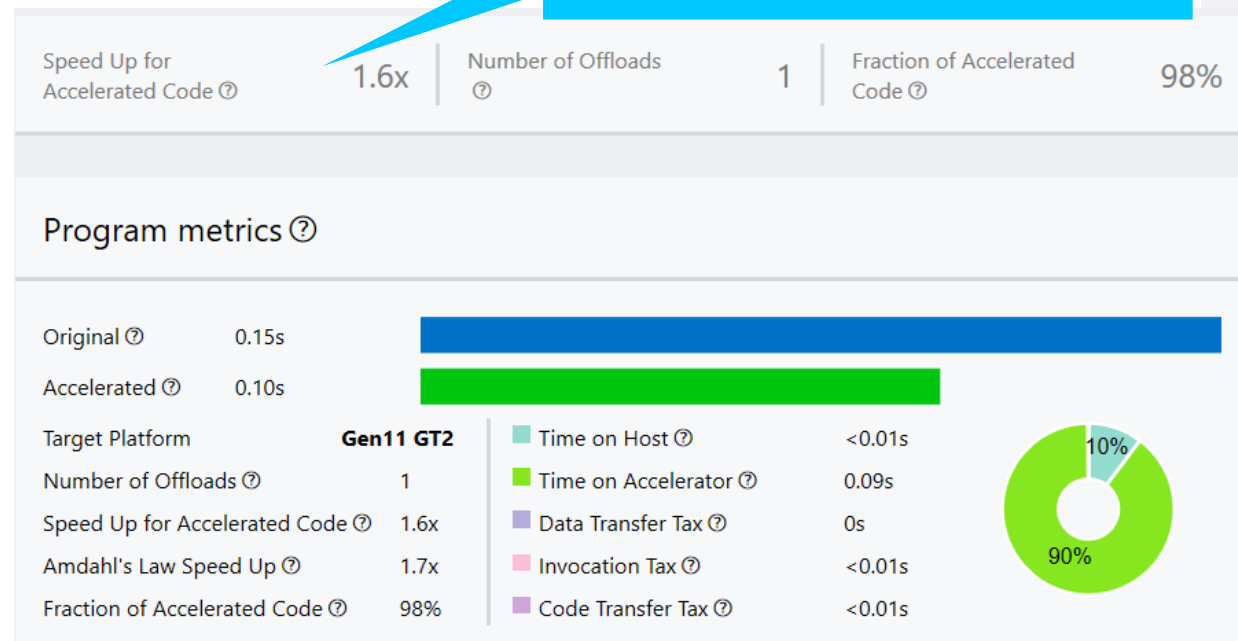
Analyze for a specific
GPU config

Compare Acceleration on Different GPUs

Gen9 – Not profitable to offload kernel



Gen11 – 1.6x speedup



Intel® Advisor Demo

Deliver reliable applications with
Intel[®] Inspector



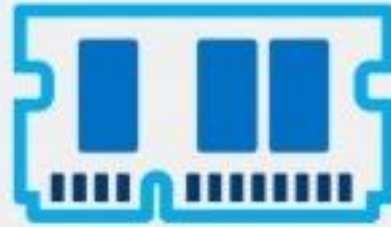
Threading, Memory and Persistence Debugger

Intel® Inspector



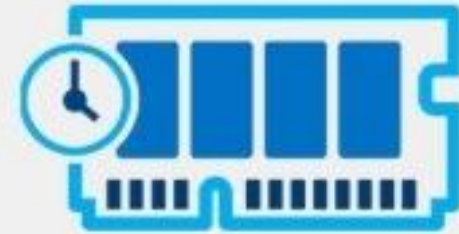
Threading Debugger

Debug hard-to-find data races and deadlocks.



Memory Debugger

Detect memory leaks, invalid accesses, and more.



Persistent Memory Debugger

Find persistence errors that include redundant cache flushes.

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 |

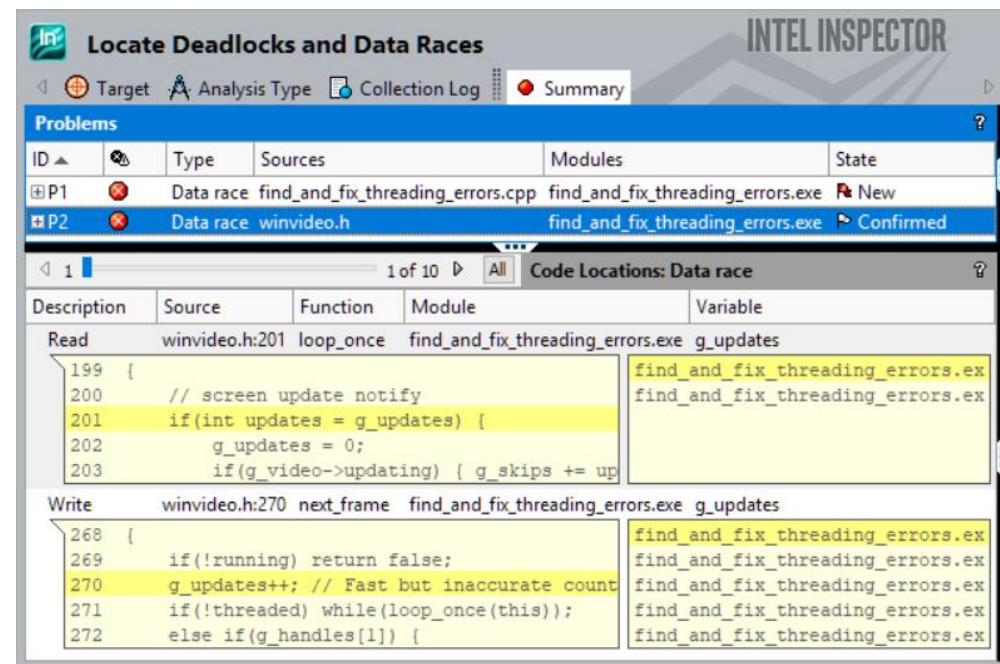
Incorrect Answer

| 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.

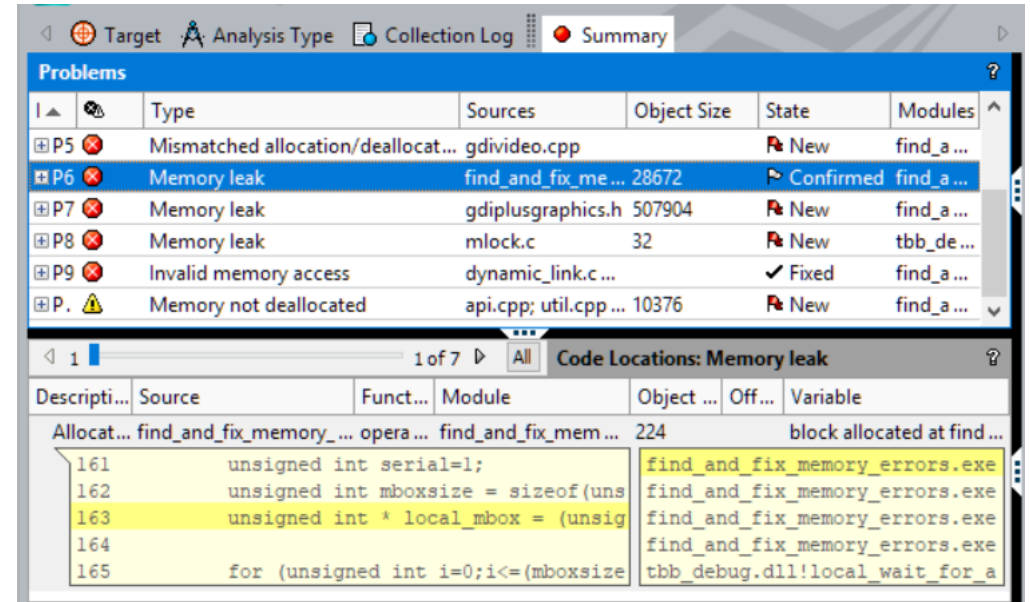


Debug Memory & Threading Errors

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 compiler¹
 - Analyzes dynamically generated or linked code
 - Inspects 3rd party libraries without source
 - Productive user interface + debugger integration
 - Command line for automated regression analysis

¹Compilers that follows common OS standards.



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

Fits your existing process

Productive User Interface Saves Time

Intel® Inspector

Select a
**problem
set**

**Code
snippets**
displayed
for
selected
problem

Detect Memory Problems

Target Analysis Type Collection Log Summary

| Problems | Type | Sources | State | M |
|----------|------------------------------------|-----------------------------|-----------|-----|
| P1 | Mismatched allocation/deallocation | find_and_fix_memory_... | Confirmed | |
| P2 | Memory leak | find_and_fix_memory_... | Deferred | fi. |
| P3 | Invalid memory access | find_and_fix_memory_... | New | fi. |
| P4 | Memory not deallocated | api.cpp; mlock.c; util.c... | New | fi. |
| | Memory not deallocated | video.cpp:82 | New | fi. |
| | Memory not deallocated | util.cpp:163 | New | fi. |
| | Memory not deallocated | api.cpp:218 | New | fi. |
| | Memory not deallocated | mlock.c:347 | New | tb. |

Filters

Severity

- Error 3 item(s)
- Warning 1 item(s)

Type

- Invalid memory access 1 item(s)
- Memory leak 1 item(s)
- Memory not deallocated 1 item(s)
- Mismatched allocation/deallocation 1 item(s)

Source

- api.cpp 1 item(s)
- find_and_fix_memory_errors.exe 2 item(s)

Code Locations: Mismatched allocation/deallocation

| Description | Source | Function | Module | Object Size | Offset |
|------------------------------|--|------------|--------------------------------|-------------|--------|
| Mismatched deallocation site | find_and_fix_memory_errors.cpp:175 | operator() | find_and_fix_memory_errors.exe | | |
| 173 | drawing->put_pixel(c); | | | | |
| 174 | } | | | | |
| 175 | free(drawing); //Memory Error: use delete instead of free | | | | |
| 176 | //delete drawing; | | | | |
| 177 | } | | | | |
| Allocation site | find_and_fix_memory_errors.cpp:170 | operator() | find_and_fix_memory_errors.exe | | |
| 168 | for (int y = r.begin(); y != r.end(); ++y) { | | | | |
| 169 | { | | | | |
| 170 | drawing_area * drawing = new drawing_area(startx, totaly-y, st | | | | |
| 171 | for (int x = startx ; x < stopx; x++) { | | | | |
| 172 | color_t c = render_one_pixel (x, y, local_mbox, serial, st | | | | |

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

Double Click for Source & Call Stack

Intel® Inspector

Source code locations displayed for selected problem

Mismatched allocation/deallocation

Target Analysis Type Collection Log Summary Sources

Mismatched deallocation site - Thread thread_video (4596) (find_and_fix_memory_errors.exe!operator() - find_and_fix_memory_errors.cp...

find_and_fix_memory_errors.cpp Disassembly (find_and_fix_memory_errors.exe!0x46d6) Call Stack

```
164
165     for (unsigned int i=0;i<=(mboxsize/(sizeof(unsigned int)));i++)
166         local_mbox[i]=0; //Memory Error: C declared arrays go from 0
167
168     for (int y = r.begin(); y != r.end(); ++y) {
169         {
170             drawing_area * drawing = new drawing_area(startx, totally
171             for (int x = startx ; x < stopx; x++) {
172                 color_t c = render_one_pixel (x, y, local_mbox, serie
173                 drawing->put_pixel(c);
174             }
175         }
176     }
```

Allocation site - Thread thread_video (4596) (find_and_fix_memory_errors.exe!operator() - find_and_fix_memory_errors.cpp:170)

find_and_fix_memory_errors.cpp Disassembly (find_and_fix_memory_errors.exe!0x4613) Call Stack

```
170     drawing_area * drawing = new drawing_area(startx, totally
171     for (int x = startx ; x < stopx; x++) {
172         color_t c = render_one_pixel (x, y, local_mbox, serie
173         drawing->put_pixel(c);
174     }
175     free(drawing); //Memory Error: use delete instead of free
176     //delete drawing;
```

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 <u>not</u> change) |
| Confirmed | Set by user (tool will <u>not</u> change) |
| Fixed | Set by user (tool <u>will</u> change) |
| Regression | Error detected with previous state of "Fixed" |

The screenshot shows the Intel Inspector 2017 interface. At the top, there's a header with the Intel logo and the text "Detect Memory Problems" and "INTEL INSPECTOR 2017". Below this is a navigation bar with tabs: "Target", "Analysis Type", "Collection Log", and "Summary". The "Summary" tab is active, showing a list of problems under the heading "Problems". The list has columns for ID, Type, Sources, State, and Modules. Four problems are listed: P1 (Mismatched allocation/deallocation, Confirmed), P2 (Memory leak, Deferred), P3 (Invalid memory access, New), and P4 (Memory not deallocated, New). A yellow arrow points from the "New" state in the table above to the "New" state of problem P3 in the list. To the right of the list, a context menu is open, showing options: "View Source", "Edit Source", "Copy to Clipboard", "Explain Problem", "Create Problem Report..", "Debug This Problem", "Change State", and "Merge States...". The "Change State" option is selected, and a sub-menu is open showing the following states: "Not fixed", "Confirmed", "Fixed", "Not a problem", and "Deferred". A mouse cursor is hovering over the "Confirmed" option.

| ID | Type | Sources | State | Modules |
|----|------------------------------------|-----------------------------------|-----------|-------------|
| P1 | Mismatched allocation/deallocation | find_and_fix_memory_errors... | Confirmed | find_and... |
| P2 | Memory leak | find_and_fix_memory_errors... | Deferred | find_and... |
| P3 | Invalid memory access | find_and_fix_memory_errors... | New | find_and... |
| P4 | Memory not deallocated | api.cpp; mlock.c; util.cpp; vi... | New | find_and... |

- View Source
- Edit Source
- Copy to Clipboard
- Explain Problem
- Create Problem Report..
- Debug This Problem
- Change State
 - Not fixed
 - Confirmed
 - Fixed
 - Not a problem
 - Deferred
- Merge States...

Filtering - Focus on What's Important

Example: See only the errors in one source file

Before – All Errors

Problems

| ID | Type | Sources | State |
|----|---------------------|-------------|-----------|
| P1 | Mismatched alloc... | find_an ... | New |
| P2 | Mismatched alloc... | api.cpp | New |
| P3 | Memory leak | api.cpp | Confirmed |
| P4 | Mismatched alloc... | video.c ... | Not fixed |
| P5 | Mismatched alloc... | video.c ... | Not fixed |
| P6 | Mismatched alloc... | video.c ... | Not fixed |

Filters

Severity: Error (55 item(s)), Warning (1 item(s))

Type: Invalid memory access (41 item(s)), Memory leak (1 item(s)), Memory not deallocated (11 item(s)), Mismatched allocation/dealloc... (2 item(s))

Source: api.cpp (21 item(s)), find_and_fix_memory_errors.cpp (3 item(s)), util.cpp (10 item(s)), video.cpp (21 item(s))

(1) Filter – Show only one source file

After – Only errors from one source file

Problems

| ID | Type | Sources | State |
|----|---------------|-------------|-----------|
| P1 | Mismatche... | find_an ... | New |
| P2 | Memory leak | find_an ... | Confirmed |
| P3 | Invalid me... | find_an ... | Deferred |

Filters

Severity: Error (3 item(s))

Type: Invalid memory access (1 item(s)), Memory leak (1 item(s)), Mismatched allocation/dealloc... (1 item(s))

Source: find_and_fix_memory_errors.cpp (3 item(s))

(2) Error count drops

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

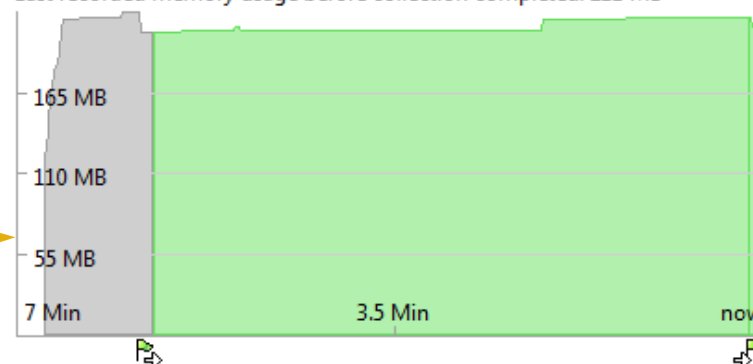
Intel® Inspector

As your app is running...

Memory usage graph
plots memory growth

Memory Used by Analysis Tool and Target Application

Last recorded memory usage before collection completed: 211 MB



Select a cause of
memory growth

| Problems | | | | | |
|----------|---------------|------------------------------------|--------------------------------|-------------|-----------|
| ID | Type | Sources | Modules | Object Size | State |
| | Memory growth | gdiplus.dll:0x47240 | gdiplus.dll | 40960 | New |
| | Memory growth | find_and_fix_memory_errors.cpp:163 | find_and_fix_memory_errors.exe | 90108 | Not fixed |
| | Memory growth | find_and_fix_memory_errors.cpp:163 | find_and_fix_memory_errors.exe | 1802160 | Not fixed |
| | Memory growth | find_and_fix_memory_errors.cpp:163 | find_and_fix_memory_errors.exe | 30036 | Not fixed |
| | Memory growth | find_and_fix_memory_errors.cpp:163 | find_and_fix_memory_errors.exe | 1621944 | Not fixed |
| | Memory growth | find_and_fix_memory_errors.cpp:170 | find_and_fix_memory_errors.exe | 40 | Not fixed |

| Code Locations: Memory growth | | | | | |
|-------------------------------|--|------------|--------------------------------|-------------|--------|
| Description | Source | Function | Module | Object Size | Offset |
| Allocation site | find_and_fix_memory_errors.cpp:163 | operator() | find_and_fix_memory_errors.exe | 90108 | |
| | 161 unsigned int serial=1; | | find_and_fix_memory_errors.exe | | |
| | 162 unsigned int mboxsize = sizeof(unsigned int)*(max_objectid() + | | find_and_fix_memory_errors.exe | | |
| | 163 unsigned int * local_mbox = (unsigned int *) malloc(mboxsize); | | find_and_fix_memory_errors.exe | | |
| | 164 | | find_and_fix_memory_errors.exe | | |
| | 165 for (unsigned int i=0;i<=(mboxsize/(sizeof(unsigned int)));i++ | | tbb_debug.dll!local_wait_for_a | | |

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

| Problems | | |
|----------|--------------------------------|---------|
| ID ▲ | Type | Sources |
| P1 | Mismatched allocation/deall... | |
| P2 | Memory leak | |
| P3 | Invalid memory access | |
| | Invalid memory access | |
| P4 | Memory growth | |
| P5 | Memory growth | |
| P6 | Memory growth | |

View Source
Edit Source
Copy to Clipboard
Explain Problem
Create Problem Report...
Debug This Problem

Threading Errors

| Problems | | | |
|----------|-----------|----------------|---------|
| ID ▲ | Type | Sources | Modules |
| P1 | Data race | winvideo.h | |
| | Data race | winvideo.h:270 | |
| | Data race | winvideo.h:270 | |
| | Data race | winvideo.h:201 | |
| | Data race | winvideo.h:202 | |
| | Data race | winvideo.h:202 | |

View Source
Edit Source
Copy to Clipboard
Explain Problem
Create Problem Report...
Debug This Problem

- 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

