Code that Outperforms

Intel® oneAPI Analyzers

Intel VTune Profiler and Intel Advisor

Xiao Zhu
xiao.zhu@intel.com

intel®

Agenda

1 **Intel® oneAPI Overview**
Introduction to the Intel oneAPI Base and HPC Toolkits

2 **Intel® VTune™ Profiler**

3 **Intel® Advisor**

4 **GPU Profiling Demo**
Demo profiling the iso3dfd sample on Intel DevCloud with Intel® Advisor and Intel® VTune™ Profiler

Multiarchitecture Programming for Accelerated Compute, Freedom of Choice for Hardware

oneAPI Initiative & Intel® oneAPI Tools

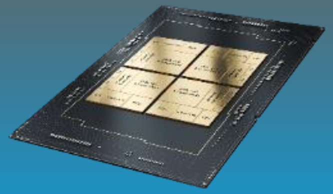
intel®

Modern Applications Demand Diverse Architectures

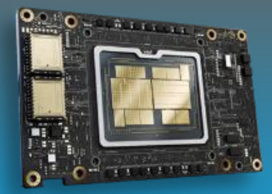
Diverse accelerators needed to meet today's performance requirements:

48% of developers target heterogeneous systems that use more than one kind of processor or core[1]

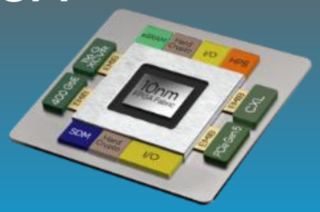
CPU



GPU



FPGA



Other Accelerators

Developer Challenges: Multiple Architectures, Vendors, and Programming Models

oneAPI™

Open, Standards-based, Multiarchitecture Programming

1. *Evans Data Global Development Survey Report 22.1, June 2022*

oneAPI Industry Initiative

Break the Chains of Proprietary Lock-in

Freedom to Make Your Best Choice

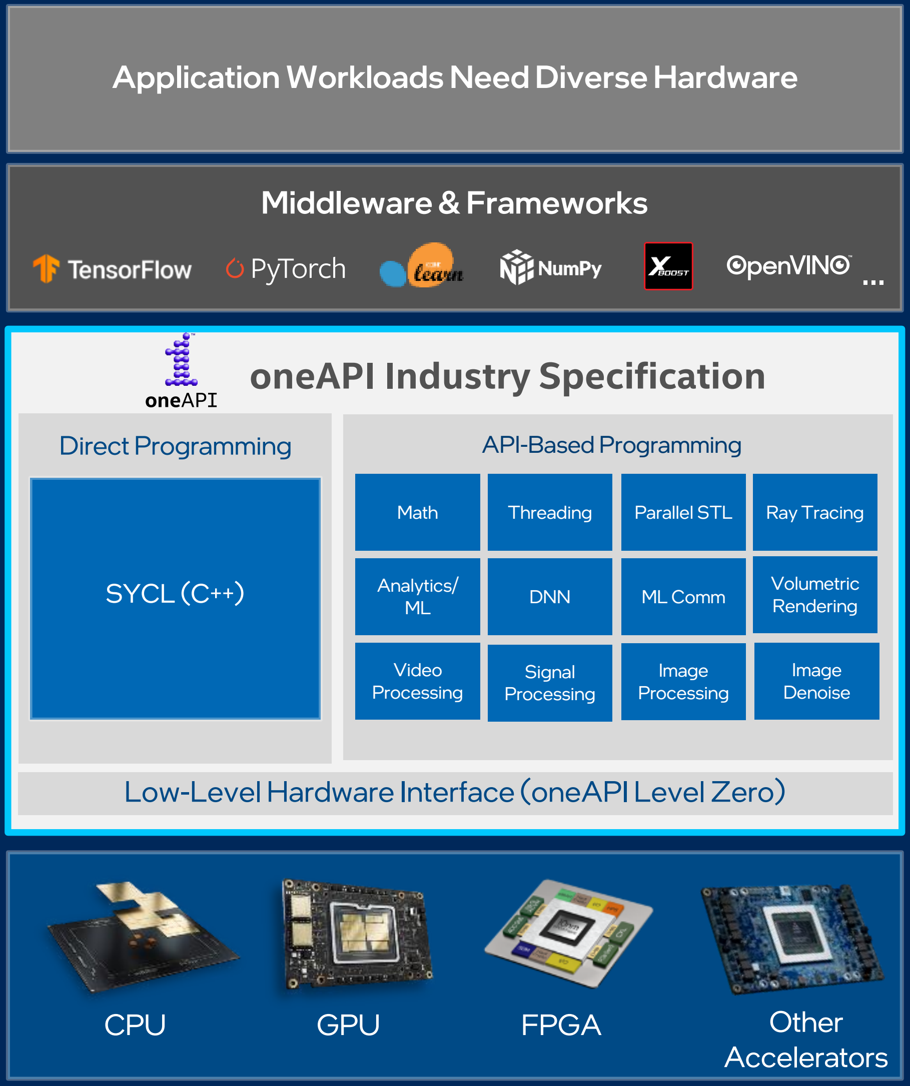
- C++ programming model for multiple architectures and vendors
- Cross-architecture code reuse for freedom from vendor lock-in

Realize all the Hardware Value

- Performance across CPU, GPUs, FPGAs, and other accelerators
- Expose and exploit cutting-edge features of the latest hardware

Develop & Deploy Software with Peace of Mind

- Open industry standards provide a safe, clear path to the future
- Interoperable with familiar languages and programming models including Fortran, Python, OpenMP, and MPI
- Powerful libraries for acceleration of domain-specific functions



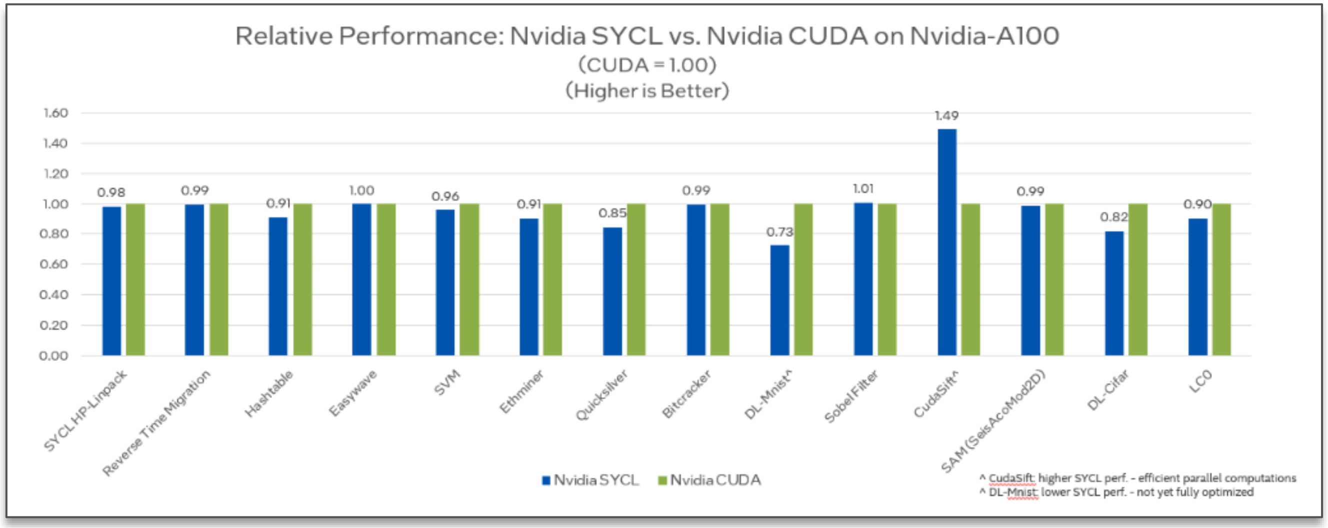
oneAPI

The productive, smart path to freedom for accelerated computing from the economic and technical burdens of proprietary programming models

Accelerating Choice with SYCL*

Khronos Group Standard

- Open, standards-based
- Multiarchitecture performance
- Freedom from vendor lock-in
- Comparable performance to native CUDA on Nvidia GPUs
- Extension of widely used C++ language
- Speed code migration via open source SYCLomatic or Intel® DPC++ Compatibility Tool

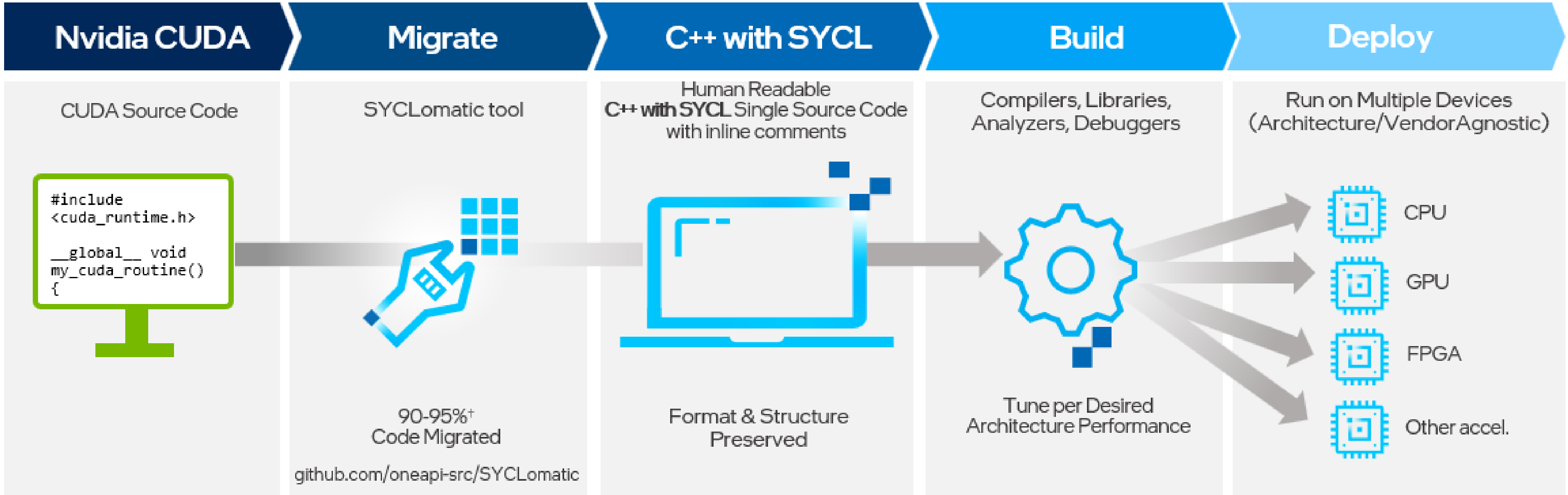


Architectures	Intel | Nvidia | AMD CPU/GPU | RISC-V | ARM Mali | PowerVR | Xilinx

Testing Date: Performance results are **based on testing by Intel as of April 15, 2023** and may not reflect all publicly available updates.
Configuration Details and Workload Setup: Intel® Xeon® Platinum 8360Y CPU @ 2.4GHz, 2 socket, Hyper Thread On, Turbo On, 256GB Hynix DDR4-3200, ucode 0xd000363. GPU: Nvidia A100 PCIe 80GB GPU memory. Software: SYCL open source/CLANG 17.0.0, CUDA SDK 12.0 with NVIDIA-NVCC 12.0.76, cuMath 12.0, cuDNN 12.0, Ubuntu 22.04.1. SYCL open source/CLANG compiler switches: -fsycl-targets=nvptx64-nvidia-cuda, NVIDIA NVCC compiler switches: -O3 –gencode arch=compute_80, code=sm_80. Represented workloads with Intel optimizations.
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.
SYCL is a trademark of the Khronos Group Inc.

SYCLomatic: CUDA* to SYCL* Migration Made Easy

Choose where to run your software, don't let the software choose for you.



Open source SYCLomatic tool assists developers migrating code written in CUDA to C++ with SYCL, generating **human readable** code wherever possible

~90-95% of code typically migrates automatically[1]

Inline comments are provided to help developers finish porting the application

Intel® DPC++ Compatibility Tool is Intel's implementation, available in the Intel® oneAPI Base Toolkit

github.com/oneapi-src/SYCLomatic

[1]Intel estimates as of March 2023. Based on measurements on a set of 85 HPC benchmarks and samples, with examples like Rodinia, SHOC, PENNANT. Results may vary.
*Other names and brands may be claimed as the property of others. SYCL is a trademark of the Khronos Group Inc.

intel.

Codeplay oneAPI Plug-ins for Nvidia* & AMD*

Support for Nvidia & AMD GPUs to Intel® oneAPI Base Toolkit

oneAPI for NVIDIA & AMD GPUs

- Free download of binary plugins to Intel® oneAPI DPC++/C++ Compiler:
- Nvidia GPU
- AMD beta GPU
- No need to build from source!
- Plug-ins updated quarterly in-sync with SYCL 2020 conformance & performance

Priority Support

- Available through Intel, Codeplay & our channel
- Requires Intel Priority Support for Intel® oneAPI DPC++/C++ Compiler
- Intel takes first call, Codeplay delivers backend support
- Codeplay provides access to older plug-in versions

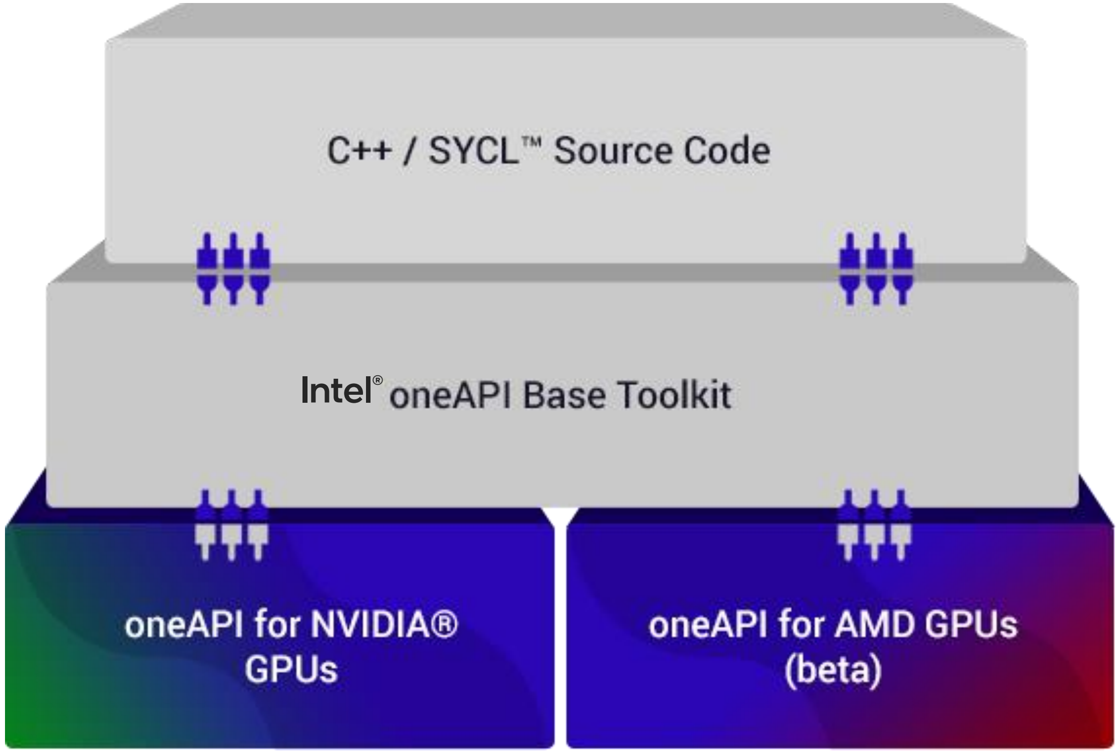


Image courtesy of Codeplay Software Ltd.

Nvidia GPU plug-in

AMD GPU plug-in

Codeplay blog

Codeplay press release

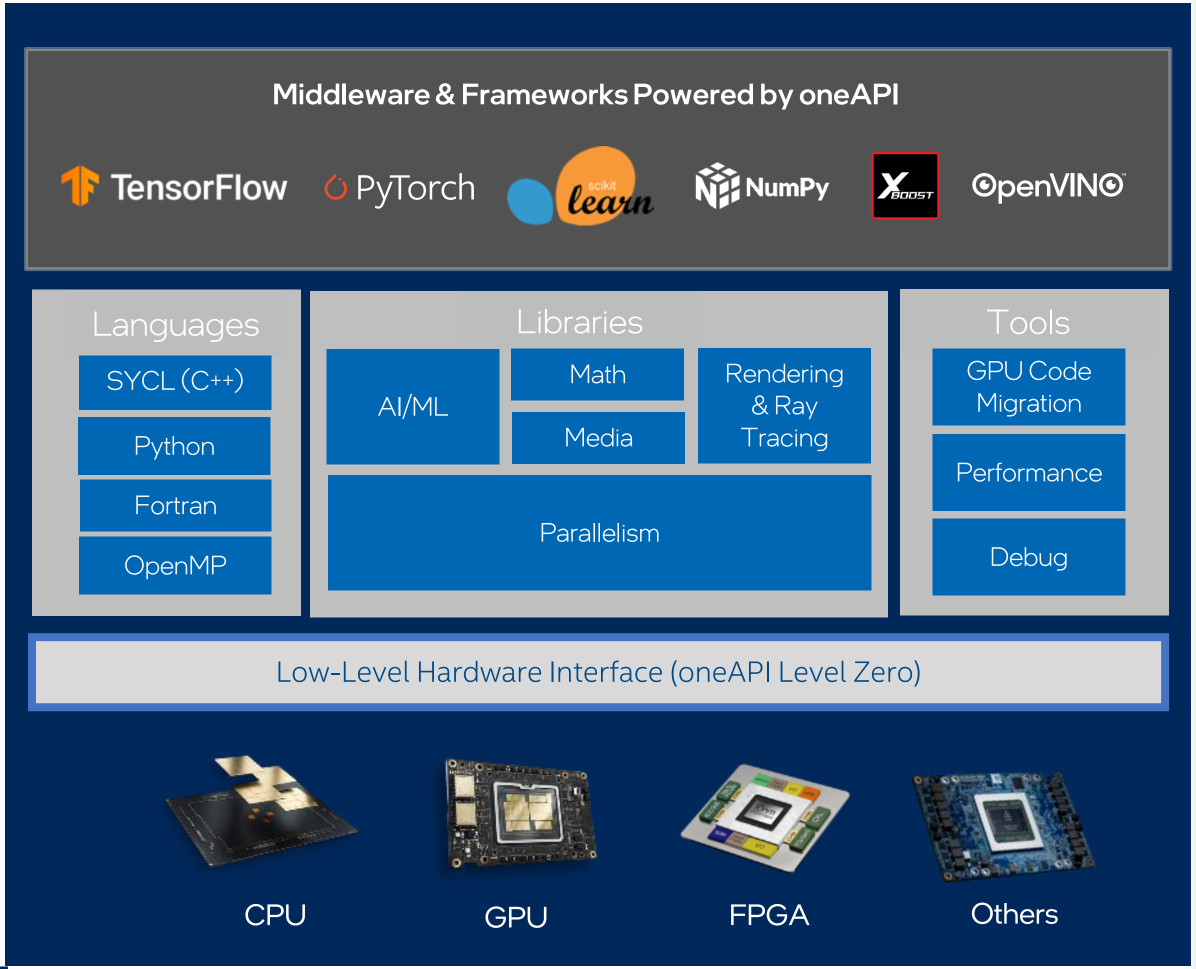
*Other names and brands may be claimed as the property of others.

Intel® Developer Tools Supporting oneAPI

A complete set of proven tools expanded from CPU to accelerators

oneAPI

- Advanced compilers, libraries, and analysis, debug, and porting tools
- Full support for C, C++ with SYCL, Python, Fortran, MPI, OpenMP
- Intel® Advisor determines device target mix before you write your code
- Intel's compilers optimize code to take full advantage of multiarchitecture workload distribution.
- Intel® VTune™ Profiler analyzes hotspots to optimize code performance
- Intel AI tools support acceleration of major deep learning and machine learning frameworks



Intel Analysis Tools for GPU Compute Analysis

Intel® Advisor

Offload Advisor

- Identify high-impact opportunities to offload
- Detect bottlenecks and key bounding factors
- Get your code ready even before you have the hardware by modeling performance, headroom, and bottlenecks

Roofline Analysis

- See performance headroom against hardware limitations
- Determine performance optimization strategy by identifying bottlenecks and which optimizations will pay off the most
- Visualize optimization progress

Intel® VTune™ Profiler

Offload Performance Tuning

- Explore code execution on your platform's various CPU and GPU cores
- Correlate CPU and GPU activity
- Identify whether your application is GPU- or CPU-bound

GPU Compute/Media Hotspots

- Analyze the most time-consuming GPU kernels, characterize GPU usage based on GPU hardware metrics
- GPU code performance at the source-line level and kernel-assembly level

Intel® oneAPI Toolkits

oneAPI

Intel® oneAPI Base Toolkit

A core set of high-performance libraries and tools for building C++, SYCL, C/OpenMP, and Python applications

Add-on Domain-specific Toolkits

For HPC developers

Intel® oneAPI Tools for HPC
Deliver fast Fortran, OpenMP & MPI applications that scale

For visual creators, scientists & engineers

Intel® oneAPI Rendering Toolkit
Accelerate visual compute, deliver high-performance, high-fidelity visualization applications.

For edge & IoT developers

Intel® oneAPI Tools for IoT
Build efficient, reliable solutions that run at network's edge

Toolkits powered by oneAPI

For AI developers & data scientists

Intel® AI Analytics Toolkit
Accelerate machine learning & data science pipelines end-to-end with optimized DL & ML frameworks & high-performing Python libraries

For deep learning inference developers

Intel® OpenVINO™ toolkit
Deploy high performance inference & applications from edge to cloud

Download at intel.com/oneAPI
Or visit Intel® DevCloud for oneAPI

Latest version available 2023

Intel® VTune™ Profiler Overview

Optimize Performance

Intel® VTune™ Profiler

Get the Right Data to Find Bottlenecks

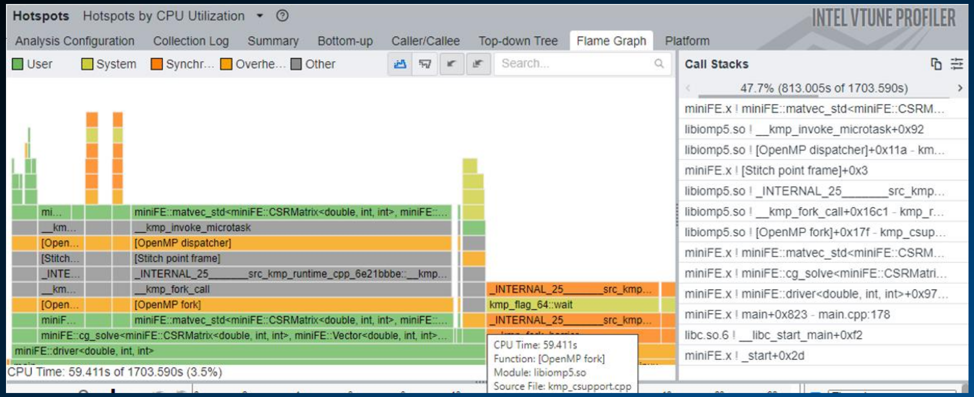
- A suite of profiling for CPU, GPU, FPGA, threading, memory, cache, storage, offload, power…
- Application or system-wide analysis
- DPC++, C, C++, Fortran, Python*, Go*, Java*, or a mix
- Linux, Windows, FreeBSD, Android, Yocto and more
- Containers and VMs

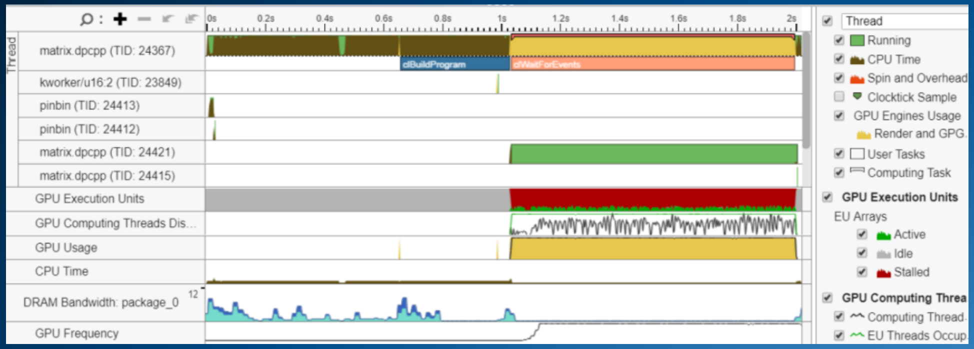
Analyze Data Faster

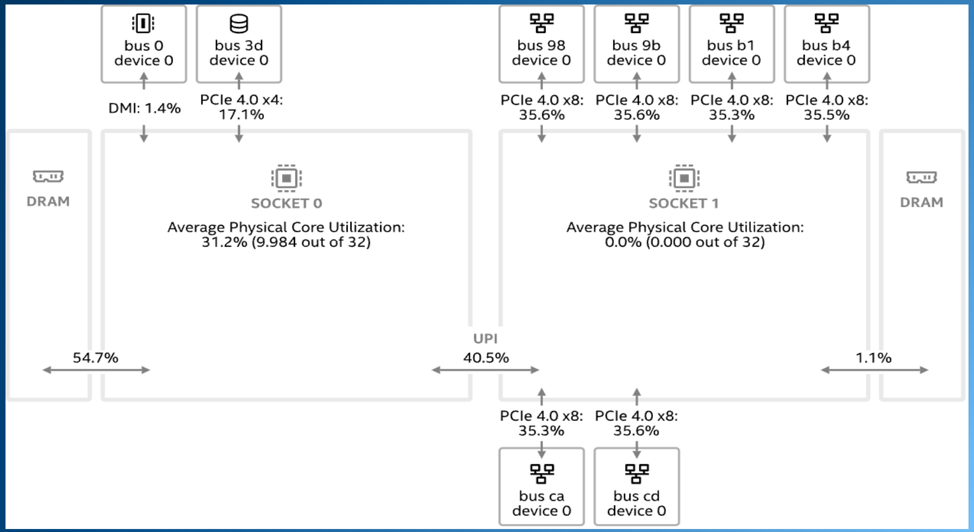
- Collect data HW/SW sampling and tracing w/o re-compilation
- See results on your source, in architecture diagrams, as a histogram, on a timeline…
- Filter and organize data to find answers

Work Your Way

- User interface or command line
- Profile locally and remotely
- GUI (desktop or web) or command line

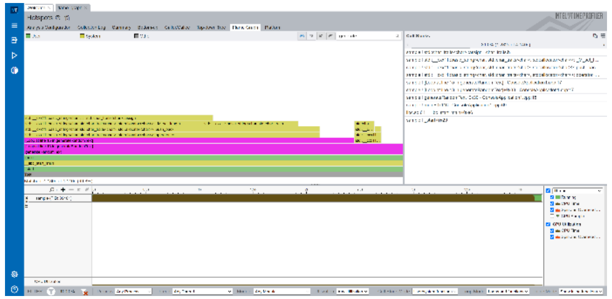


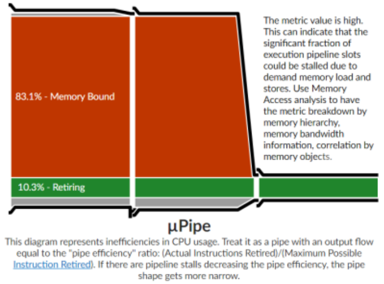


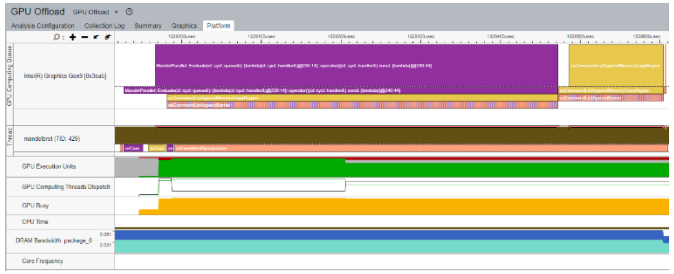


Rich Set of Profiling Capabilities

Intel® VTune™ Profiler







Algorithm Optimization

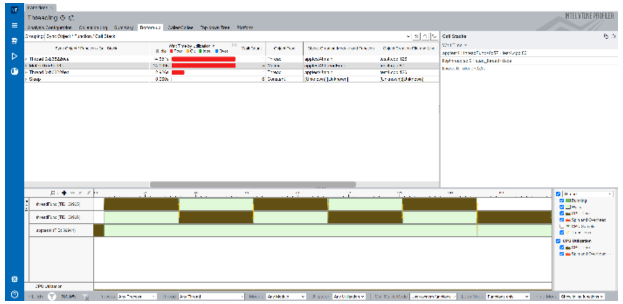
- ✓ Hotspots
- ✓ Anomaly Detection
- ✓ Memory Consumption

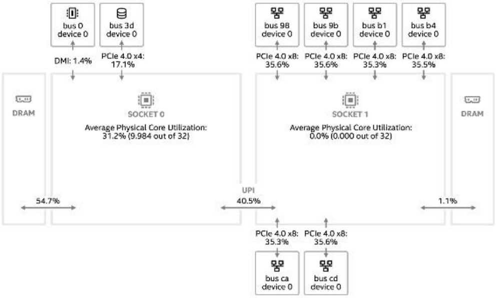
Microarch.&Memory Bottlenecks

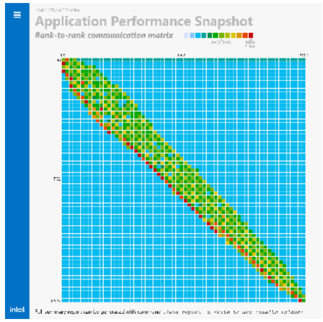
- ✓ Microarchitecture Exploration
- ✓ Memory Access

Accelerators / xPU

- ✓ GPU Offload
- ✓ GPU Compute / Media Hotspots
- ✓ CPU/FPGA Interaction







Parallelism

- ✓ Threading
- ✓ HPC Performance Characterization

Platform & I/O

- ✓ Input and Output
- ✓ System Overview
- ✓ Platform Profiler

Multi-Node

- ✓ Application Performance Snapshot

What's New in Intel® VTune™ Profiler

2023.1 and 2023.0 Releases

Profile your applications running on latest Intel HW

- 4th generation Intel® Xeon® Scalable processors (formerly code named Sapphire Rapids)
- Intel® Xeon® Max Series CPUs (code named Sapphire Rapids HBM)
- 13th generation Intel® Core™ processors (formerly code named Raptor Lake),
- Intel® Data Center GPU Max Series (formerly code named Ponte Vecchio).

Accelerate GPU code

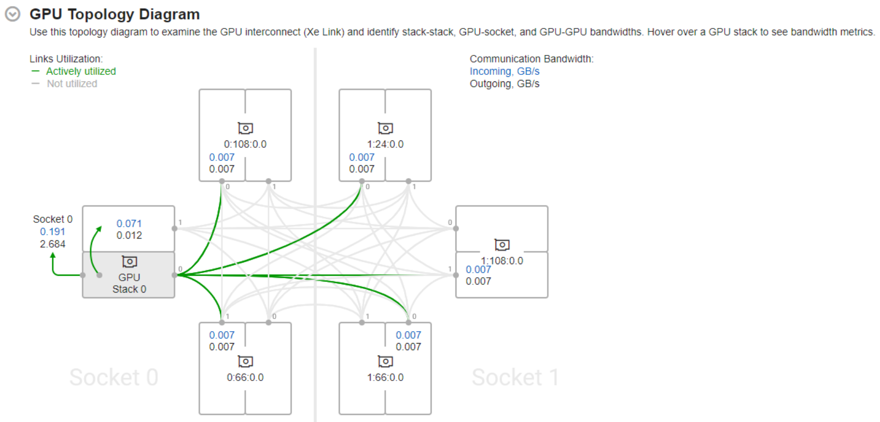
- Get visibility into XeLink cross-card traffic for issues such as stack-to-stack traffic, throughput and bandwidth bottlenecks. Identify imbalances of traffic between CPU and GPU through a GPU topology diagram.
- Identify the the reasons of the stalls in Xe Vector Engines (XVEs), formerly known as Execution Units (EUs). Use this information to better understand and resolve the stalls in your busiest computing tasks.
- Profile applications executing on multiple GPUs.

Optimize Python code

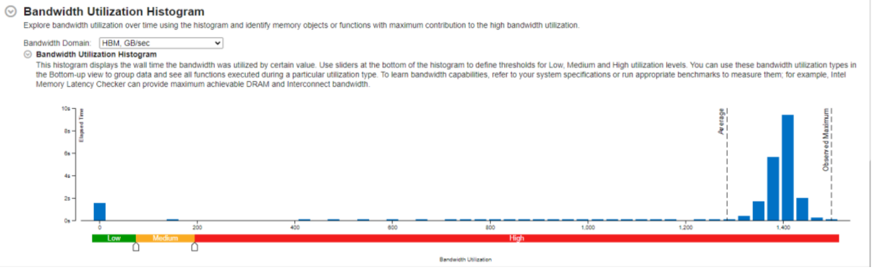
- Identify and optimize performance hotspots of Python code, now supporting Python 3.9.*.

Decide memory mode for your workload

- Identify performance gained from high bandwidth memory (HBM). Run Intel® VTune Profiler for each mode (HBM only, Flat, Cache) to identify which profile offers the best performance.

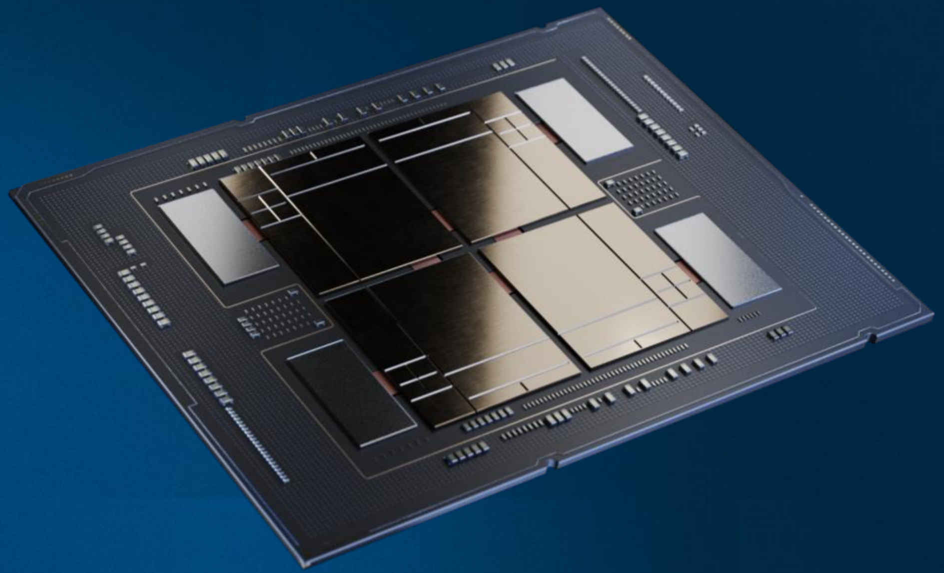


Cross-card, stack-to-stack, and card-to-socket bandwidth are presented on GPU Topology Diagram.



The histogram shows the distribution of the elapsed time per maximum bandwidth utilization among all packages.

intel
Xeon
MAX SERIES



Only x86 CPU with High Bandwidth Memory

64GB
HBM2e

up to
112.5MB
shared LLC

DDR5
8 channels per CPU @ 4800MTS (1DPC) / 16 DIMMs per socket

~1TB/s memory BW

>1GB/core HBM memory capacity

* Relative performance ISO TDP and core count

Memory modes

HBM Only

Workloads ≤ 64GB capacity

No code change
No DDR

System boots and operates with HBM only

HBM Flat Mode

Flat Mem Regions w/ HBM & DRAM
Workloads > 64GB capacity

Code change may be needed to optimize perf

Provides flexibility for applications that require large memory capacity

HBM Caching Mode

DRAM backed cache
Improved performance for workloads > 64GB capacity

No code change
HBM Caches DDR

Blend of both prior modes. Whole applications may fit in HBM cache
Blurs line between cache and memory

intel.

High Bandwidth Memory (HBM) Utilization

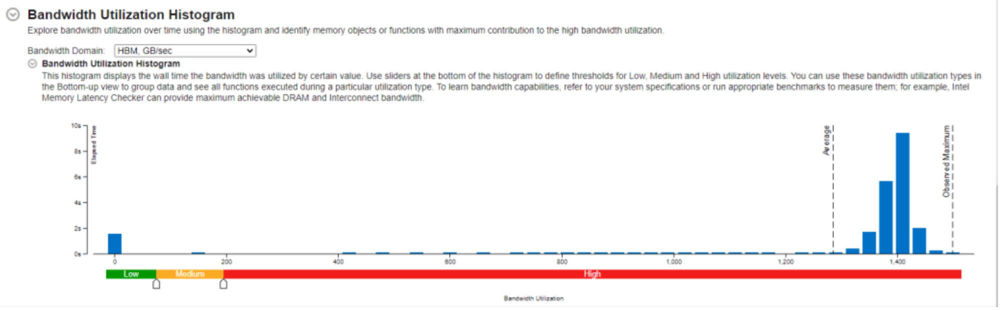
Intel® VTune™ Profiler

Understand HBM memory usage

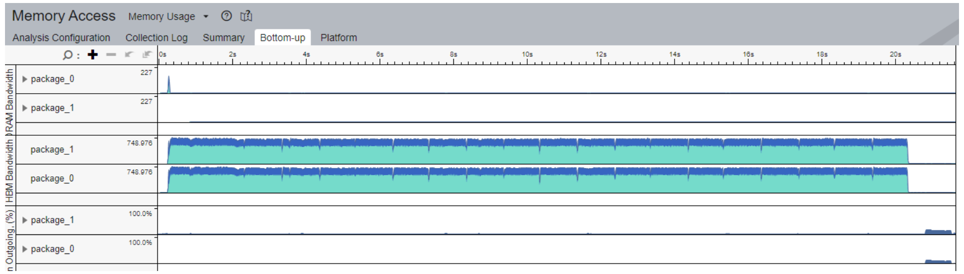
- Is the application performance affected by HBM utilization?
- How is the bandwidth distributed between DRAM vs. HBM?

Identify memory mode for your workload

- Does your workload benefit from HBM?
    - Profile your workload for each mode - HBM, flat or cache



The histogram shows the distribution of the elapsed time per maximum bandwidth utilization among all packages.



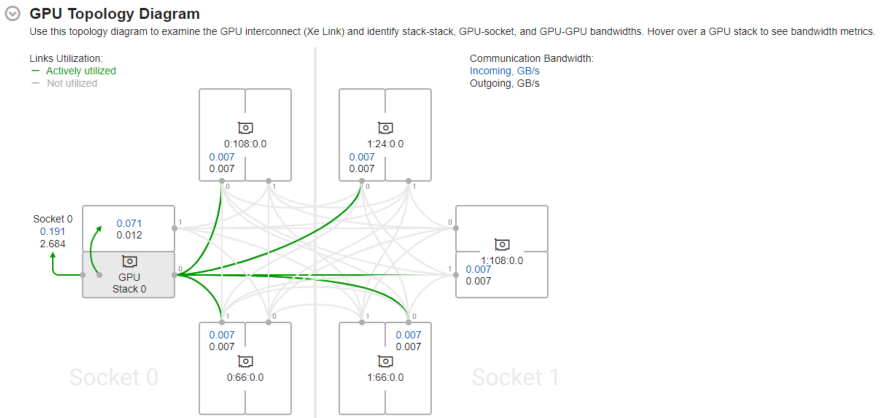
The workload performance in various HBM modes can be evaluated by running the collection in each mode and analyzing the bandwidth as described above.

Get Visibility into Xe Link Cross-card Traffic

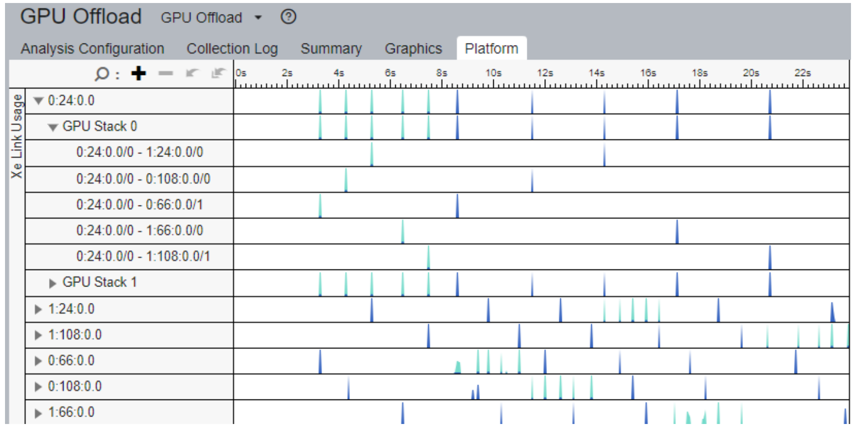
Intel® VTune™ Profiler

Identify bottlenecks related to Xe Link

- Understand cross-card memory transfers and Xe Link utilization
- Visualize GPU Topology of the system and estimate bandwidth of each link, stack or card.
- See usage of Xe Link and correlate with code execution.



Cross-card, stack-to-stack, and card-to-socket bandwidth are presented on GPU Topology Diagram.



Timeline view can show bandwidth usage of Xe Link over time.

Command Line Interface

Automate analysis

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

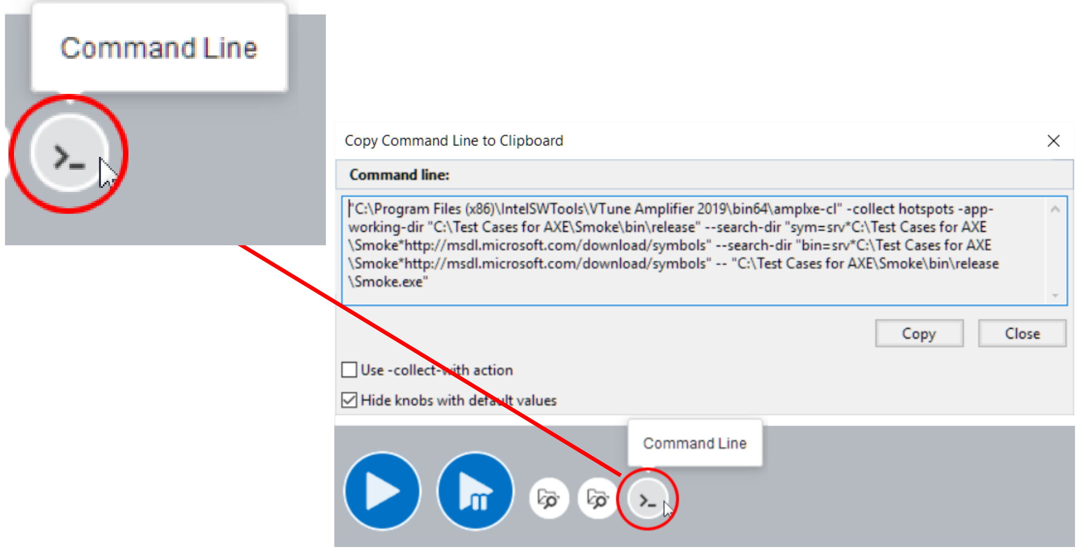
Help:

vtune –help

vtune –help collect hotspots

Use UI to setup

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



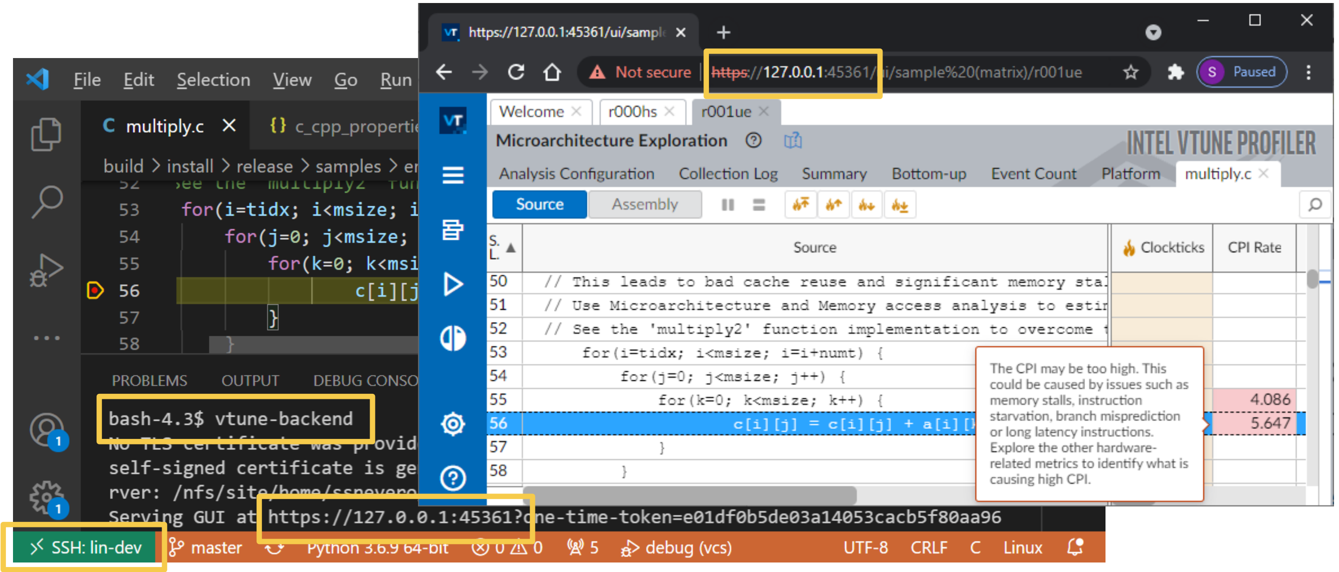
vtune -collect hpc-performance [-knob <knobName=knobValue>] [--] <app>

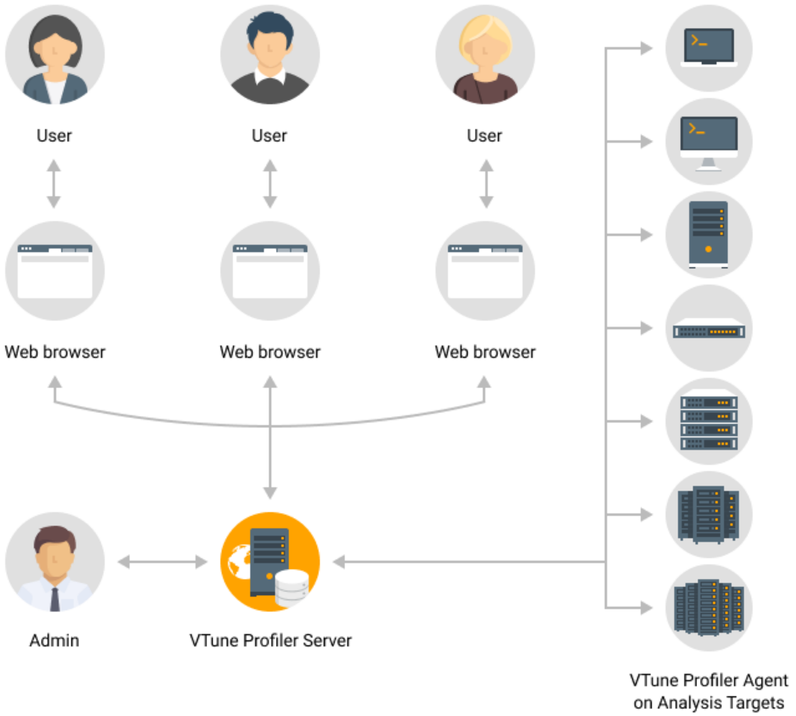
mpiexec –n 12 vtune –c gpu-hotspots –r gpuhs_mpi –trace-mpi [-knob <knobName=knobValue>] [--] <app>

Intel® VTune™ Profiler Server

Does your development move to Cloud? VTune is ready to follow!

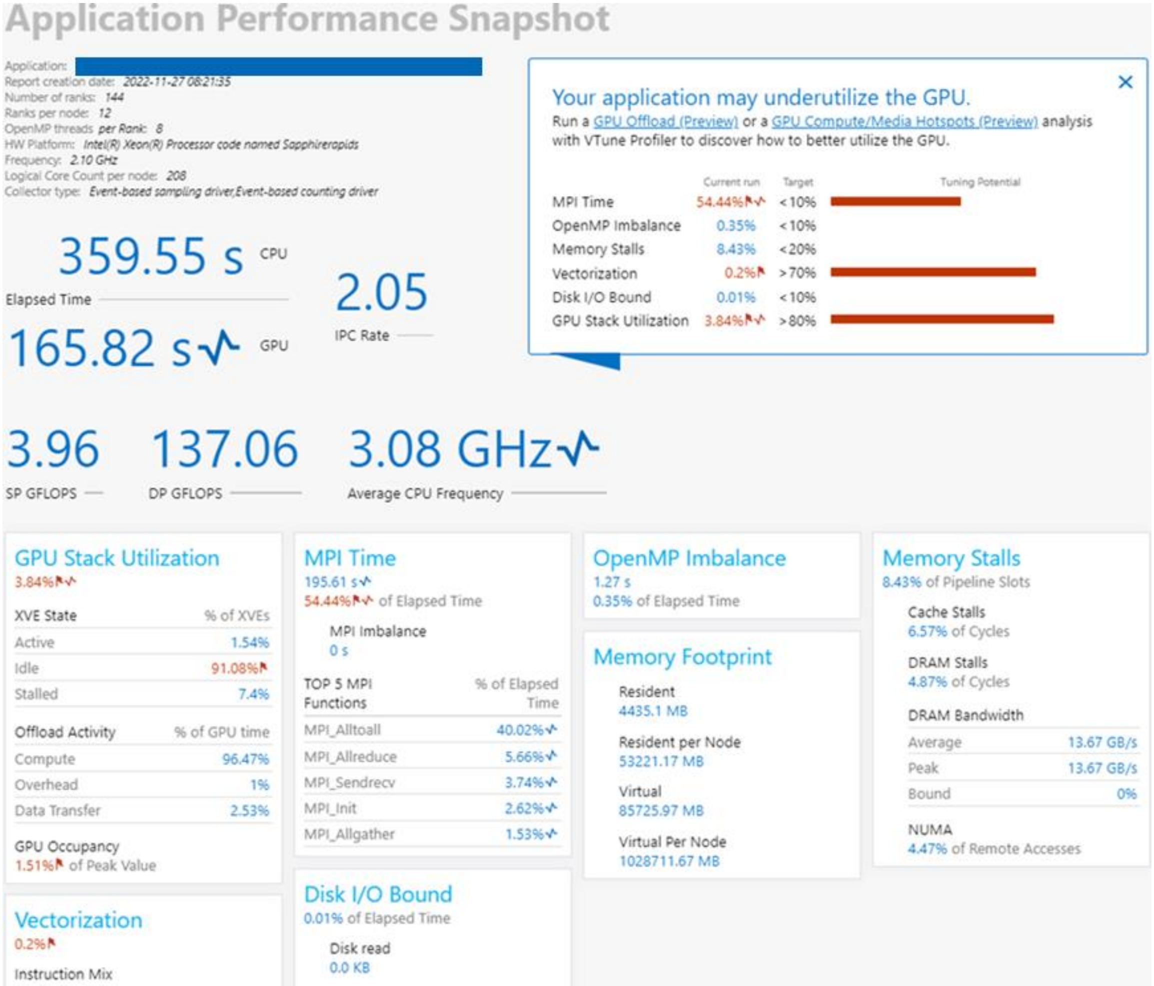
- VTune server for remote development





- VTune server for teams
  - Easy onboarding
  - Data sharing & collaboration

Intel® VTune™ Profiler Application Performance Snapshot (APS)

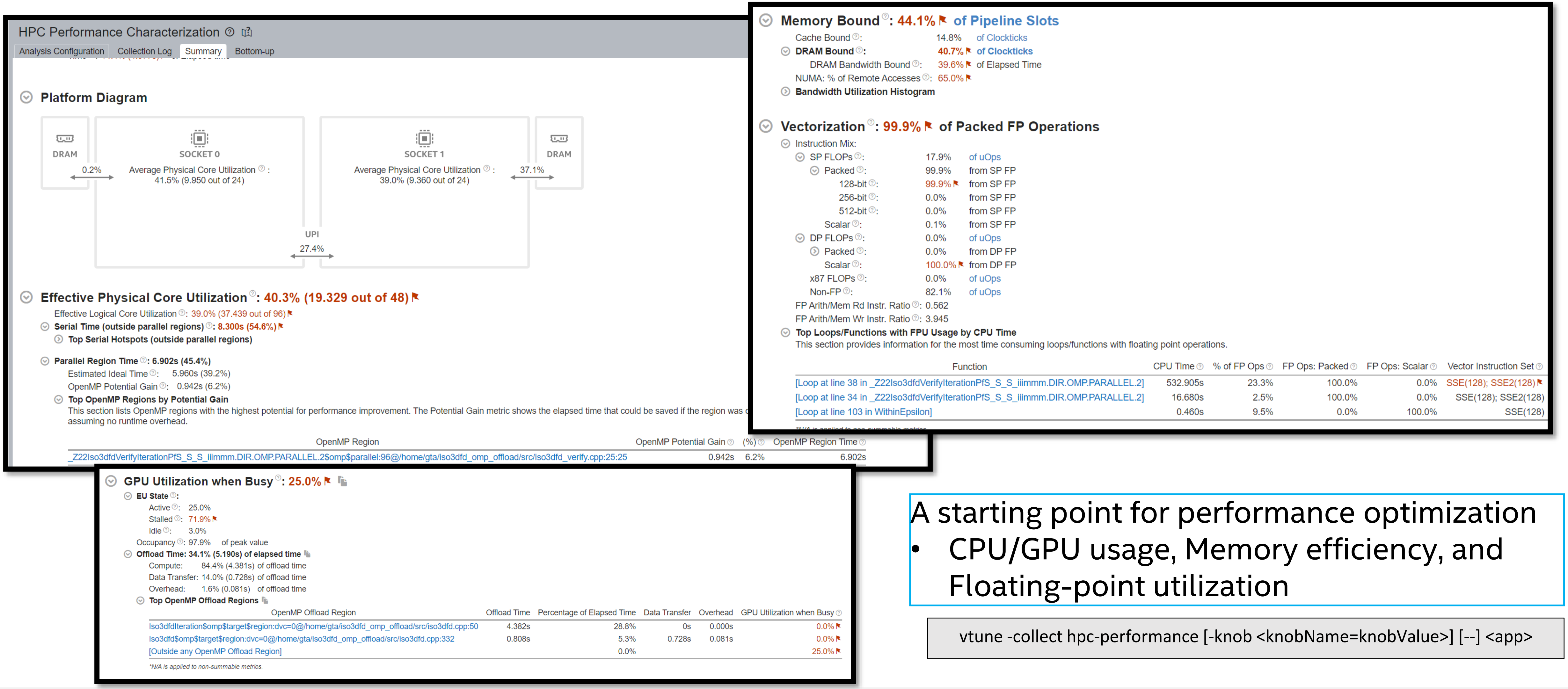


- High-level **overview** of application performance
  - Detailed reports on MPI statistics
- Primary optimization areas and **next steps** in analysis with deep tools – e.g. outlier analysis for MPI applications at scale
  - Explore on source of imbalance
  - Choose nodes/ranks for detailed profiling with VTune
- **Low** collection overhead – 1-3%*
- **Scales** to large jobs
  - Tested and worked on 64K ranks
  - Trace size on default statistics level ~ 4Kb per rank
- Command Line:

```
<mpi launcher> <mpi parameters> aps <app>
```

Intel® VTune™ Profiler

HPC Performance Characterization



A starting point for performance optimization

- CPU/GPU usage, Memory efficiency, and Floating-point utilization

```
vtune -collect hpc-performance [-knob <knobName=knobValue>] [--] <app>
```

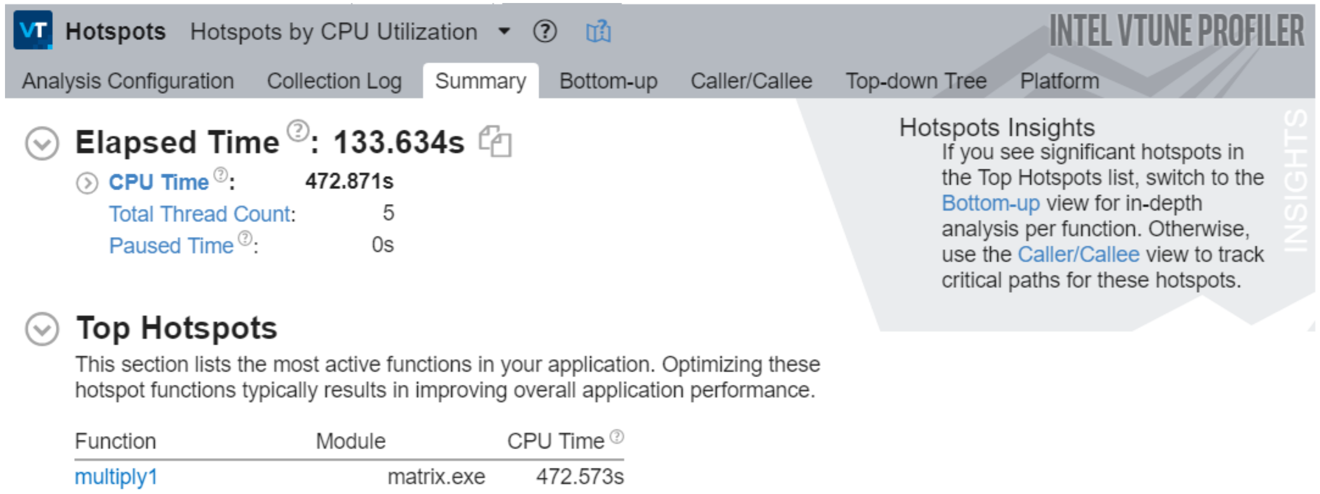
Hotspots Analysis

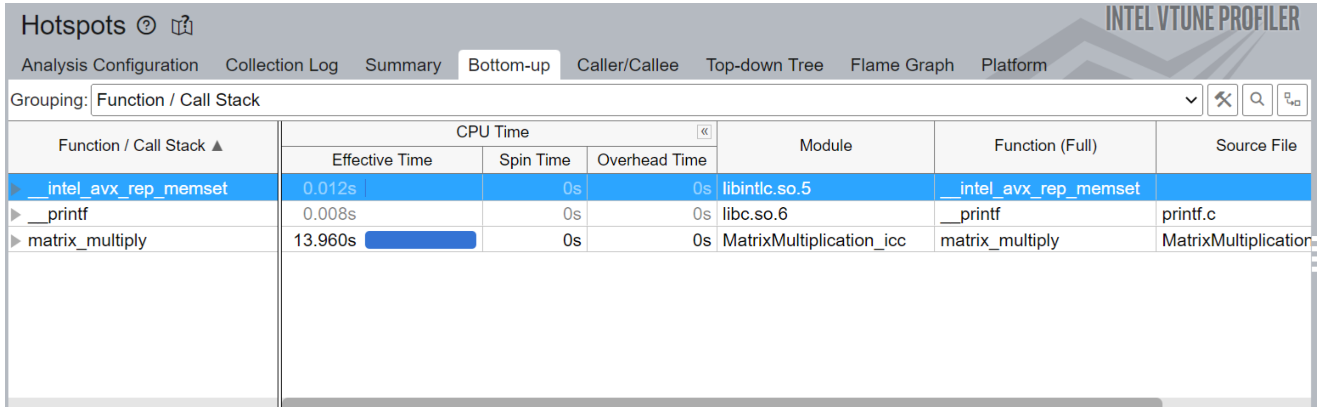
- Understand an application flow
- Identify sections of code that get a lot of execution time
- Sampling-based collection modes
  - User-Mode Sampling
  - Hardware Event Based Sampling
- Define a performance baseline.

- Identify the hottest function.

- Identify algorithm issues.

- Analyze source.



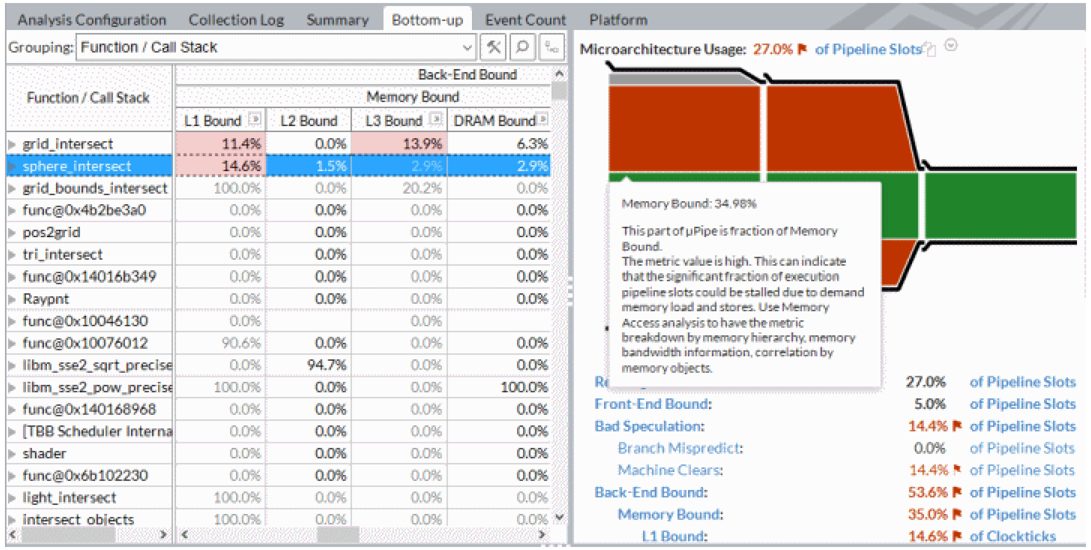


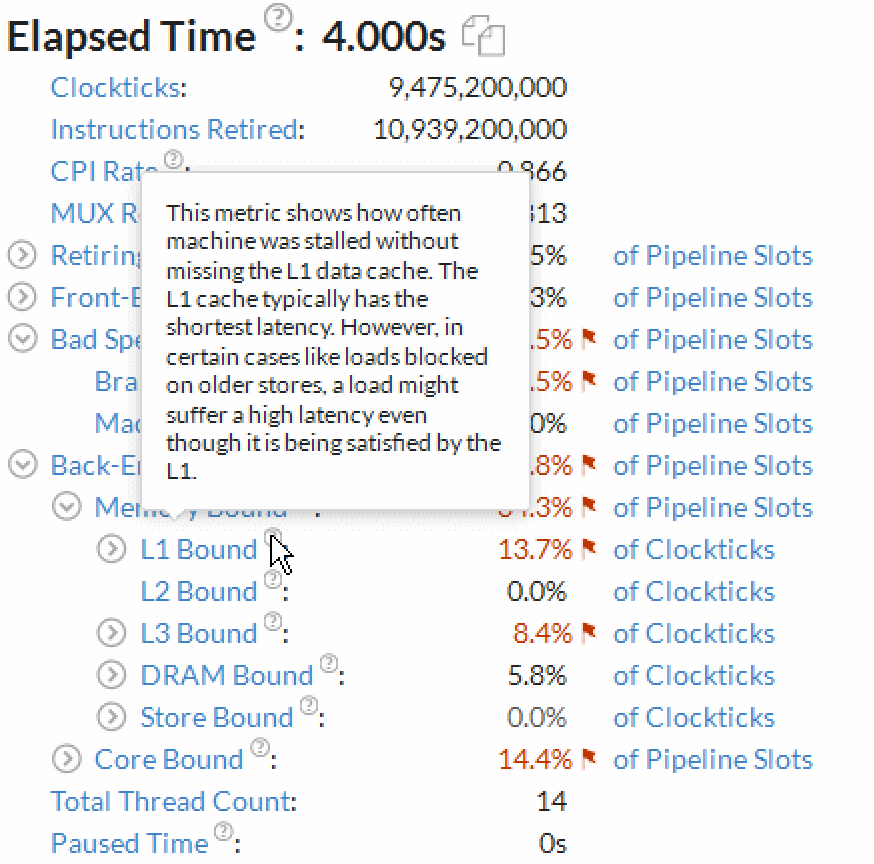
Microarchitecture Exploration

Hierarchical view of the execution pipeline

- Pinpoint sections of the pipeline with performance problems flagged by VTune
- Hover over metrics for a detailed description

Visualize the pipeline at the function level in the bottom-up tab





What's Using All The Memory?

Memory Consumption Analysis

See What Is Allocating Memory

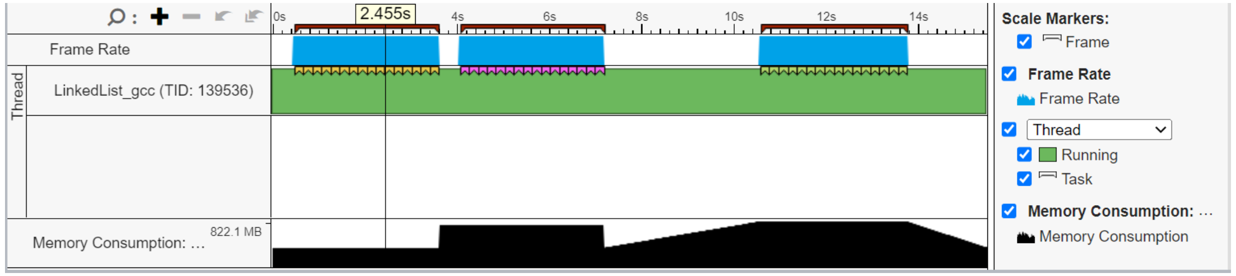
- Lists top memory consuming functions
- memory consumption distribution over time.
- View source to understand cause
- Filter by time using the memory consumption timeline
  - Focus on the peak values on the Timeline pane
- Introduce additional overhead due to instrumentation .

Top Memory-Consuming Functions

This section lists the most memory-consuming functions in your application.

Function	Memory Consumption	Allocation/Deallocation Delta	Allocations	Module
create_linked_list	469.8 MB	0.0 B	4,194,304	LinkedList_gcc
create_data	402.7 MB	0.0 B	1	LinkedList_gcc
create_array_data	352.3 MB	352.3 MB	7	LinkedList_gcc
itt_init	47.7 KB	8.3 KB	99	LinkedList_gcc
[Unknown stack frame(s)]	528.0 B	528.0 B	11	[Unknown]
[Others]	96.0 B	96.0 B	3	N/A*

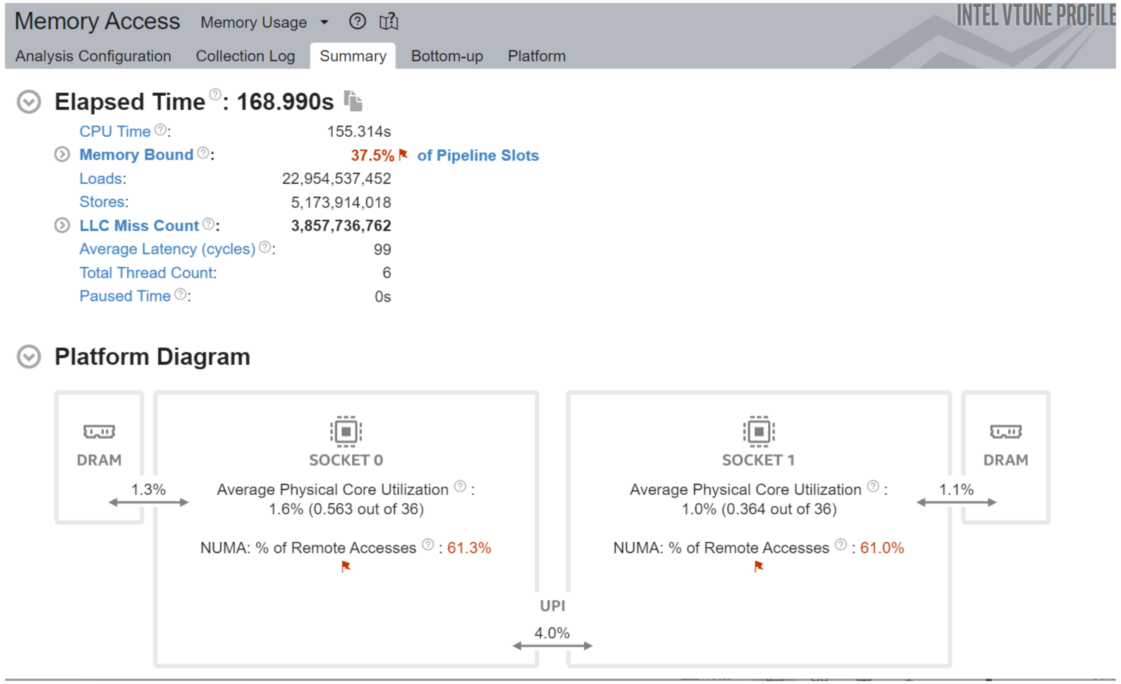
*N/A is applied to non-summable metrics.



Native language support is not currently available for Windows*

Optimize Memory Access
Memory Access Analysis - Intel® VTune™ Profiler

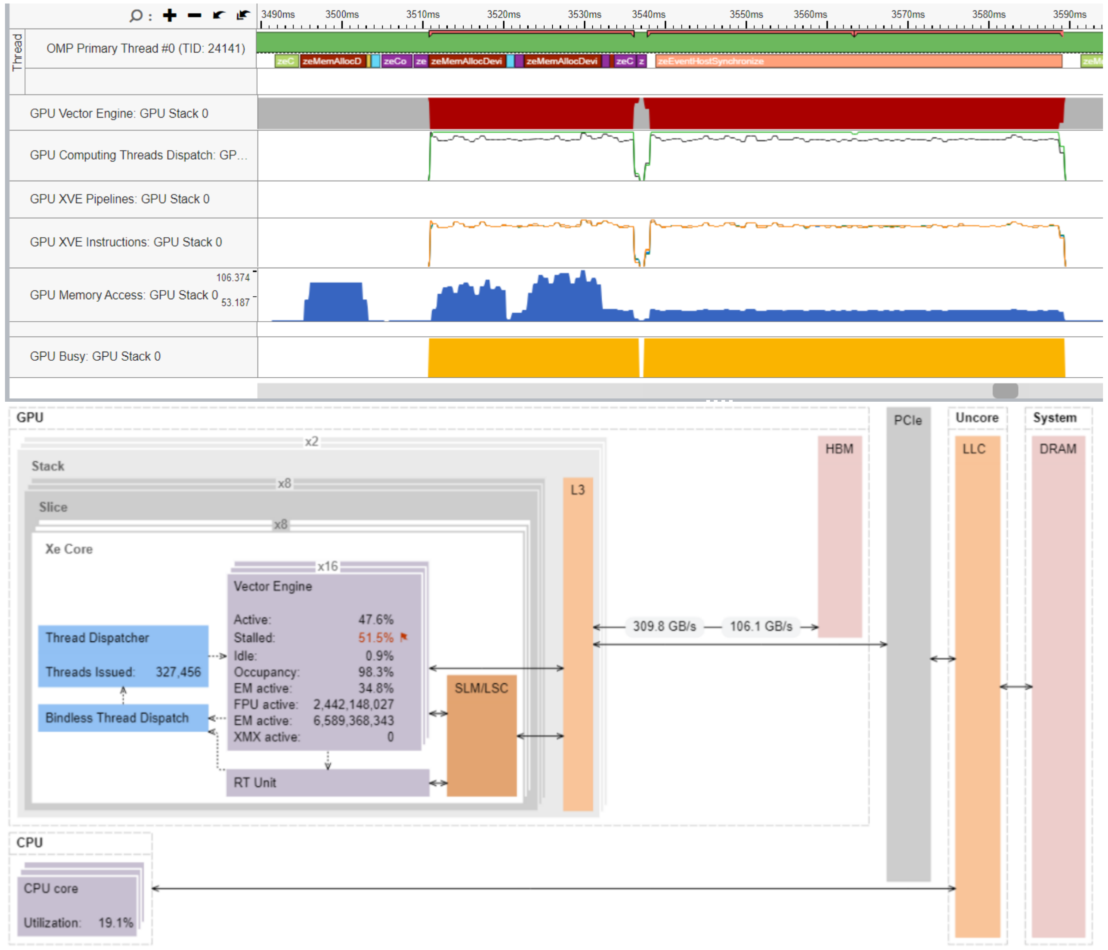
- Tune data structures for performance
  - Attribute cache misses to data structures (not just the code causing the miss)
  - Support for custom memory allocators
  - Shows average load latency in cycles
- Optimize NUMA latency & scalability
  - Auto detect max system bandwidth
  - Detects inter-socket bandwidth



Intel® VTune™ Profiler

Profile GPU Performance

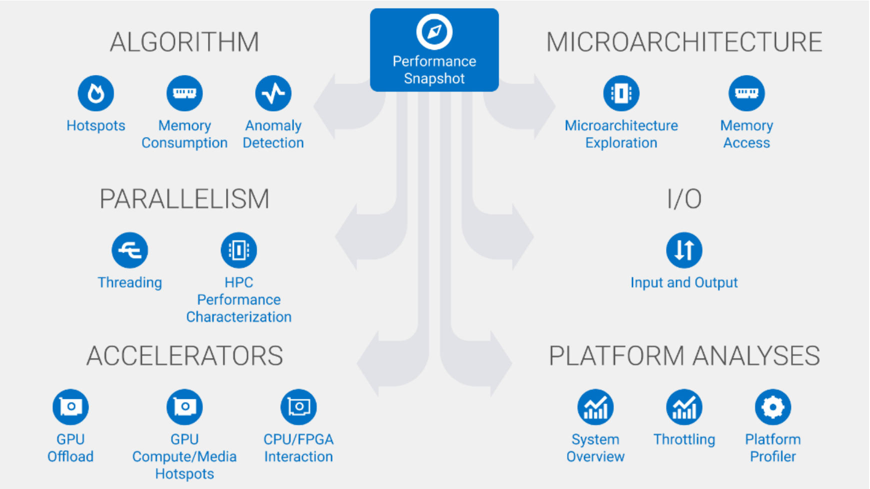
- Explicit support of **DPC++, DirectX, Intel® Media SDK, OpenCL™, and OpenMP-offload** software technology
- **Multi-GPU systems analysis**
- **GPU Offload cost profiling**
  - CPU vs GPU boundness
  - Offload overhead & host-to-device traffic, GPU compute vs data transfer
  - GPU utilization and software queues per DMA packet domain
- **GPU Hotspots analysis**
  - EU and memory efficiency metrics, GPU Occupancy limiting factors
  - Memory hierarchy diagram and throughput analysis
- **Source level in-kernel profiling**
  - Dynamic instruction count
  - Basic Block execution latency
  - Memory latency

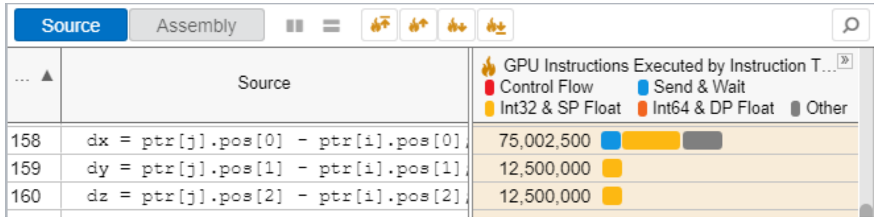


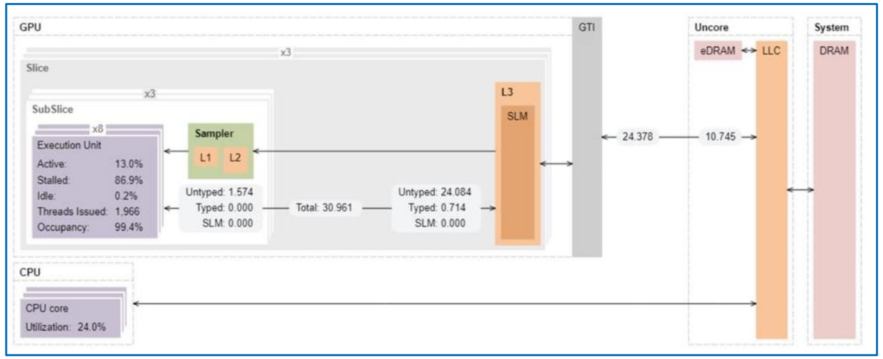
GPU Performance Problems

Addressing performance issues with dynamic analysis tools

- Work Distribution
- Data transfer
- GPU occupancy
- Memory access
- Kernel inefficiencies
- Non-scaling implementations
- …



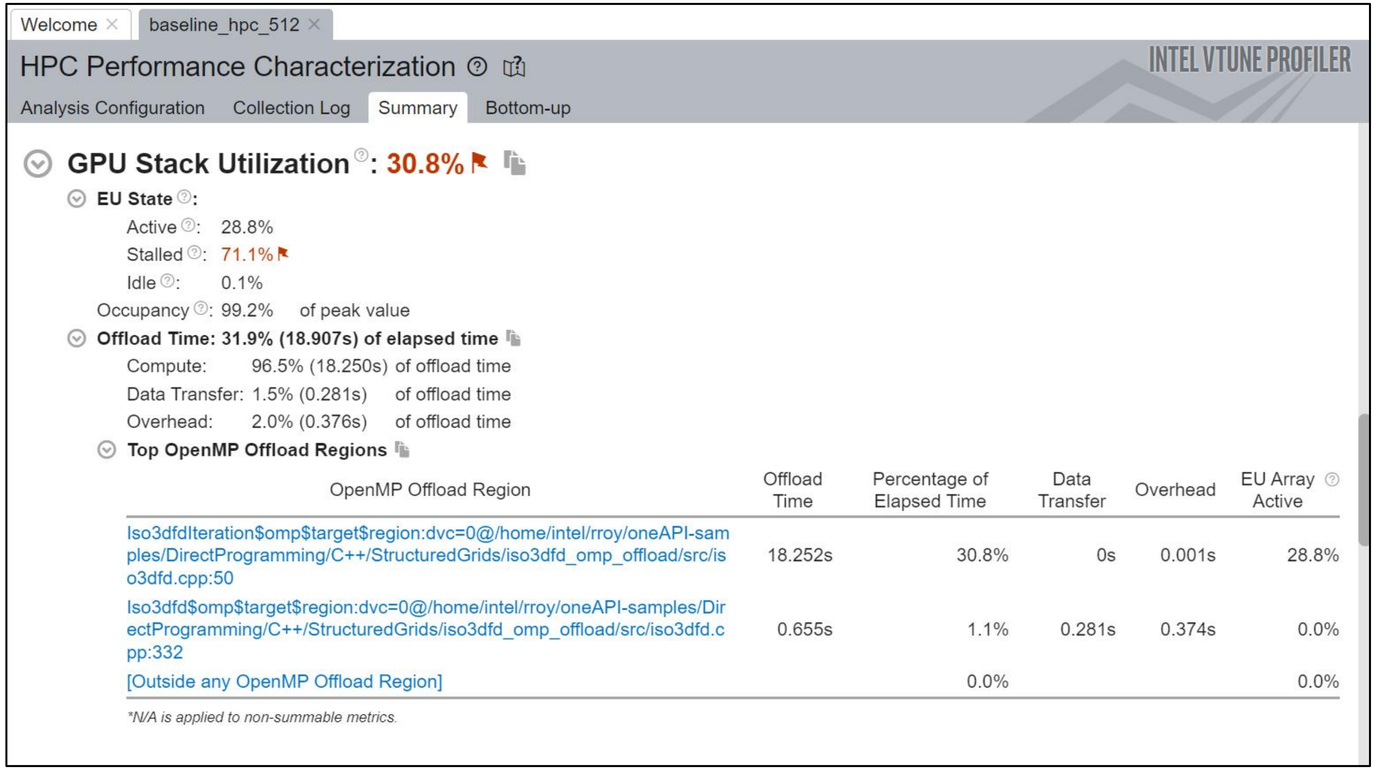




Work Distribution

Work distribution among computing resources

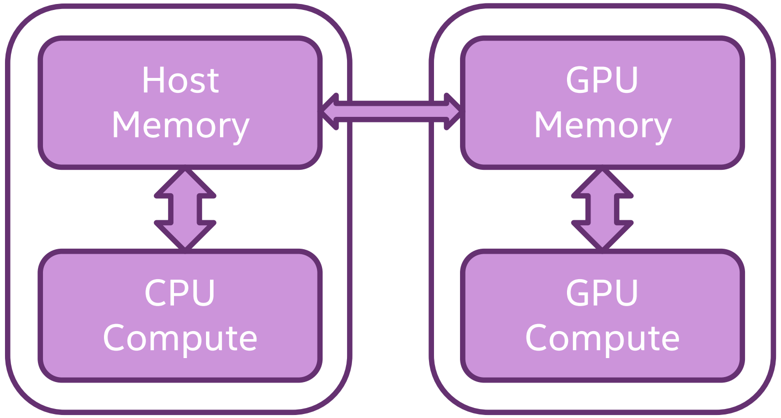
- CPU or GPU bound?
- GPU Utilization for OpenMP regions/SYCL kernels
- EU/XVEs efficiency (Active, Stalled, Idle)
- Offload Time characterization
  - Compute
  - Data Transfer
  - Overhead



Host and GPU Data Transferring

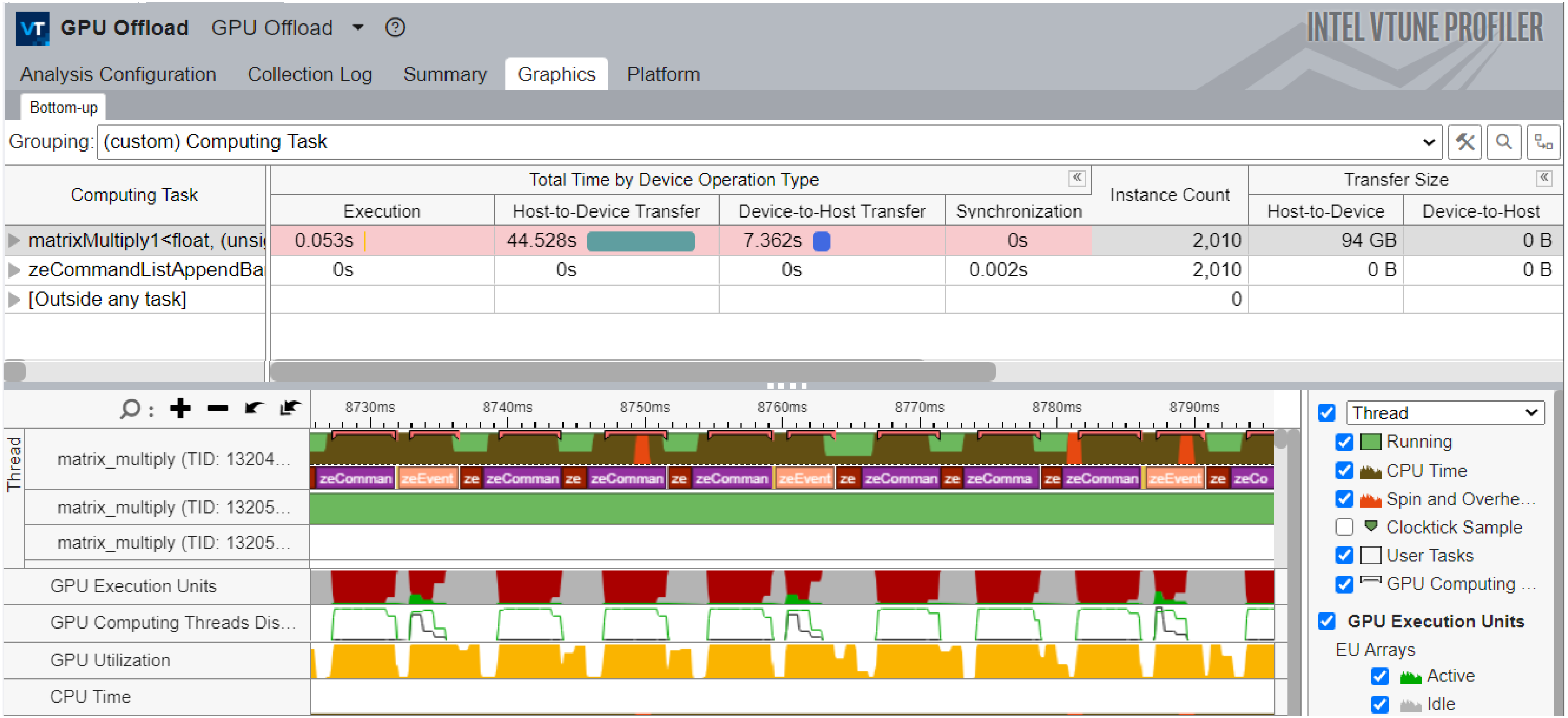
A commonly known problem of host-to-device transfer performance

- Data transfer time
- Amount of transferred data
- Transfer direction
- Execution time




```
vtune -collect gpu-offload [-knob <knobName=knobValue>] [--] <target>
```

Graphics View of GPU Offload

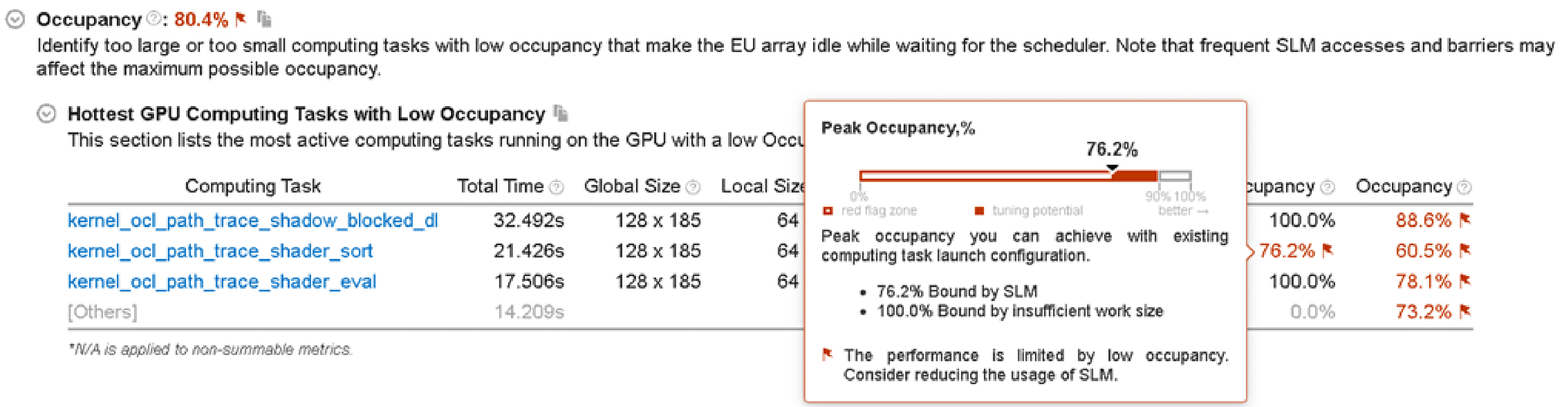


Achieving High XVE Threads Occupancy

Occupancy analysis helps identifying problems with work mapping

- Detecting workgroups by global and local sizes
- SIMD Width
- Barriers usage
- Tiny/huge kernels

  scheduling issues

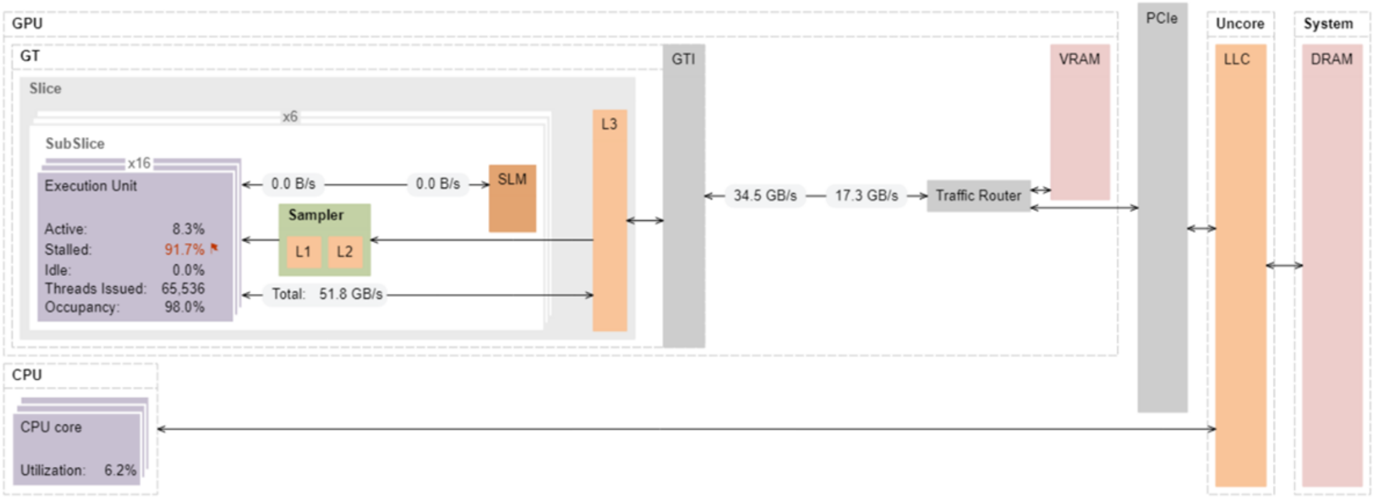



```
vtune -collect gpu-hotspots [-knob <knobName=knobValue>] [--] <target>
```

Memory Access problems

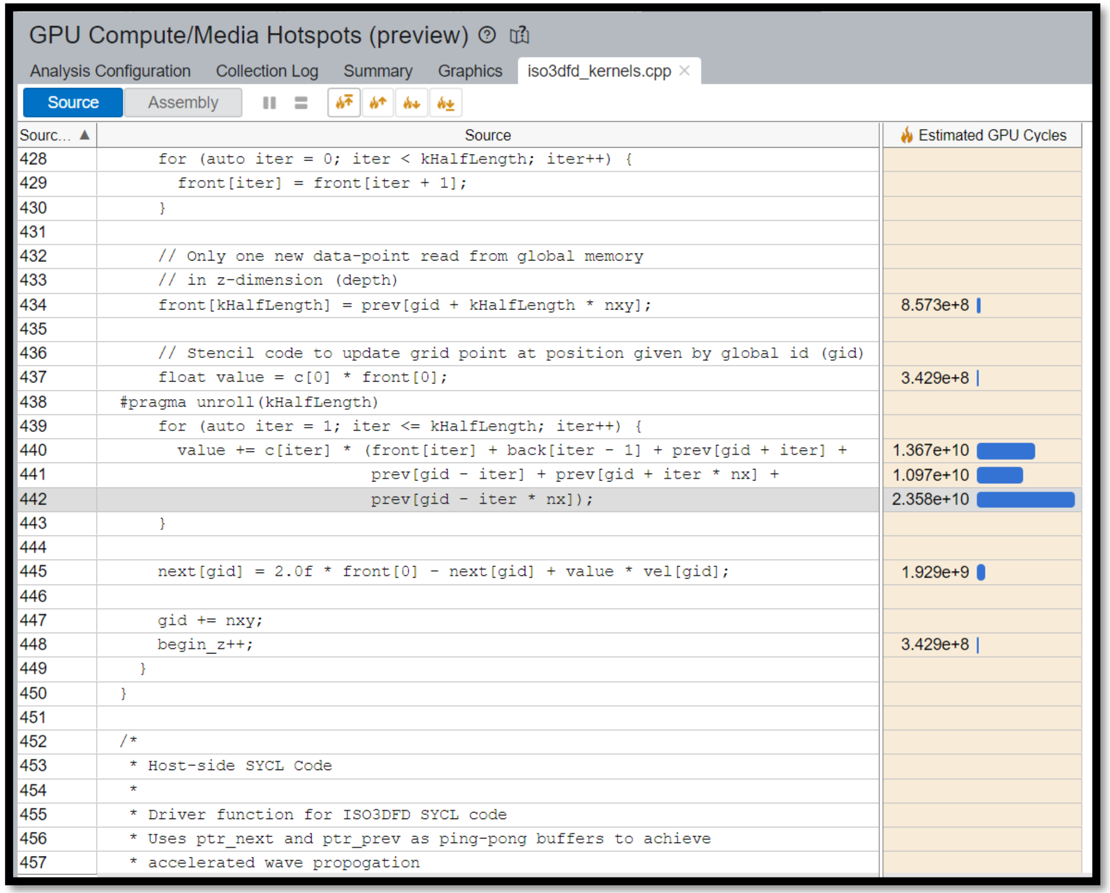
- Global memory access penalty
- Cache memory resource limit

- Which code is responsible for latency?
- Per Basic Block and latencies per individual instructions

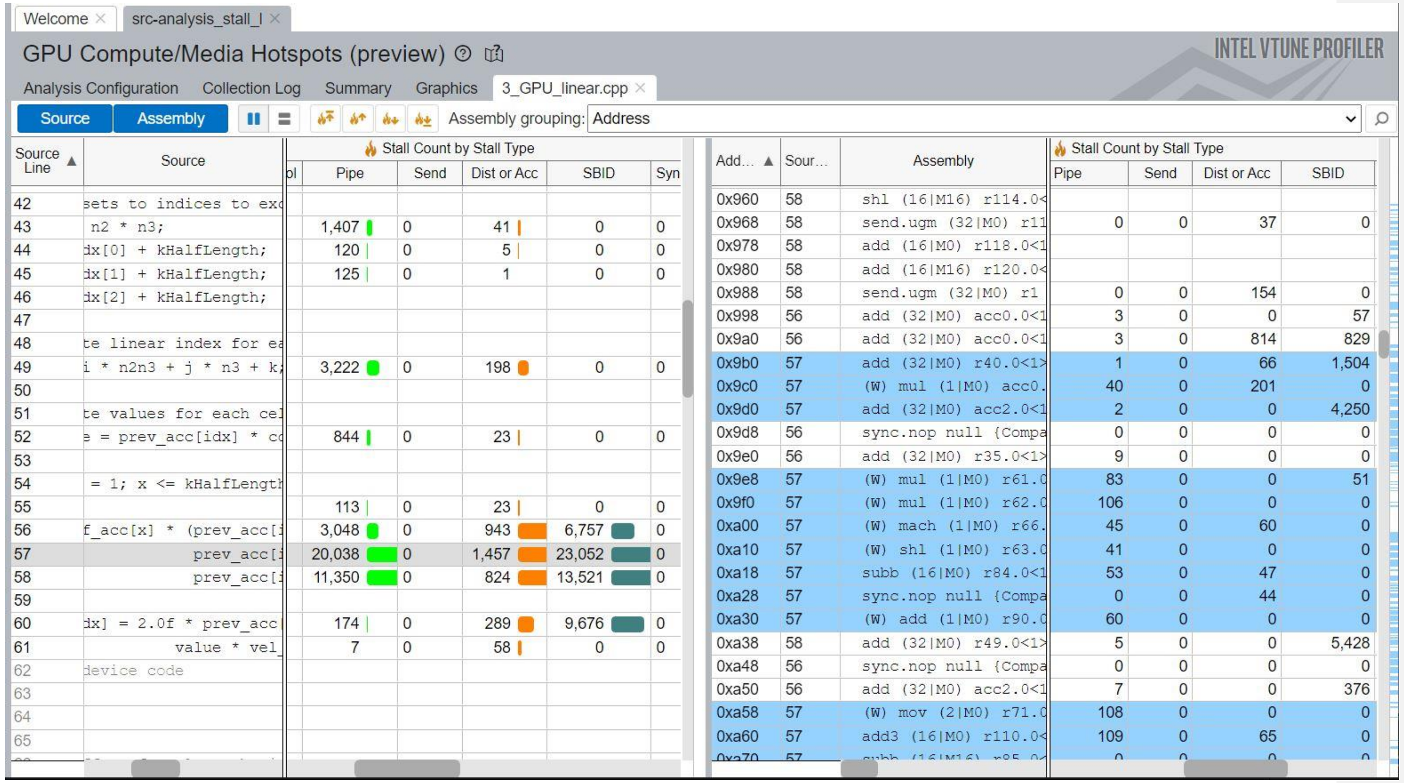


Source level in-kernel profiling

-knob computing-task-of-interest=*pattern*#1#10#100

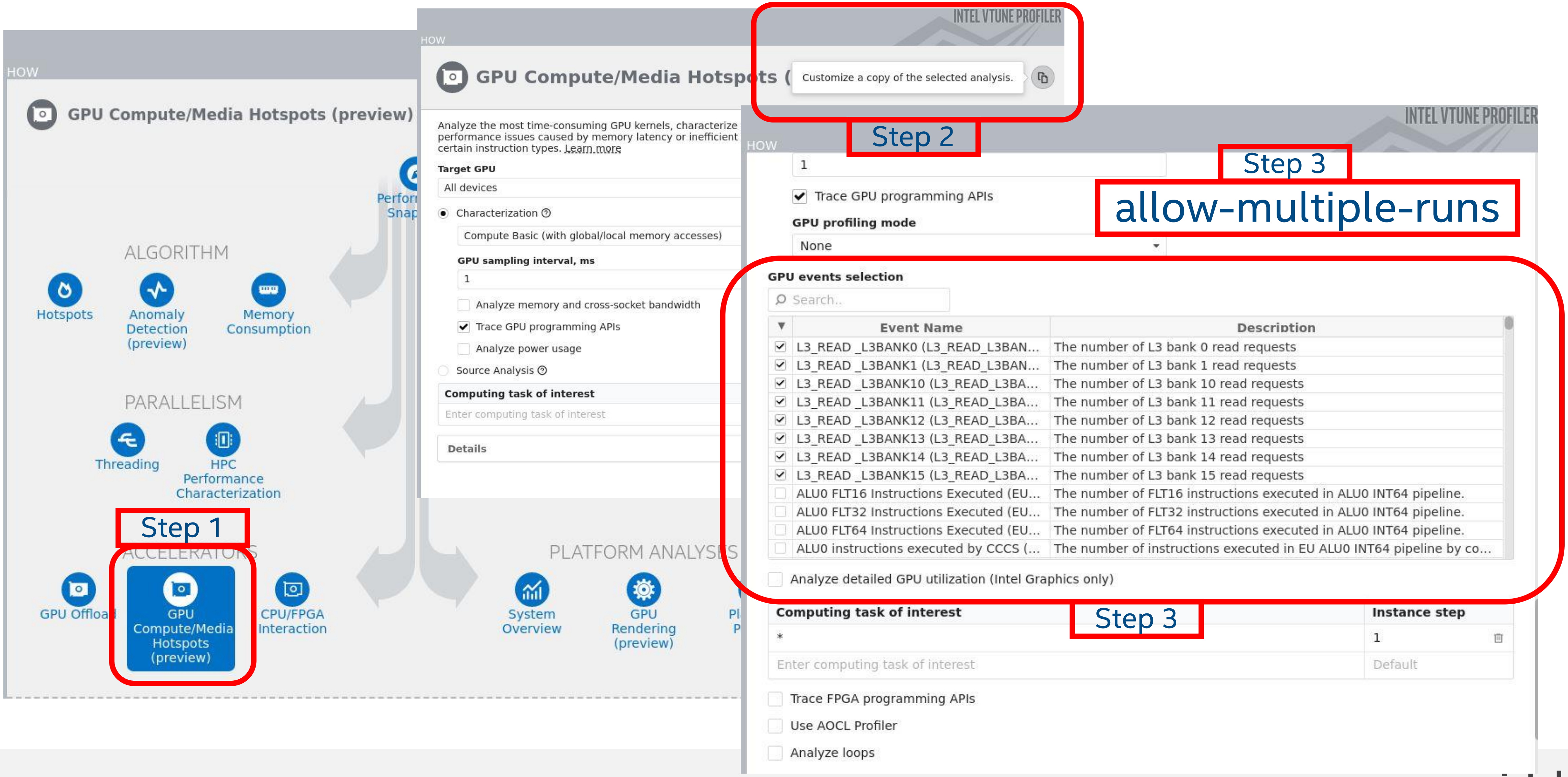


Basic Block Latency



Stall Sampling

Custom Analysis with VTune Profiler



Intel® Advisor Overview

Intel® Advisor

Design code for modern hardware

Offload Modelling

- Efficiently offload your code to GPUs even before you have the hardware

Automated Roofline Analysis

- Optimize your GPU/CPU code for memory and compute

Vectorization Optimization

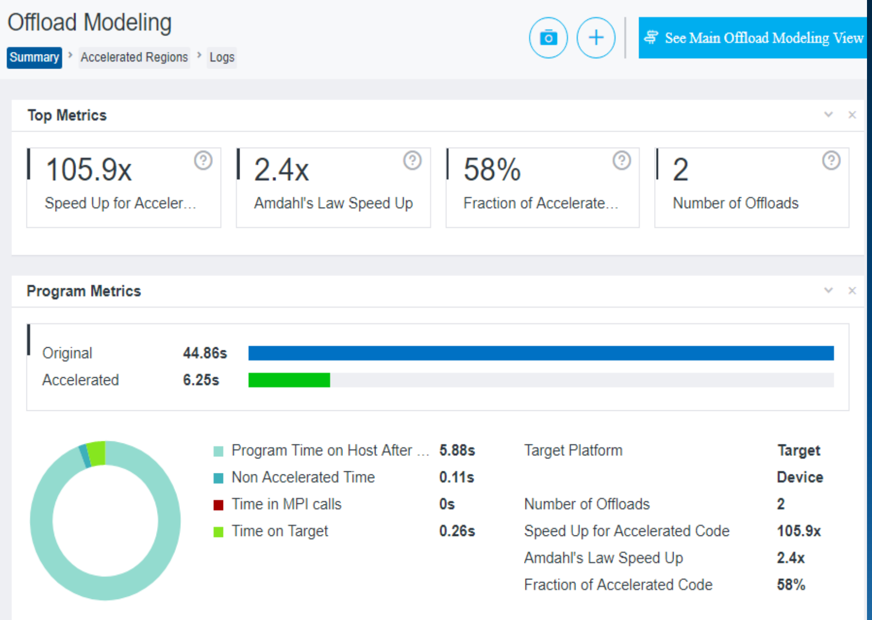
- Enable more vector parallelism and improve its efficiency

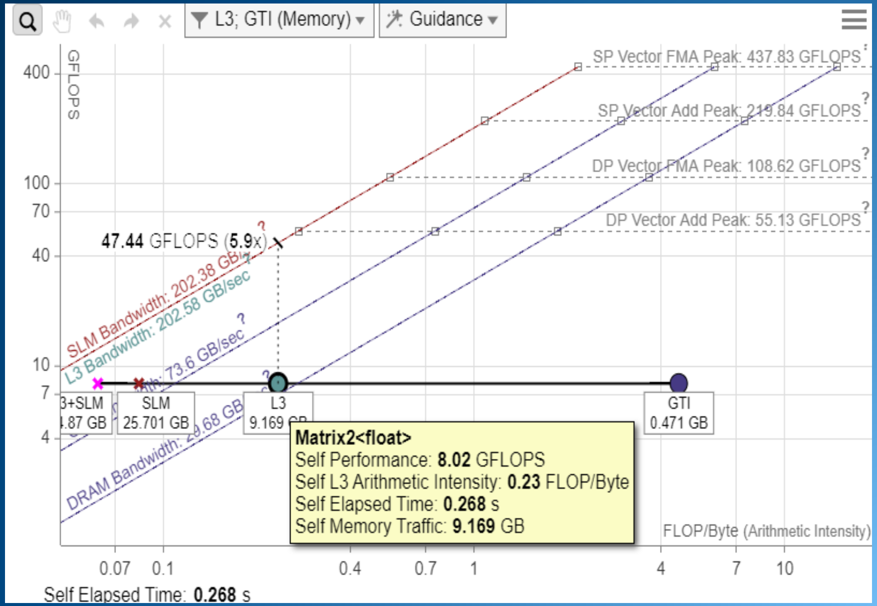
Thread Prototyping

- Add effective threading to unthreaded applications

Flow Graph Analyzer

- Create, visualize and analyze task and dependency computation graphs





Learn more: Intel.com/advisor

“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

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.

Get Breakthrough Vectorization Performance

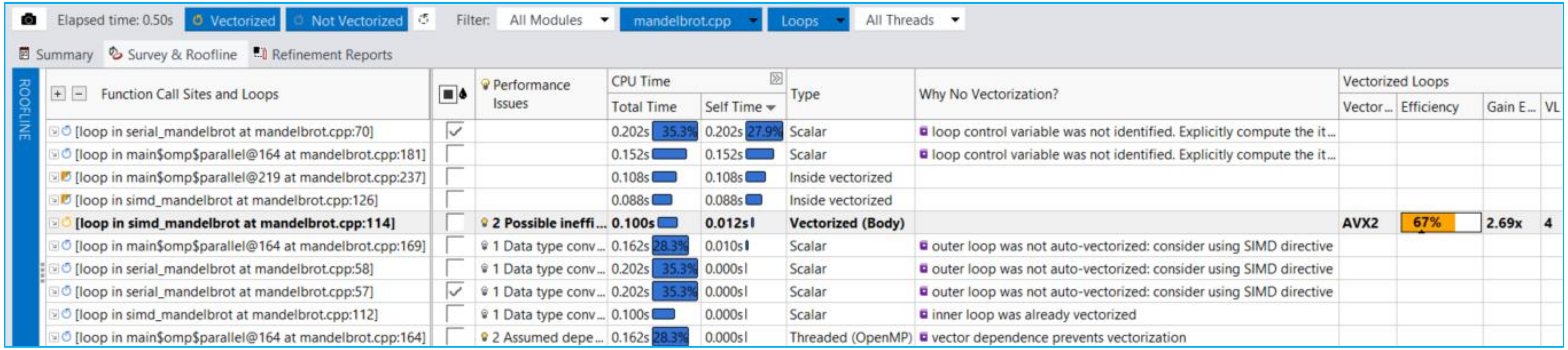
Intel® Advisor—Vectorization Advisor

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

Data & Guidance You Need

- Compiler diagnostics + Performance Data + SIMD efficiency
- Detect problems & recommend fixes
- Loop-Carried Dependency Analysis
- Memory Access Patterns Analysis



Elapsed time: 0.50s | Vectorized | Not Vectorized | Filter: All Modules | mandelbrot.cpp | Loops | All Threads

Summary | Survey & Roofline | Refinement Reports

Function Call Sites and Loops		Performance Issues	CPU Time: Total Time	CPU Time: Self Time	Type	Why No Vectorization?	Vectorized Loops: Vector...	Efficiency	Gain E...	VL
[loop in serial_mandelbrot at mandelbrot.cpp:70]	✓		0.202s 35.3%	0.202s 27.9%	Scalar	loop control variable was not identified. Explicitly compute the it...				
[loop in mainompparallel@164 at mandelbrot.cpp:181]			0.152s	0.152s	Scalar	loop control variable was not identified. Explicitly compute the it...				
[loop in mainompparallel@219 at mandelbrot.cpp:237]			0.108s	0.108s	Inside vectorized					
[loop in simd_mandelbrot at mandelbrot.cpp:126]			0.088s	0.088s	Inside vectorized					
[loop in simd_mandelbrot at mandelbrot.cpp:114]		**2 Possible ineffi...**	**0.100s**	**0.012s**	**Vectorized (Body)**		**AVX2**	**67%**	**2.69x**	**4**
[loop in mainompparallel@164 at mandelbrot.cpp:169]		1 Data type conv...	0.162s 28.3%	0.010s	Scalar	outer loop was not auto-vectorized: consider using SIMD directive				
[loop in serial_mandelbrot at mandelbrot.cpp:58]		1 Data type conv...	0.202s 35.3%	0.000s	Scalar	outer loop was not auto-vectorized: consider using SIMD directive				
[loop in serial_mandelbrot at mandelbrot.cpp:57]	✓	1 Data type conv...	0.202s 35.3%	0.000s	Scalar	outer loop was not auto-vectorized: consider using SIMD directive				
[loop in simd_mandelbrot at mandelbrot.cpp:112]		1 Data type conv...	0.100s	0.000s	Scalar	inner loop was already vectorized				
[loop in mainompparallel@164 at mandelbrot.cpp:164]		2 Assumed depe...	0.162s 28.3%	0.000s	Threaded (OpenMP)	vector dependence prevents vectorization				

Optimize for Intel® AVX-512 with or without access to AVX-512 hardware

Intel.com/advisor

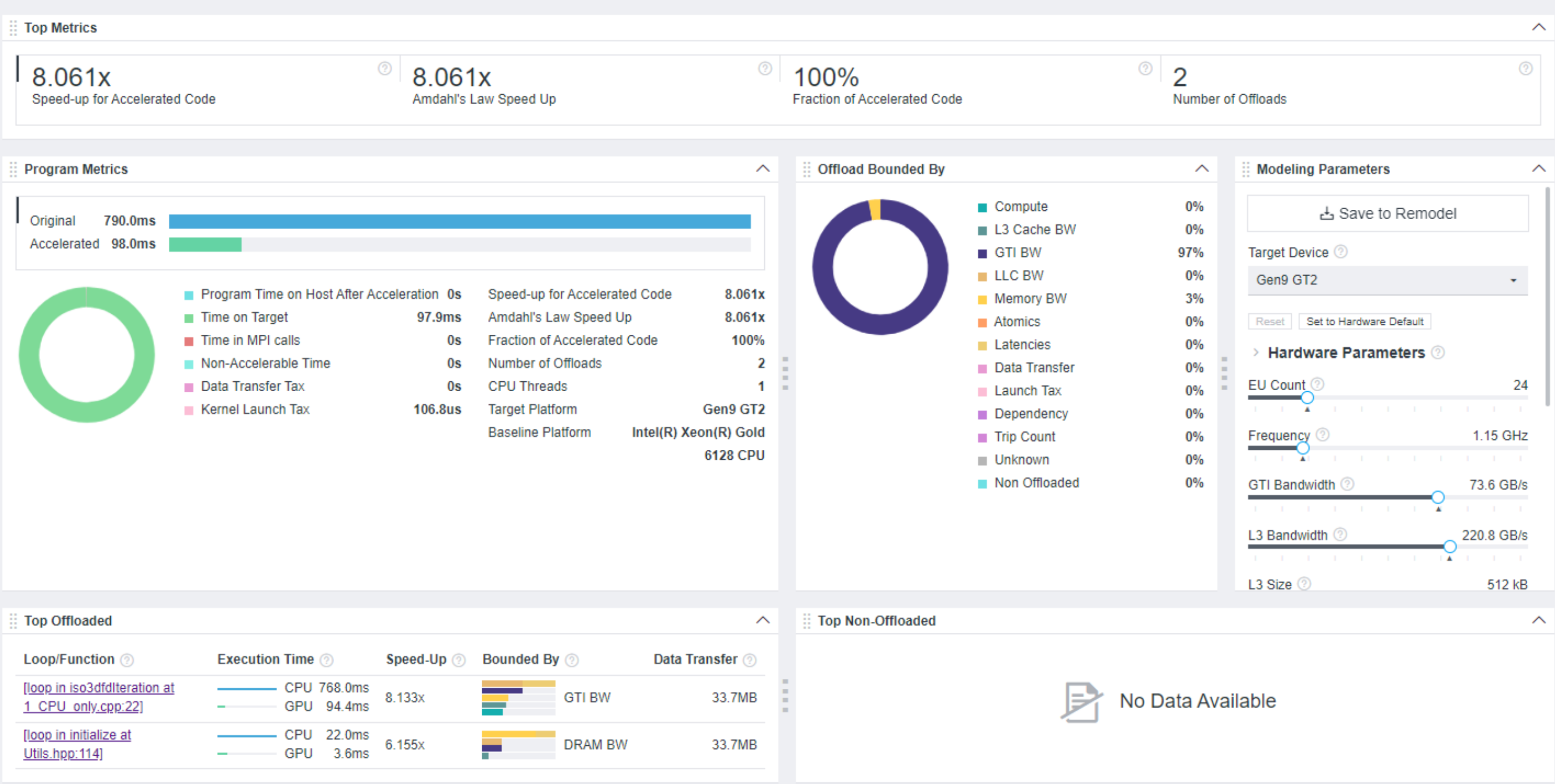
Design your code for efficient offload

Intel® Advisor - Offload Modeling

- Will your code benefit from GPU porting?
- How much performance acceleration will your code get from moving to the next-generation GPU?

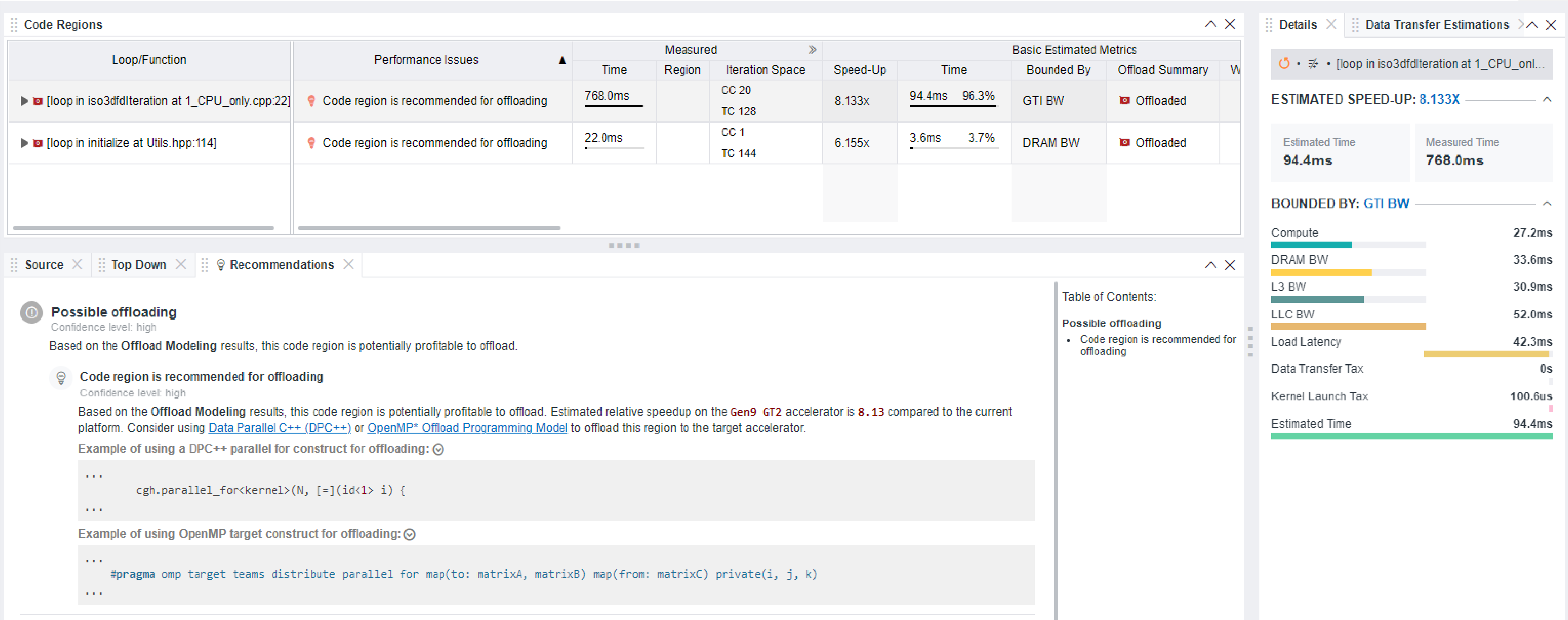
With Offload Modeling, you can:

- Pinpoint offload opportunities where it pays off the most.
- Project the performance on GPU.
- Identify bottlenecks and potential performance gains.
- Get guidance for optimizing a data transfer between host and target devices.



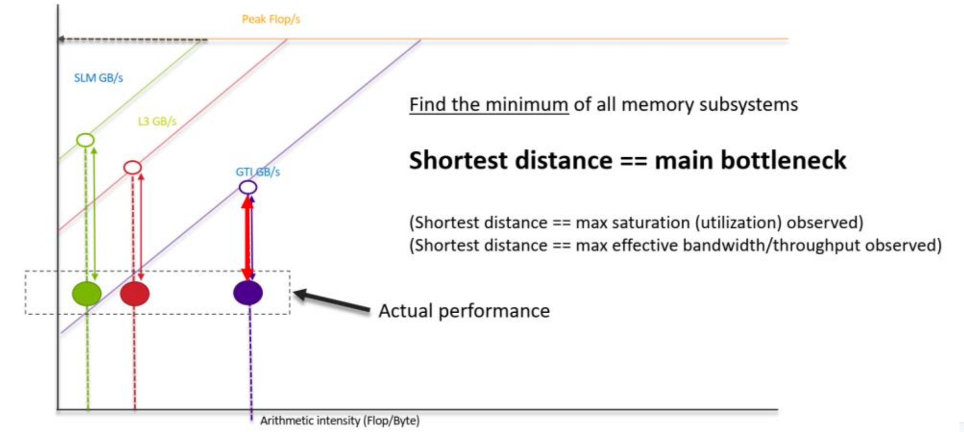
GPU Offload Modeling

Estimate the performance gain of offloading to the GPU



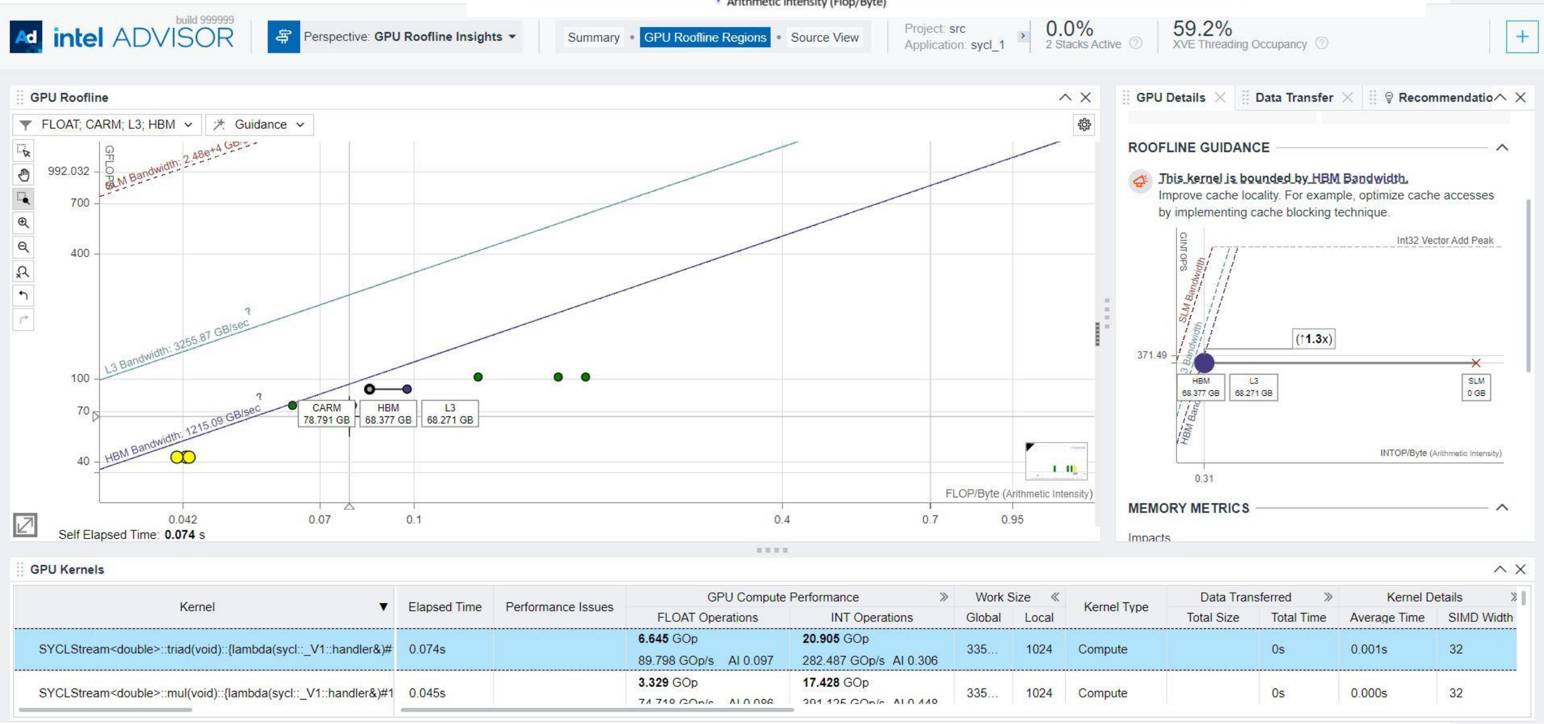
Find Effective Optimization Strategies

Intel® Advisor - Roofline Analysis on GPU

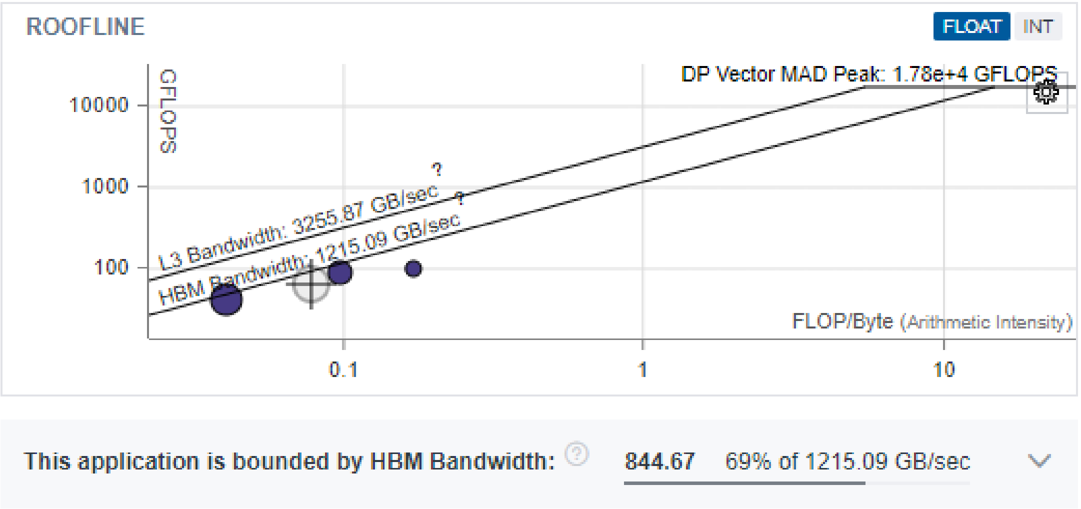


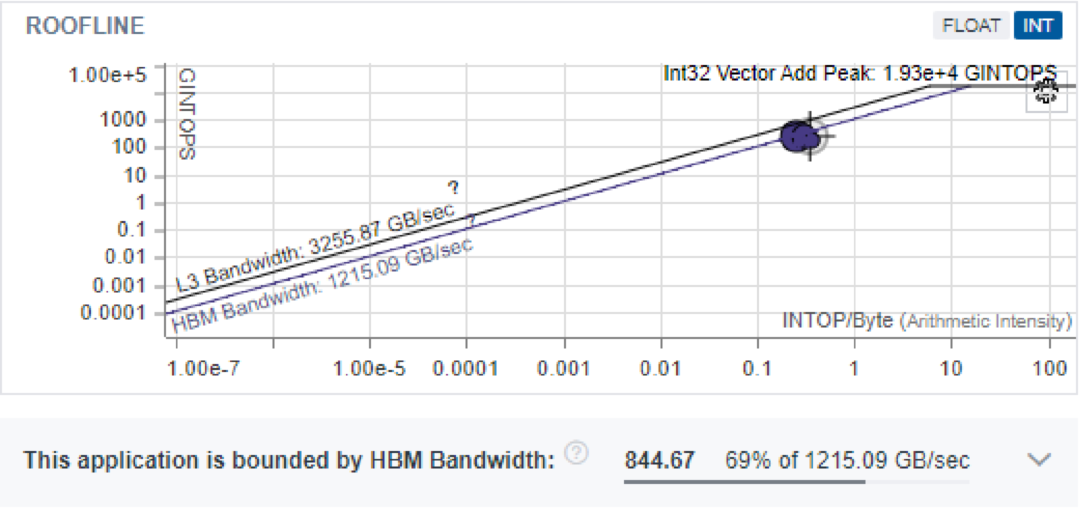
GPU Roofline Performance Insights

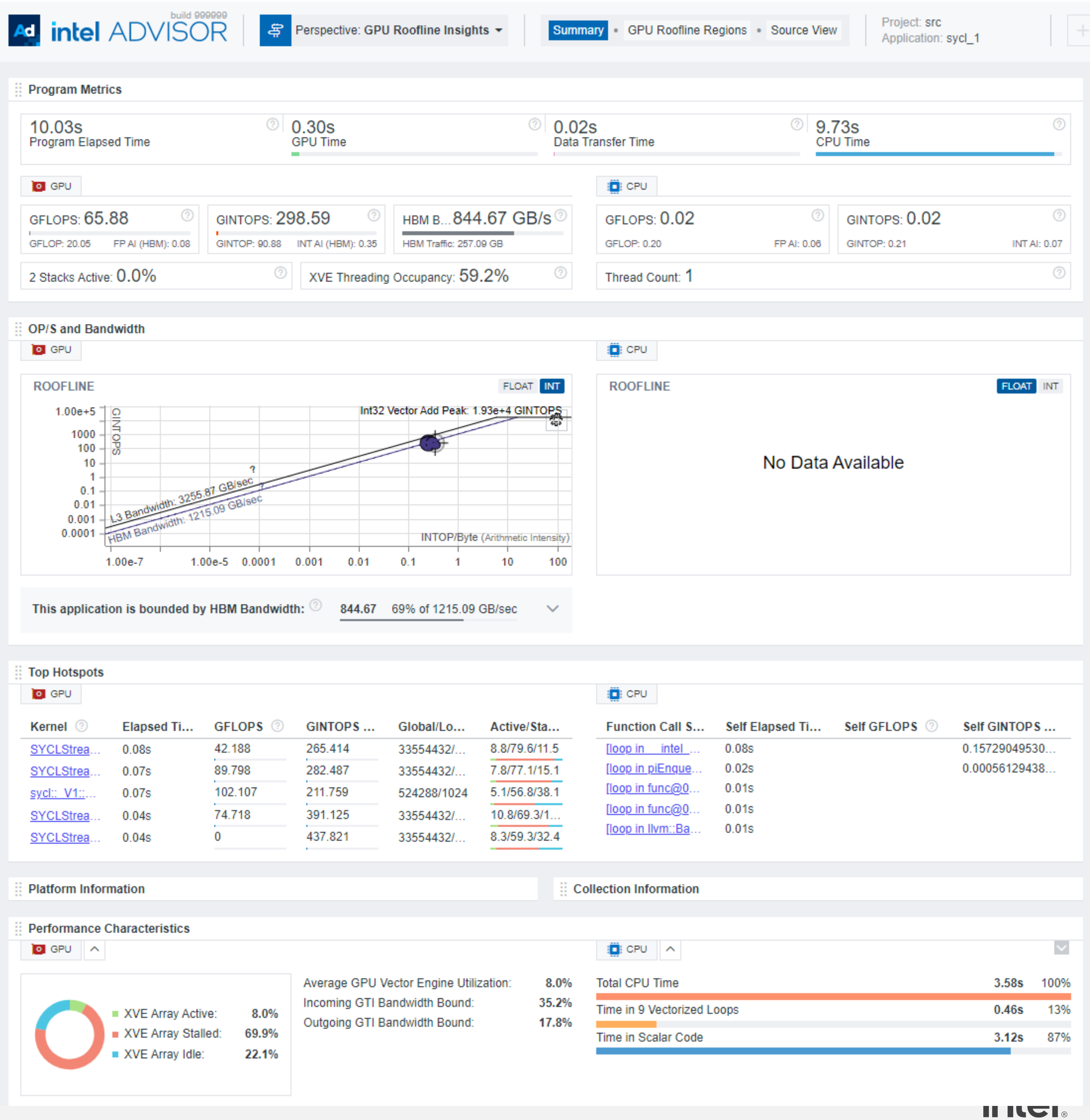
- Highlights poor performing kernels
- Shows performance 'headroom' for each kernels
  - Which can be improved
  - Which are worth improving
- Shows likely causes of bottlenecks
  - Memory bound vs. compute bound
- Suggests next optimization steps



GPU Roofline Analysis

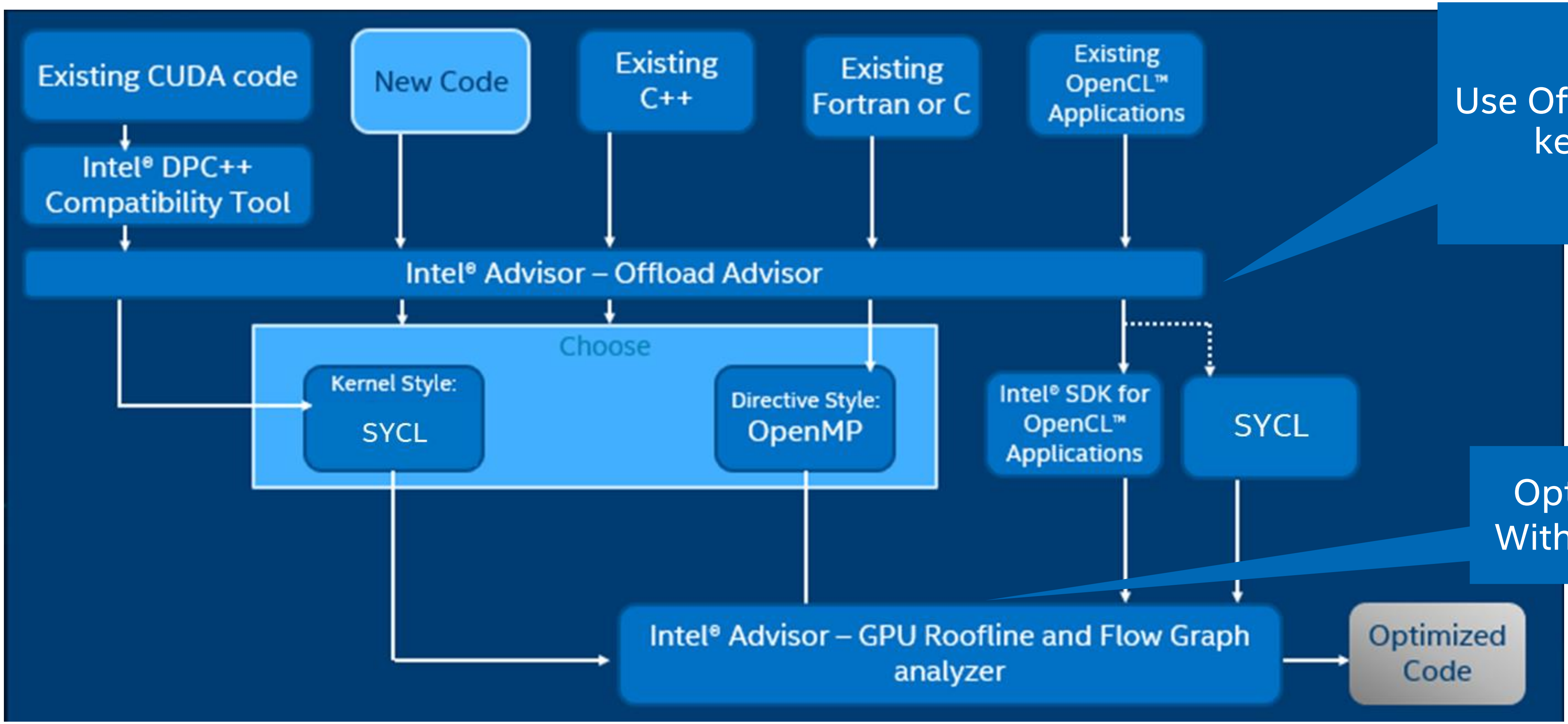






Recommended Workflow

Using Intel® Analyzers to increase performance



Use Offload Advisor to find kernels to offload

Optimize your kernels
With Advisor and VTune

More Resources

Intel® VTune™ Profiler – Performance Profiler

- Product page – overview, features, FAQs…
- Training materials – Cookbooks, User Guide, Processor Tuning Guides
- Support Forum
- Online Service Center - Secure Priority Support
- What's New?

Additional Analysis Tools

- Intel® Advisor – Design code for efficient vectorization, threading, memory usage, and accelerator offload
- Intel® Inspector – memory and thread checker/ debugger
- Intel® Trace Analyzer and Collector - MPI Analyzer and Profiler

Additional Development Products

- oneAPI: A new era of heterogenous computing

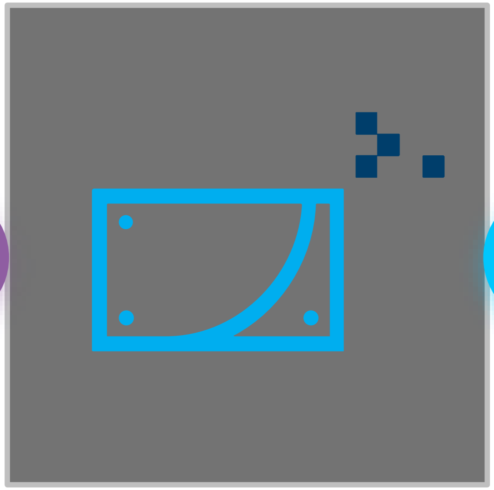
Performance Profiling Exercises

Iso3dfd Sample

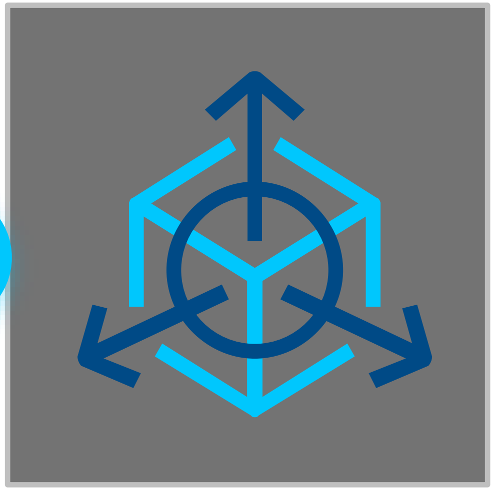
Workflow

Log into an Intel® **DevCloud** GPU node and configure the **MandelbrotOMP** sample

Run Intel Advisor: **Offload Advisor** to estimate performance on Gen9 GT2 GPU



Run Intel Advisor: **GPU Roofline** on offloaded implementation to visualize GPU performance



Run Intel VTune Profiler: **GPU Hotspots** for deeper insights into GPU kernels and device metrics

Basic steps

Log into DevCloud via ssh

Create sample and build the example

```
$ git clone https://github.com/oneapi-src/oneAPI-samples.git
```

Start an interactive gpu node:

```
$ qsub -I -l nodes=1:gpu:ppn=2 or
$ qsub -I -l nodes=1:gen9:ppn=2
```

Run the profling tools in commandline and view the results

DevCloud Setup

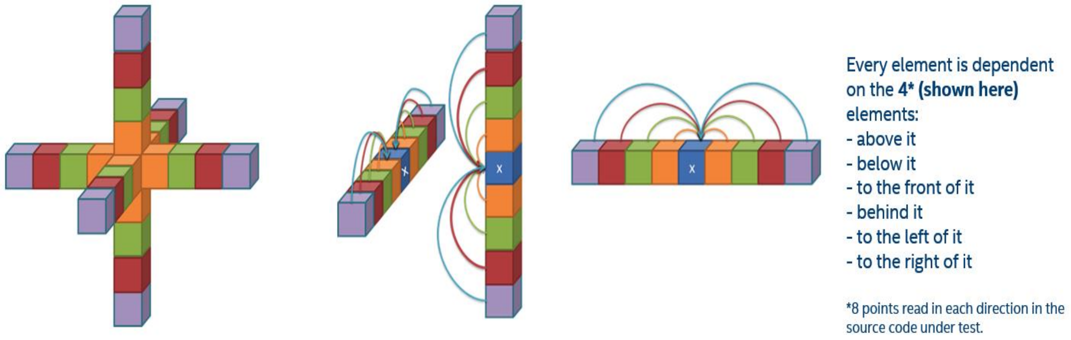
Intel DevCloud provides a free environment for testing the Intel CPUs and GPUs. Intel oneAPI toolkits are already installed and set up for use.

DevCloud Document:

https://devcloud.intel.com/oneapi/documentation/shell-commands/

Iso3dfd example

- The ISO3DFD sample refers to Three-Dimensional Finite-Difference Wave Propagation in Isotropic Media; it is a three-dimensional stencil to simulate a wave propagating in a 3D isotropic medium
- The sample provides a guided example to optimize code for GPU offload. https://github.com/oneapi-src/oneAPI-samples/tree/master/DirectProgramming/C%2B%2BSYCL/StructuredGrids/guided_iso3dfd_GPUOptimization
- Git repo: `git clone https://github.com/oneapi-src/oneAPI-samples.git`
  - oneAPI-samples -> DirectProgramming -> C++SYCL -> StructuredGrids -> guided_iso3dfd_GPUOptimization



Used 16th order 51pt stencil (But 8th order stencil shown here in figure)

Build the code

```
cd oneAPI-samples/DirectProgramming/C++SYCL/StructuredGrids/guided_iso3dfd_GPUOptimization

mkdir build

cd build

make ..

make
```

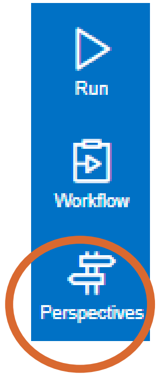
Intel® Advisor Exercise

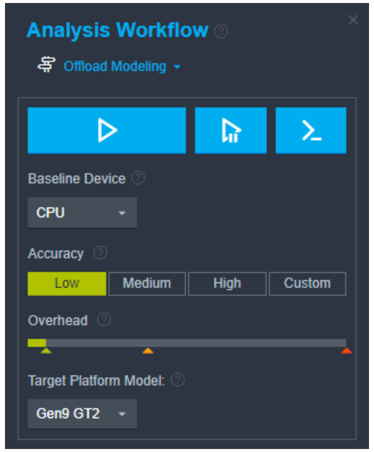
Offload modeling, GPU Roofline

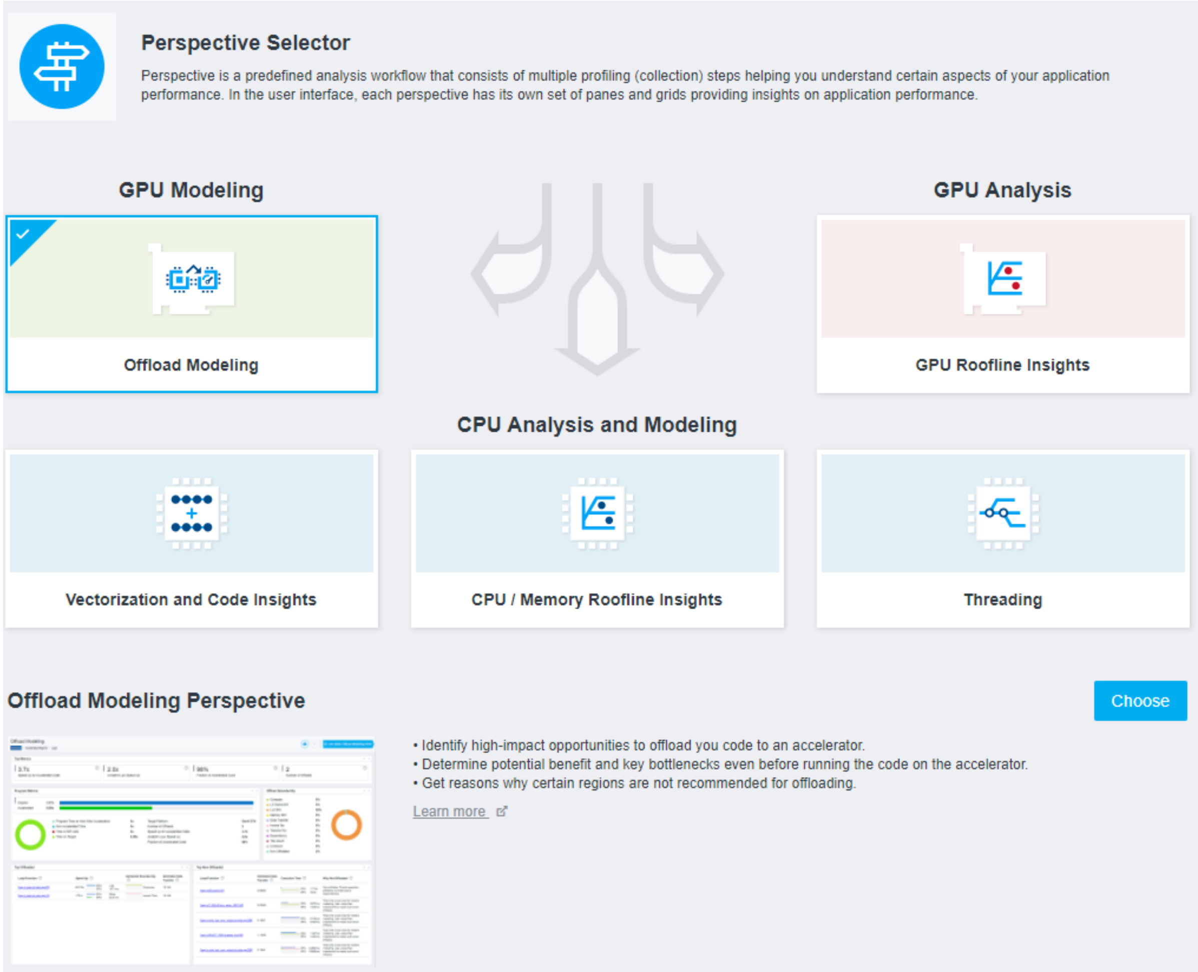
Collect Offload Advisor

Run from GUI - cont

1. Go back to the Perspective Selector and select Offload Modeling
2. Press Choose button
3. From the new Analysis Workflow panel:
    1. Select Low for Accuracy
    2. Select Gen9 GT2 from the Target Platform Model drop-down
    3. Press the Run button







Run Advisor in CLI

1. Run the offload collection:

```
$ advisor --collect=offload --config=gen9_gt2 --project-
dir=./../advisor/1_cpu -- ./src/1_CPU_only 128 128 128 20
```

Long running but more accurate performance modeling.

```
$ advisor --collect=offload --accuracy=high --config=gen9_gt2 -
-project-dir=./../advisor/1_cpu -- ./src/1_CPU_only 256 256 256
100
```

2. Package results for viewing on the local host:

```
$ advisor --snapshot --project-dir=./../advisor/1_cpu --pack --
cache-sources --cache-binaries -- ./offload_advisor_snapshot
```

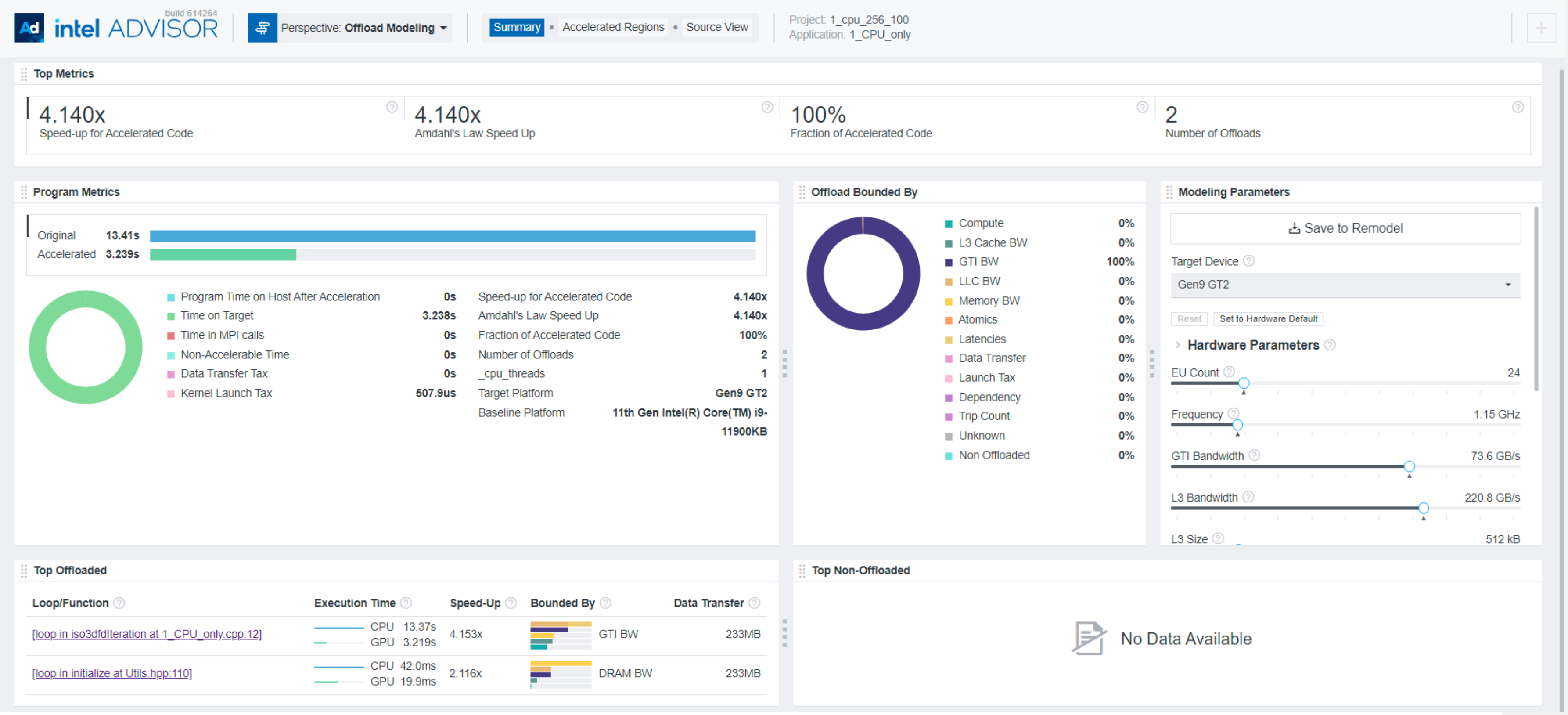
Collect Offload Advisor

- The Offload Modeling workflow includes the following analyses:

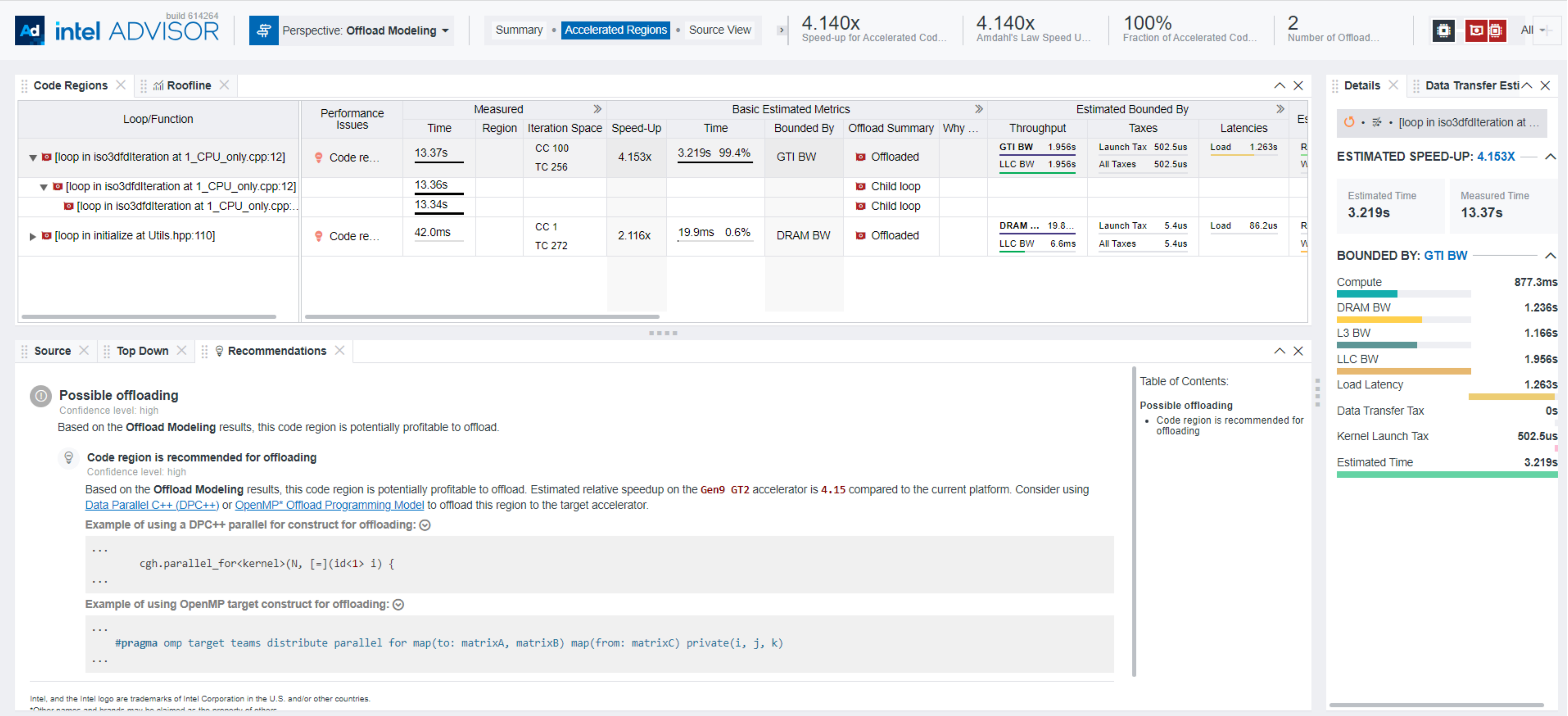
1. Survey to collect initial performance data.
2. Characterization with trip counts and FLOP to collect performance details.
3. Dependencies (optional) to identify loop-carried dependencies that might limit offloading.
4. Performance Modeling to model performance on a selected target device.

Collect Offload Advisor

- **Top Metrics** shows that the speed-up for accelerated code and Amdahl's Law are very close, indicating that the offloaded code makes up most of the workload. If accelerated code speed up is high but the Amdahl's law speed up is close to 1.000x, then offloading likely isn't worth it.
- **Program Metrics** contains more details about the accelerated code and how much program time will remain on the host.
- **Offload Bounded By** shows the items that may impact performance of the code once it is offloaded.
- **Modeling Parameters** are the hardware characteristics of the target device. Advisor provides configurations for many Intel GPUs.
- **Top Offloaded / Non-Offloaded** – these are loops or functions that have the potential to be offloaded. If the speed-up is significant enough, Advisor will recommend offloading. Some loops or functions will incur too much overhead to make offloading profitable.



Collect Offload Advisor



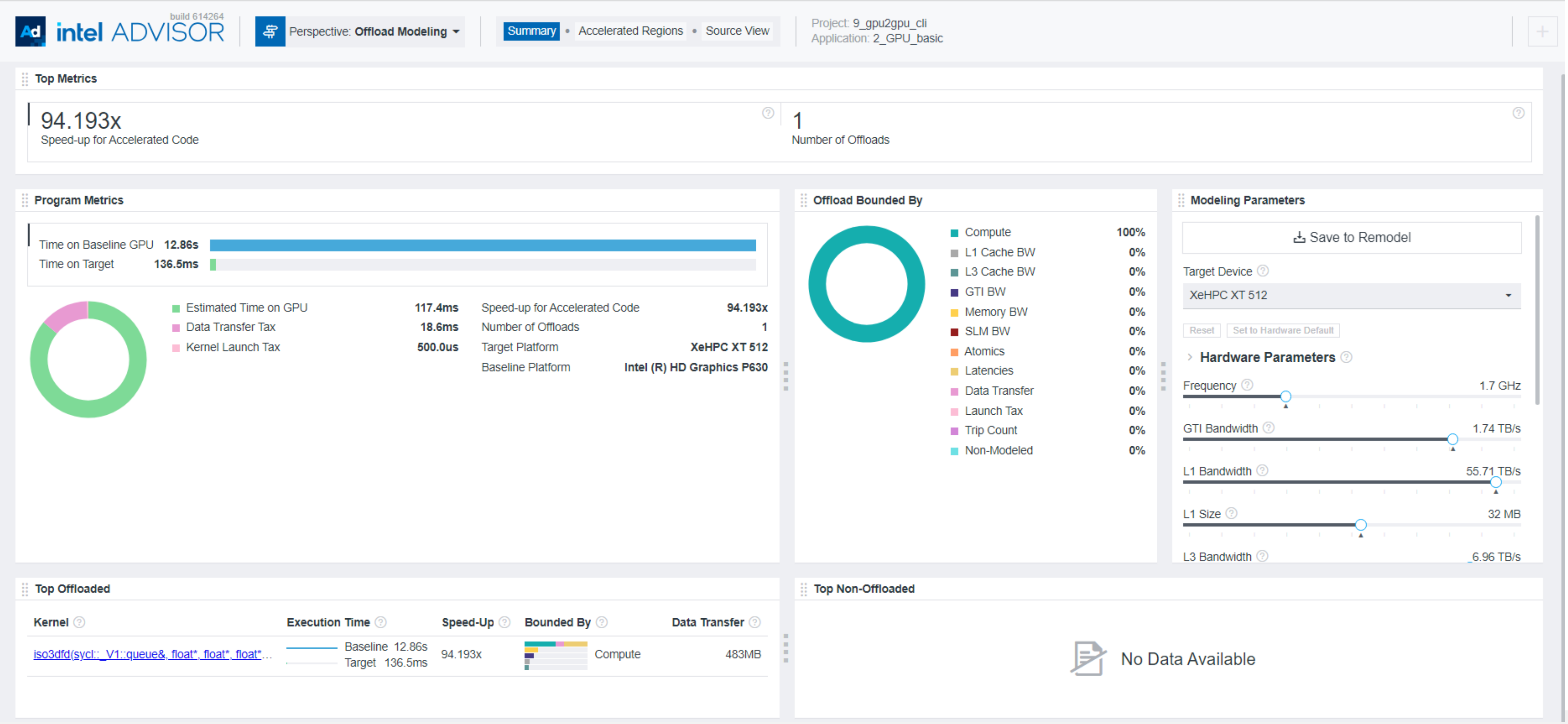
GPU-to-GPU performance modeling

Run the offload collection:

```
$ advisor --collect=offload --profile-gpu --target-device=pvc_xt_512xve --project-dir=./../advisor/gpu2gpu  -- ./src/2_GPU_basic 256 256 256 100

$ advisor-python $ APM/run_oa.py ./../advisor/gpu2gpu --gpu --config=pvc_xt_512xve -- ./src/2_GPU_basic 256 256 256 100

Or

$ advisor-python $APM/collect.py ./../advisor/gpu2gpu --gpu --config=pvc_xt_512xve -- ./src/2_GPU_basic 256 256 256 100

$ advisor-python $APM/analyze.py ./../advisor/gpu2gpu --gpu --config=pvc_xt_512xve
```

- GPU-to-GPU performance modeling is recommended to analyze SYCL, OpenMP target, and OpenCL application because it provides more accurate estimations. The
- GPU-to-GPU modeling analyzes only GPU compute kernels and ignores the application parts executed on a CPU.

GPU-to-GPU performance modeling



GPU Roofline

1st method: Run the shortcut command, simple

```
$ advisor --collect=roofline --profile-gpu --project-dir ./advi_results -- <app-with-parameters>
```

2nd method: Run the analyses separately, compatible with MPI, more flexible

```
$ advisor --collect=survey --profile-gpu --project-dir ./advi_results -- <app-with-parameters>

$ advisor --collect=tripcounts --flop --profile-gpu  -- project-dir ./advi_results -- <app-with-parameters>
```

- Add –target-gpu option on mutli-gpu systems

```
$ advisor --collect=roofline –-profile-gpu –-project-dir ./advi_results --target-gpu 0:77:0.0 -- <app-with-parameters>
```

GPU Roofline

View results in Intel® Advisor GUI or generate an HTML report

- HTML GPU Roofline chart

```
$ advisor --report roofline –gpu –-project-dir ./advisor_dir  --report-
output=./roofline.html
```

- interactive HTML report

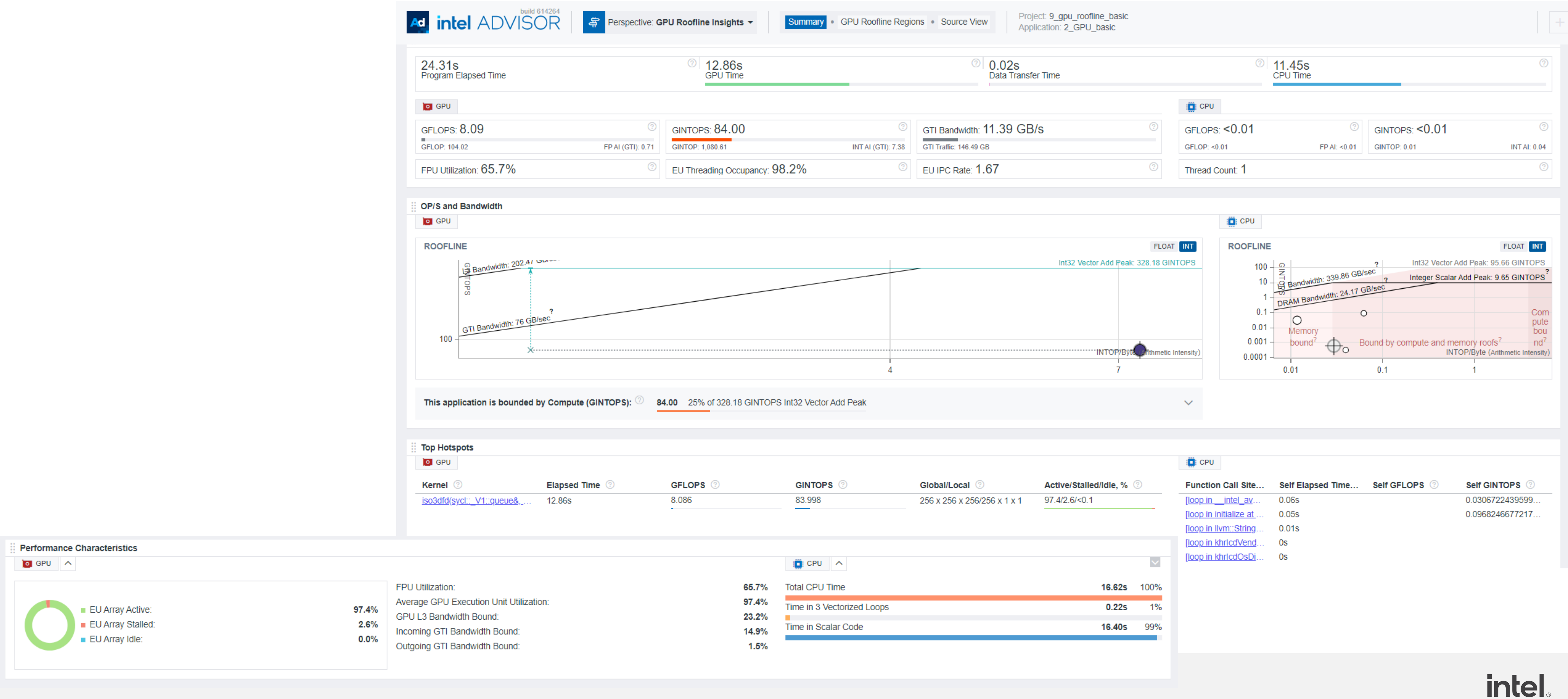
```
$ advisor --report all –-project-dir ./advisor_dir –report-
output=./roofline_report.html
```

- Create a snapshot for download to the local GUI:

```
$ advisor --snapshot --project-dir=./advisor_dir --pack --cache-sources
--cache-binaries -- ./adv_snapshot
```

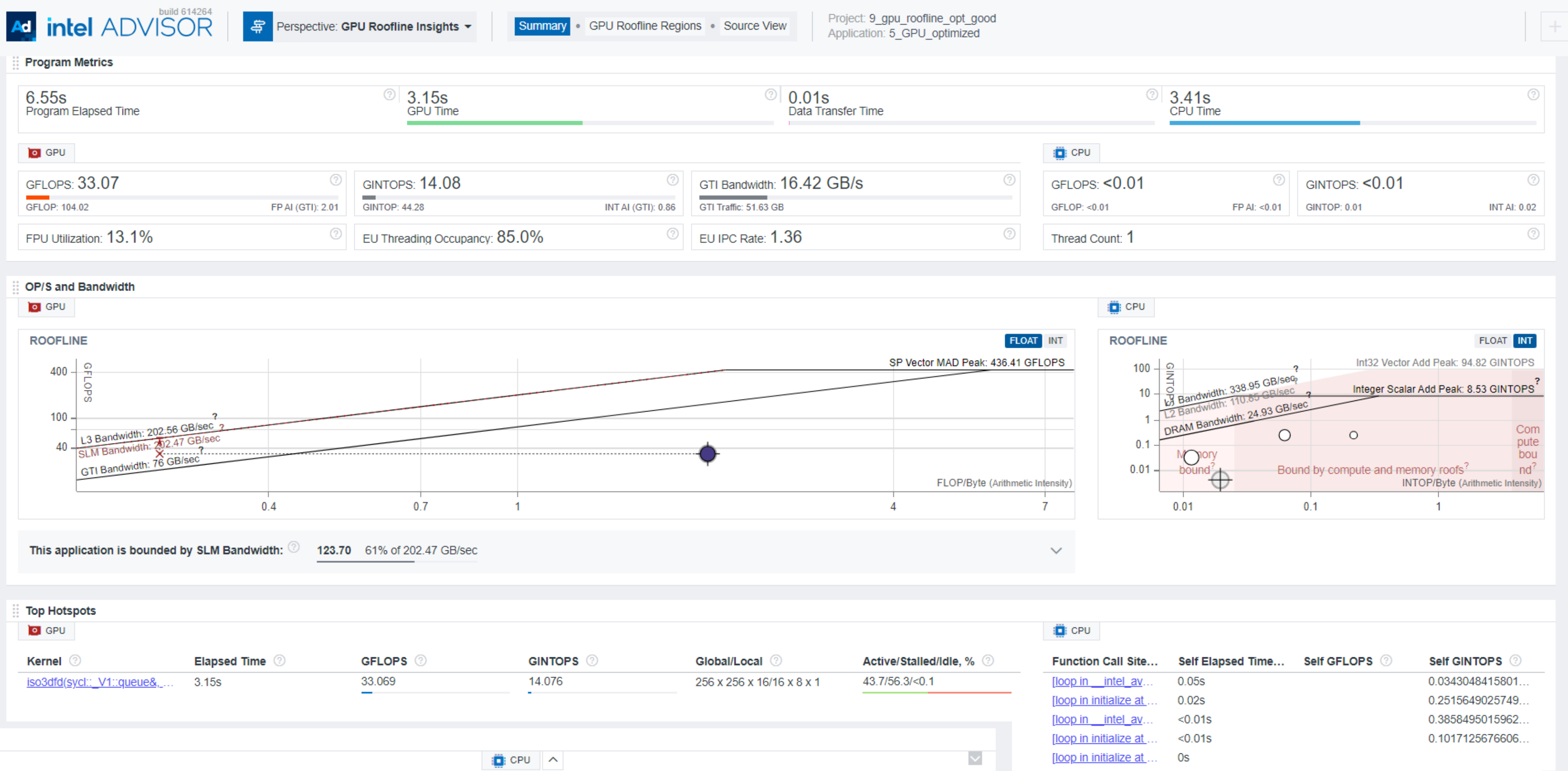
GPU Roofline

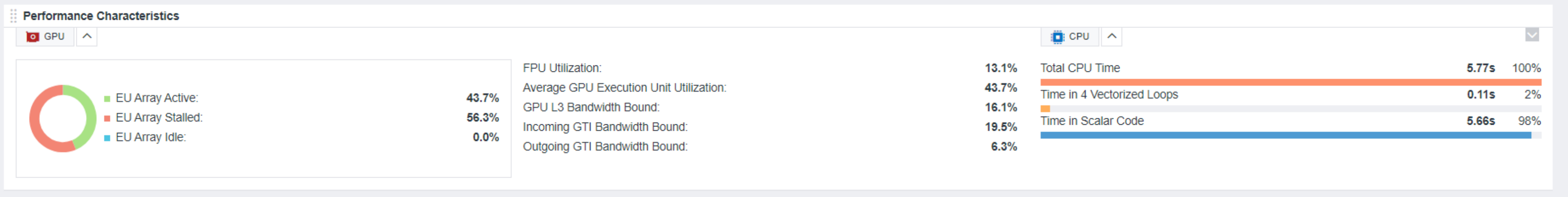
```
advisor --collect=roofline --profile-gpu  --project-dir=./../advisor/gpu_roofline_basic  -- ./src/2_GPU_basic 256 256 256 100
```



GPU Roofline

```
advisor --collect=roofline --profile-gpu  --project-dir=./../advisor/gpu_roofline_opt_good  -- ./src/5_GPU_optimized 256 256 256 100 16 8 16
```

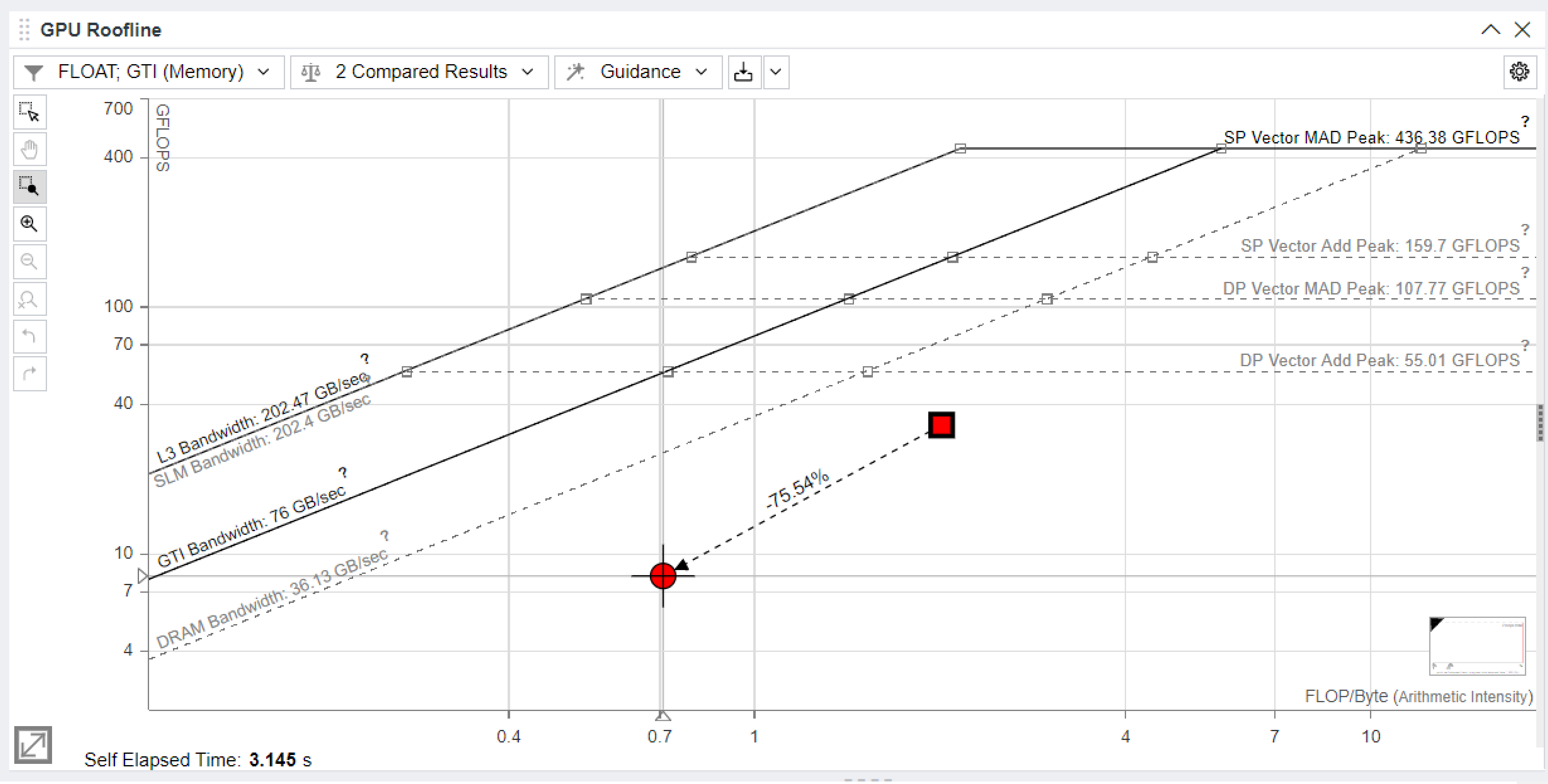




Compare Rooflines

- Available in the client!

https://www.intel.com/content/www/us/en/developer/articles/tool/oneapi-standalone-components.html#advisor



Intel® VTune™ Profiler Exercise

HPC-Performance, GPU Offload, GPU Hotspots

Intel® VTune™ Profiler Analysis Types

```
vtune
└── gpu-hotspots
    ├── profiling-mode
    │   ├── source-analysis
    │   │   └── source-analysis
    │   │       ├── mem-latency              - to explore memory access issues
    │   │       └── (bb-latency)             - identify stall reasons or basic blocks latency due to algorithm inefficiencies
    │   └── (characterization)
    │       └── characterization-mode
    │           ├── full-compute             - to combine Overview and Compute Basic event sets
    │           ├── global-local-accesses    - additional metrics that distinguish accessing different types of data on a GPU
    │           ├── instruction-count        - to analyze GPU instruction frequency per certain instruction types
    │           └── (overview)               - metrics to analyze sampler, general memory and cache accesses
    ├── collect-memory-bandwidth             - Collect the data required to compute memory and cross-socket bandwidth
    │   ├── true
    │   └── (false)
    ├── collect-programming-api              - Analyze DPC++, OpenCL, and Intel Media SDK programs
    │   ├── (true)
    │   └── false
    ├── analyze-power-usage                  - Enable energy consumption characterization and overtime tracking
    │   ├── true
    │   └── (false)
    └── computing-tasks-of-interest
        └── computing_task_name[#start_idx#step#stop_idx]
```

+ gpu-offload – To investigate CPU-GPU runtime analysis

+ Other experimental feature – Stall Sampling, Memory Access Analysis

Intel® VTune™ Profiler CLI

List all Vtune Analysis Types

- **vtune - -help collect**

List all the knobs for specific Analysis Type

- **vtune - -help collect gpu-hotspots**

HPC performance

- **vtune -c hpc-performance -r hpc_perf -- ./app**

GPU Offload

- **vtune -c gpu-offload -r gpu_go -- ./app**

GPU Hotspots

- **vtune -c gpu-hotspots -r gpu-hs -- ./app**

characterization with hotspots and instruction count

- **vtune -c gpu-hotspots -knob characterization-mode=instruction-count -r inst_cnt -- ./app**

source analysis with hotspots [with basic block latency - default]

- **vtune -c gpu-hotspots -knob profiling-mode=source-analysis -r src-analysis -- ./app**

source analysis with hotspots and memory latency

- **vtune -c gpu-hotspots -knob profiling-mode=source-analysis -knob source-analysis=mem-latency -r src-analysis_mem --./app**

Set up VTune server

1. Start an interactive job on DevCloud
   - ssh devcloud
   - qsub -I -l nodes=1:gpu:ppn=2

2. Run the vtune-backend command
   - vtune-backend --web-port=8080 --data-directory=$HOME/vtune_results

   A URL will be printed by the above the command

3. Set up local port forwarding
   - Open another terminal to launch additional SSH sessions to enable port forwarding:
     - ssh -L 8080:127.0.0.1:8080 devcloud
     - ssh –L 8080:127.0.0.1:8080 <compute-node from Step1>
   - Copy the URL printed by Step 2 and paste it in the local web browser

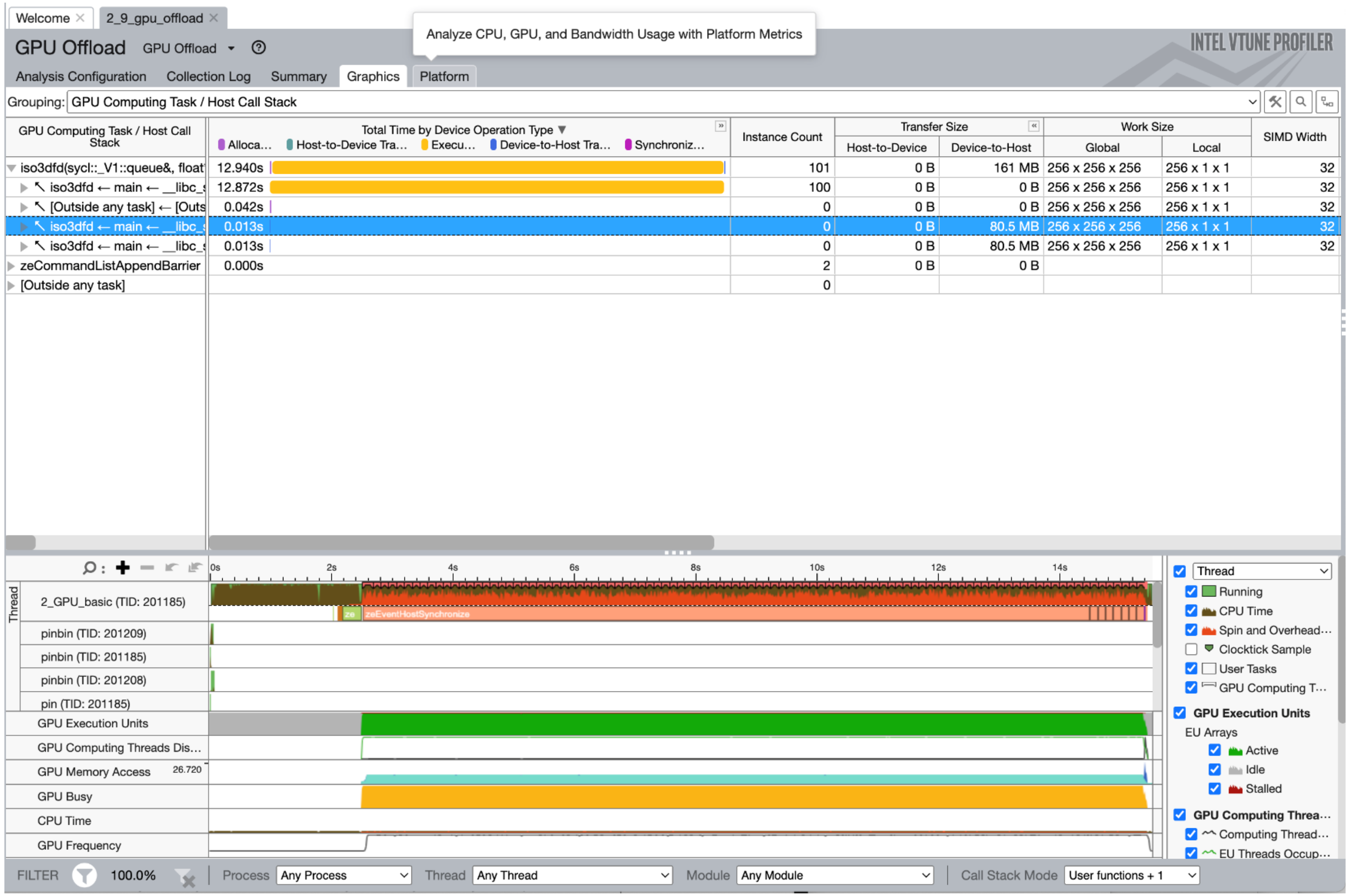
Case Study (SYCL Offload on GPU)

- Profile the baseline version with HPC-performance, GPU-offload, GPU-hotspots as well as
- Identify the related metrics correlated with the bottlenecks
- Profile the optimized version
- Compare baseline vs optimized

Intel® VTune™ Profiler

GPU Offload Analysis

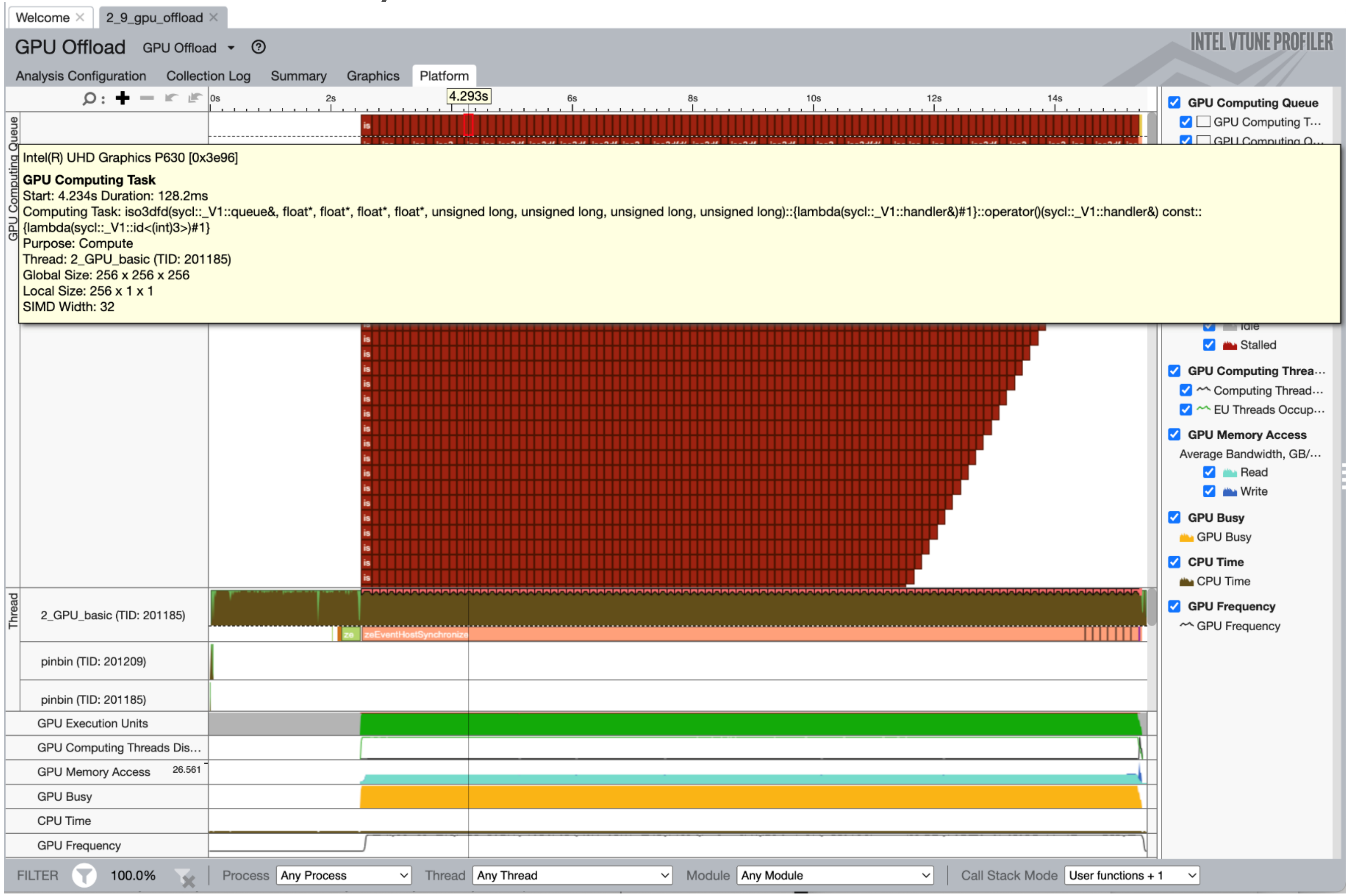
vtune -c gpu-offload -r gpu_go -- ./app



Intel® VTune™ Profiler

GPU Offload Analysis

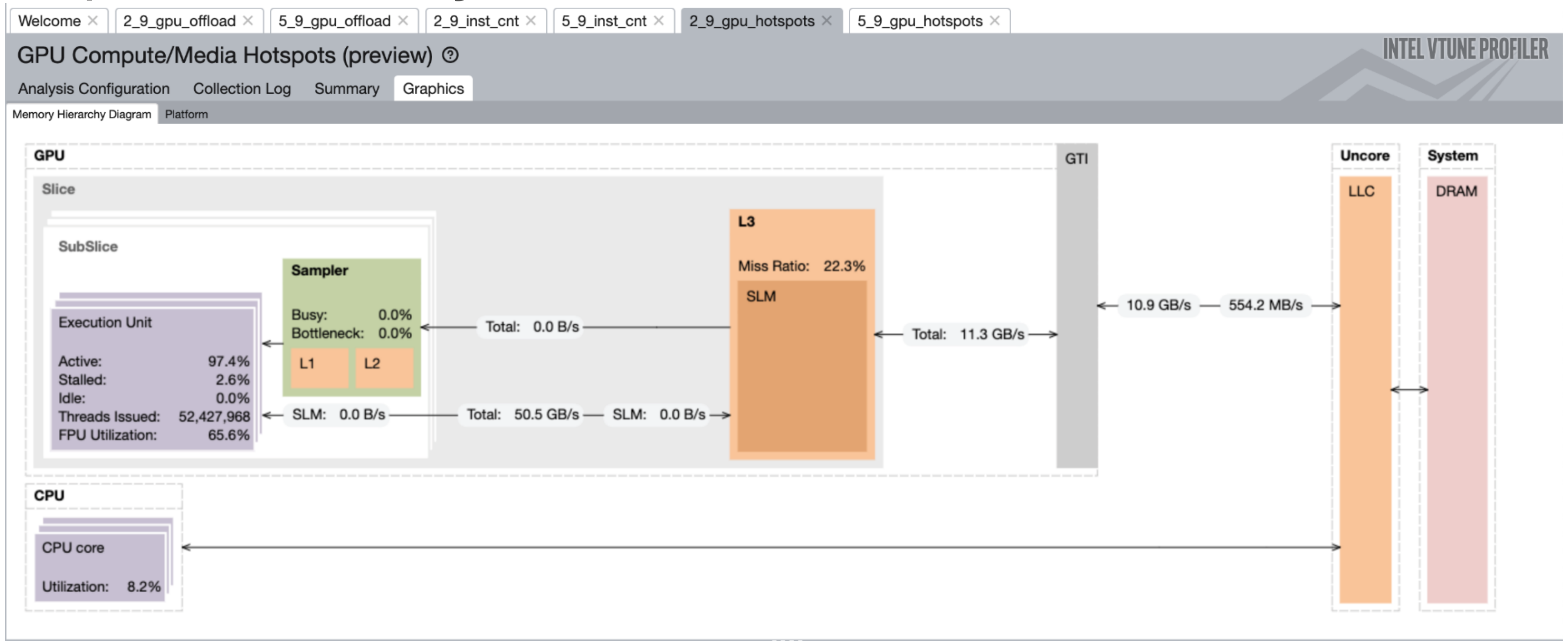
vtune -c gpu-offload -r gpu_go -- ./app



Intel® VTune™ Profiler

GPU Hotspots Analysis

vtune -c gpu-hotspots -r gpuhs -- ./app



Grouping: Computing Task

Computing Task		Computing Threads Started	L3 Bandwidth, GB/sec	L3 <-> GTI Total Bandwidth, GB/sec	Shared Local Me...		GPU Memory Band...		Sampler		GPU L3 Misses, Misses/sec	GPU Barriers	GPU Atomics
					Read	Write	Read	Write	Busy	Bottleneck			
iso3dfd(sycl::_V1::queue&, float*, float*, float*, flc	%	52,427,968	50.550	11.294	0.000	0.000	10.912	0.554	0.0%	0.0%	176475097.811	0	0
zeCommandListAppendMemoryCopyRegion	%	4,810,271	53.333	33.878	0.000	0.000	7.242	26.666	0.0%	0.0%	529351496.332	0	0
zeCommandListAppendBarrier	%	0	0.000	0.000	0.000	0.000	0.000	0.000	0.0%	0.0%	0.000	0	0
[Outside any task]	%	221,473	0.027	0.015	0.000	0.000	0.004	0.012	0.0%	0.0%	241101.471	0	0

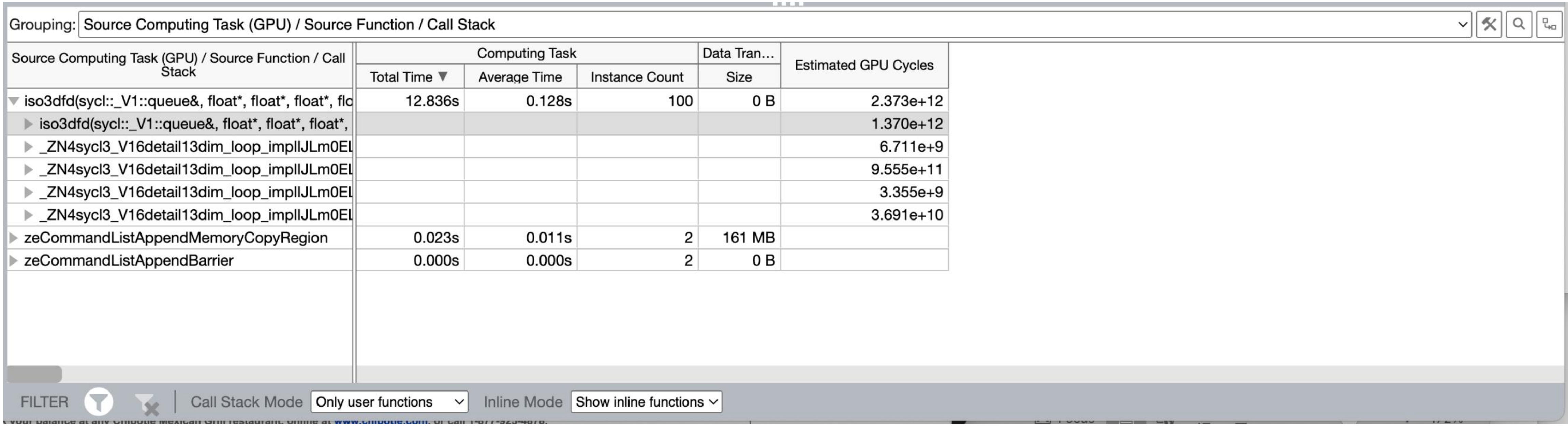
FILTER

Intel® VTune™ Profiler

GPU Hotspots Analysis

Source level in-kernel profiling

vtune -c gpu-hotspots -knob profiling-mode=source-analysis -r src-analysis -- ./app



Grouping: Source Computing Task (GPU) / Source Function / Call Stack

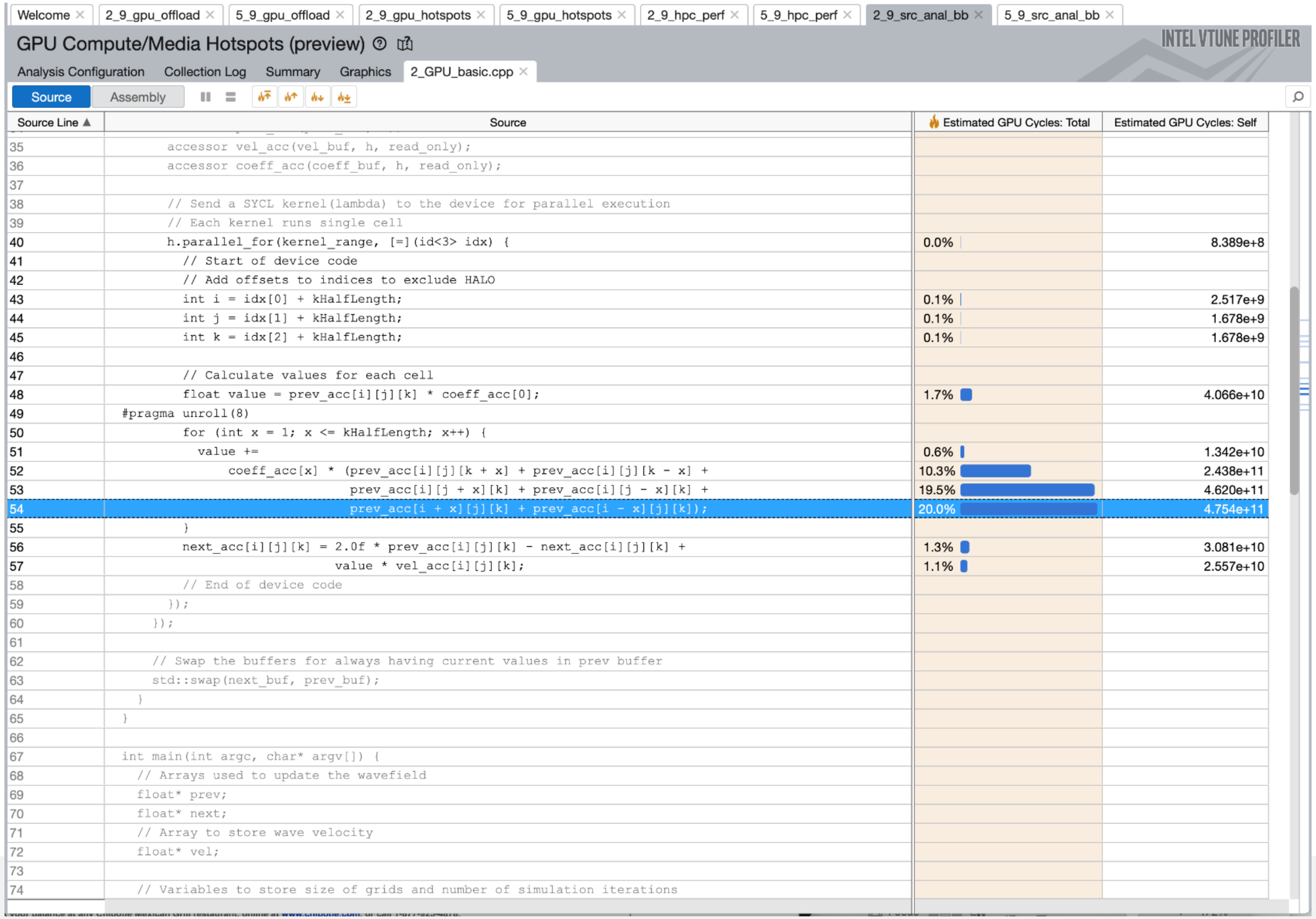
Source Computing Task (GPU) / Source Function / Call Stack	Computing Task			Data Tran…	Estimated GPU Cycles
	Total Time ▼	Average Time	Instance Count	Size	
▼ iso3dfd(sycl::_V1::queue&, float*, float*, float*, flo	12.836s	0.128s	100	0 B	2.373e+12
▶ iso3dfd(sycl::_V1::queue&, float*, float*, float*,					1.370e+12
▶ _ZN4sycl3_V16detail13dim_loop_implIJLm0EL					6.711e+9
▶ _ZN4sycl3_V16detail13dim_loop_implIJLm0EL					9.555e+11
▶ _ZN4sycl3_V16detail13dim_loop_implIJLm0EL					3.355e+9
▶ _ZN4sycl3_V16detail13dim_loop_implIJLm0EL					3.691e+10
▶ zeCommandListAppendMemoryCopyRegion	0.023s	0.011s	2	161 MB	
▶ zeCommandListAppendBarrier	0.000s	0.000s	2	0 B	

FILTER | Call Stack Mode: Only user functions | Inline Mode: Show inline functions

Intel® VTune™ Profiler

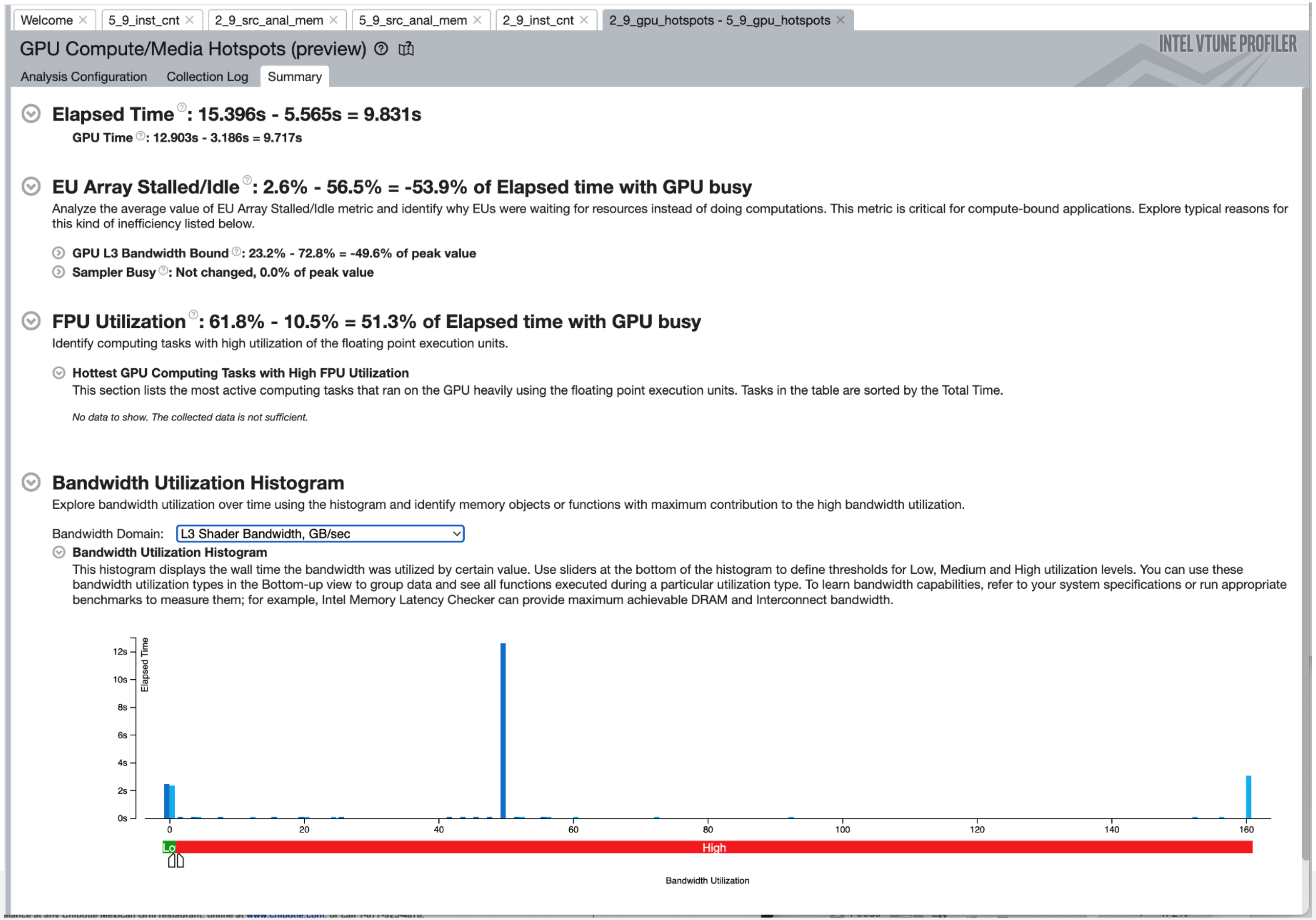
Source level in-kernel profiling

vtune -c gpu-hotspots -knob profiling-mode=source-analysis -r src-analysis -- ./app



Intel® VTune™ Profiler

Comparisons of two profiles



More Resources

Intel® VTune™ Profiler – Performance Profiler

- Product page – overview, features, FAQs…
- Training materials – Cookbooks, User Guide, Processor Tuning Guides
- Support Forum
- Online Service Center - Secure Priority Support
- What's New?

Additional Analysis Tools

- Intel® Advisor – Design code for efficient vectorization, threading, memory usage, and accelerator offload
- Intel® Inspector – memory and thread checker/ debugger
- Intel® Trace Analyzer and Collector - MPI Analyzer and Profiler

Additional Development Products

- oneAPI: A new era of heterogenous computing

How to get

- As part of the oneAPI Base Toolkit:
  - https://software.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit/download.html
- Standalone component:
  - https://software.intel.com/content/www/us/en/develop/articles/oneapi-standalone-components.html
- Linux:
  - Package managers:
    - https://software.intel.com/content/www/us/en/develop/articles/oneapi-repo-instructions.html
  - Containers:
    - https://github.com/intel/oneapi-containers