

Code that Outperforms

Intel[®] oneAPI Analyzers

Intel VTune Profiler and Intel Advisor

Xiao Zhu

xiao.zhu@intel.com

The Intel logo, consisting of the word "intel" in a lowercase, sans-serif font with a registered trademark symbol (®) to its upper right. The logo is positioned in the bottom left corner of the slide, partially overlapping a decorative graphic of blue squares of varying sizes and shades.

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



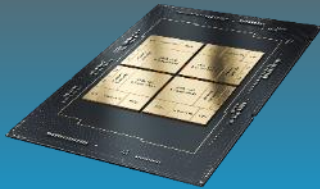
All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Modern Applications Demand Diverse Architectures

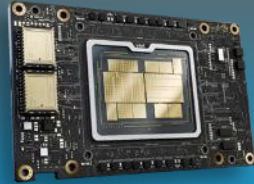
Diverse accelerators needed to meet today's performance requirements:

48% of developers target heterogeneous systems that use more than one kind of processor or core¹

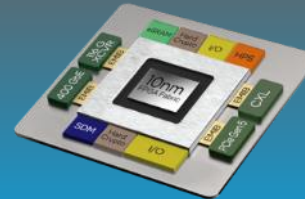
CPU



GPU



FPGA



Other Accelerators



Developer Challenges: Multiple Architectures, Vendors, and Programming Models



Open, Standards-based, Multiarchitecture Programming

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

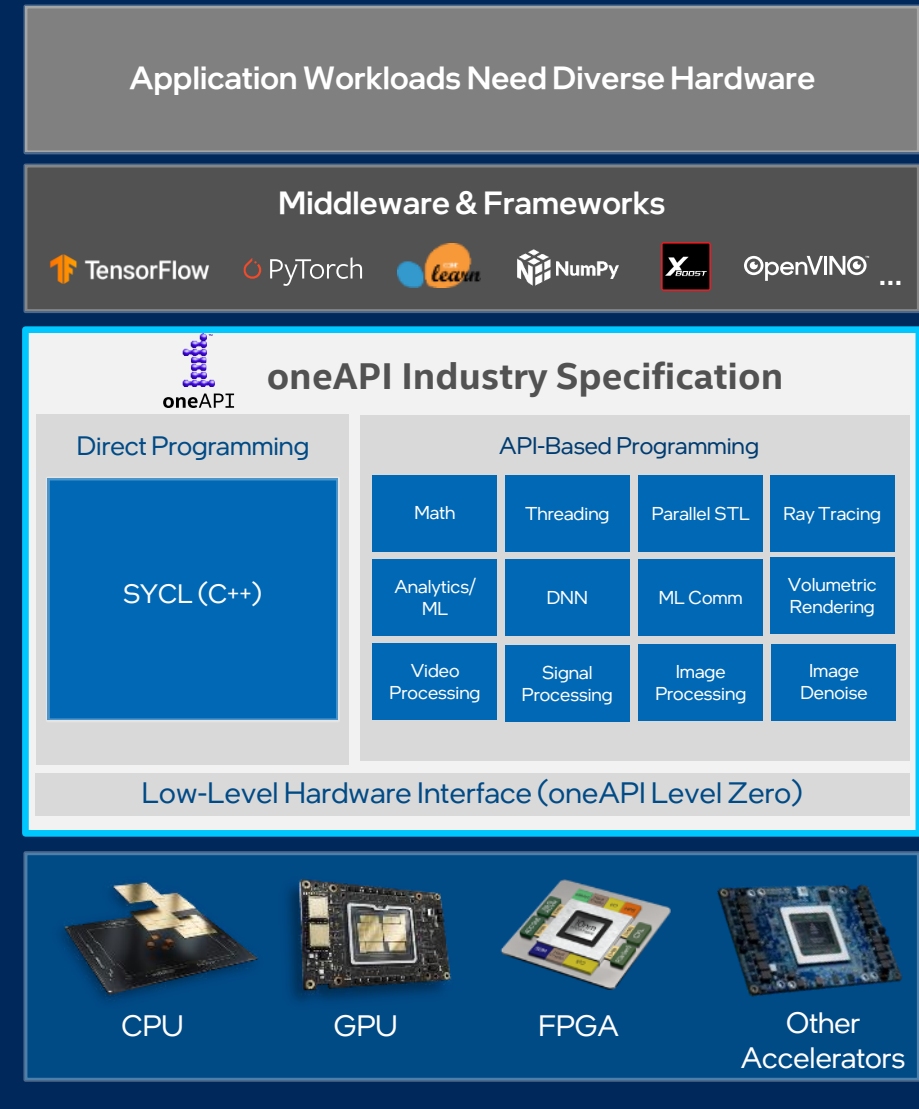
- Performance across CPU, GPUs, FPGAs, and other accelerators
- Expose and exploit cutting-edge features of the latest hardware

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



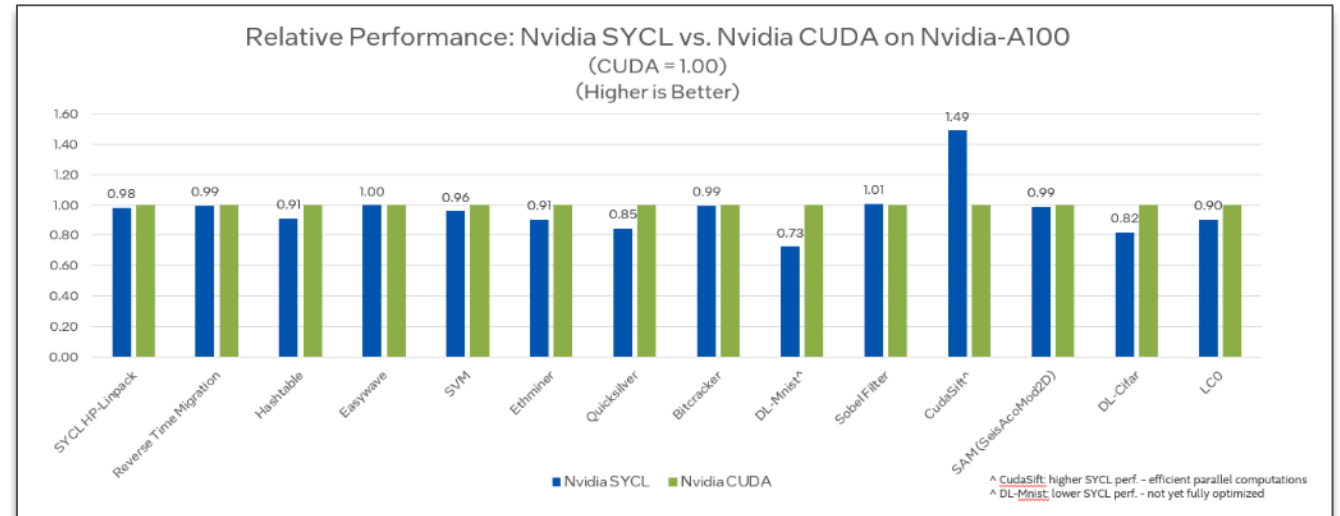
The productive, smart path to freedom for accelerated computing from the economic and technical burdens of proprietary programming models



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: -fscycl-targets=nvptx64-nvidia-cuda, NVIDIA NVCC compiler switches: -O3 -gencode arch=compute_80, code=sm_80. Represented workloads with Intel optimizations.

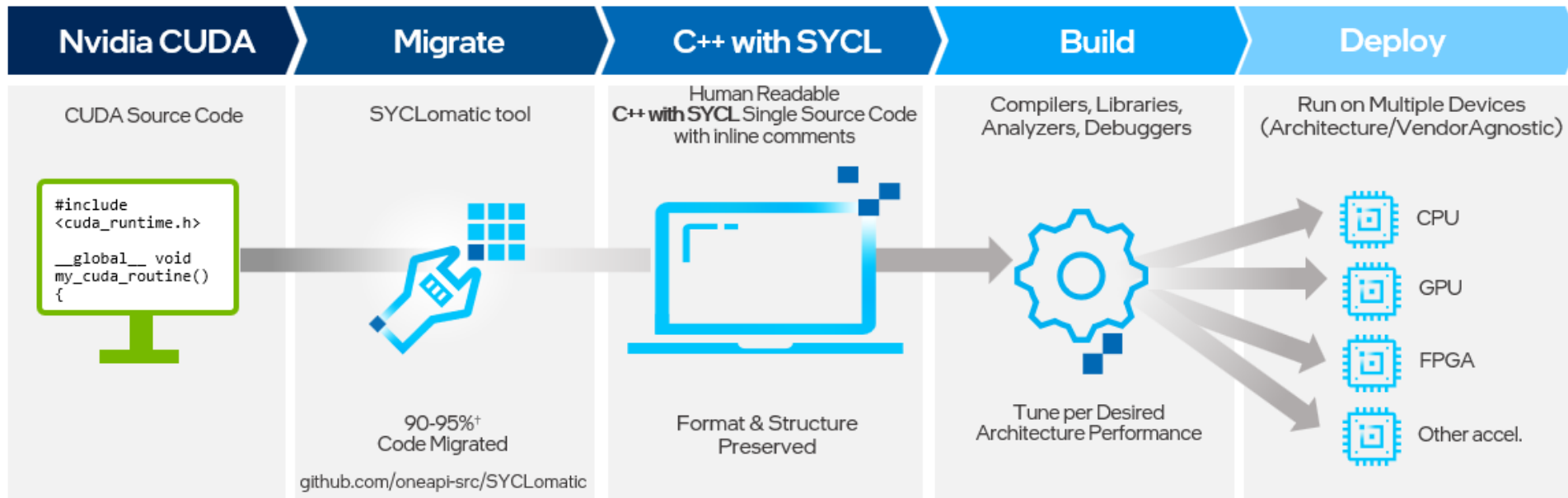
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

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 automatically¹

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

¹Intel 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.

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

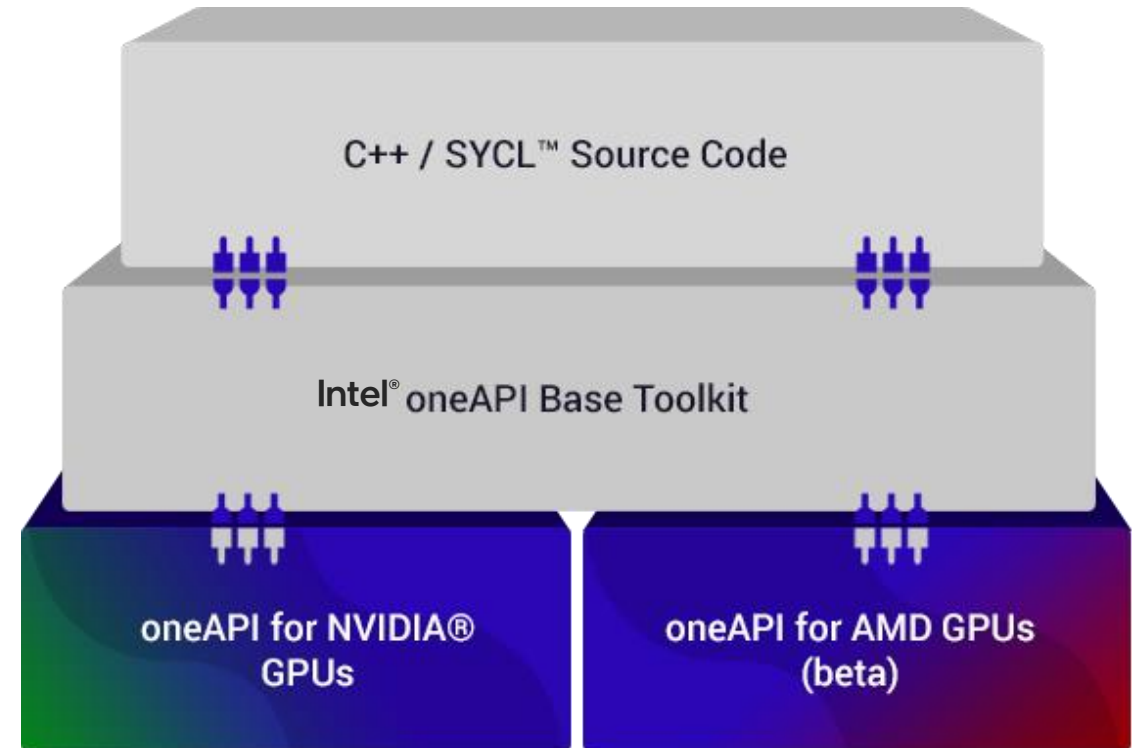


Image courtesy of Codeplay Software Ltd.

[Nvidia GPU plug-in](#)

[AMD GPU plug-in](#)

[Codeplay blog](#)

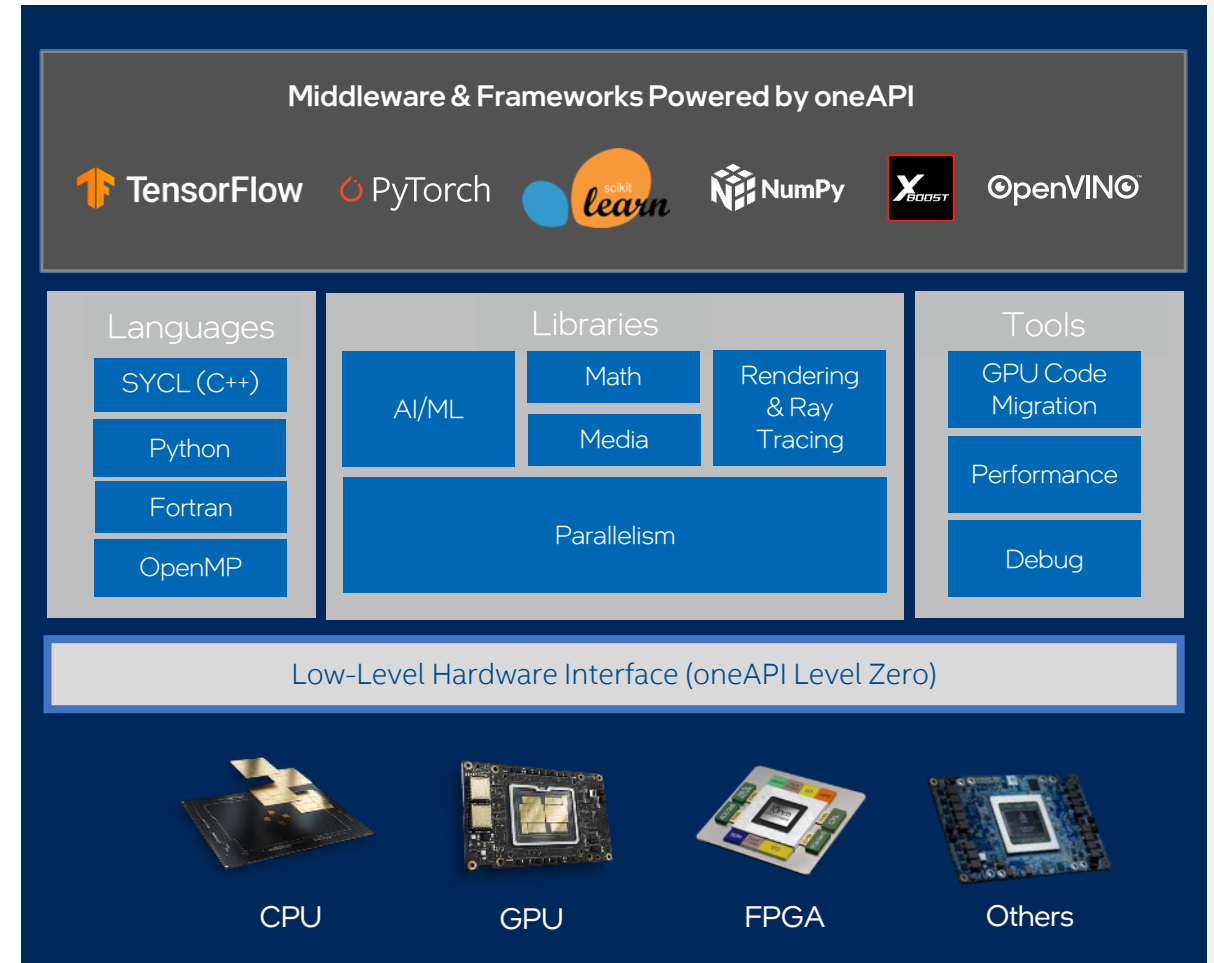
[Codeplay press release](#)

Intel® Developer Tools Supporting oneAPI

A complete set of proven tools expanded from CPU to accelerators



- 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



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			

Intel® VTune™ Profiler Overview

Optimize Performance

Intel® VTune™ Profiler

Get the Right Data to Find Bottlenecks

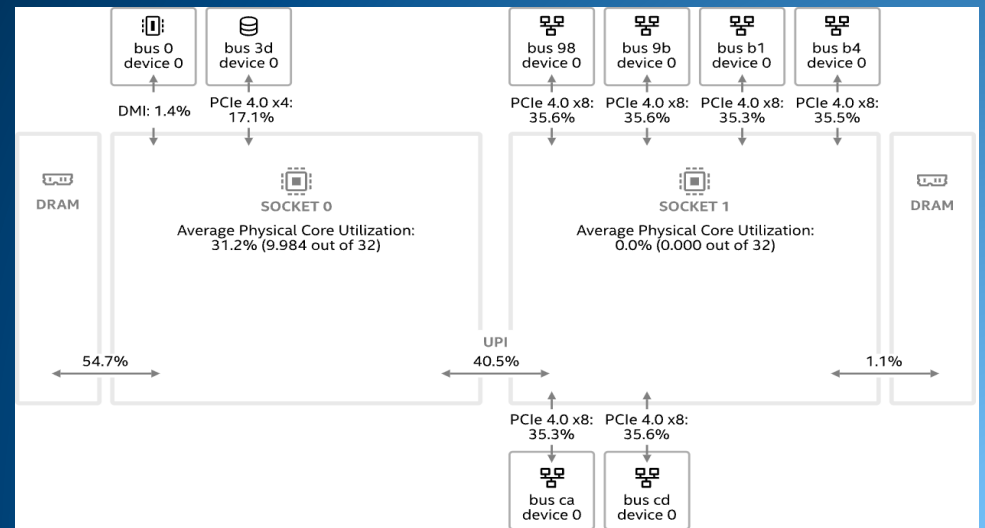
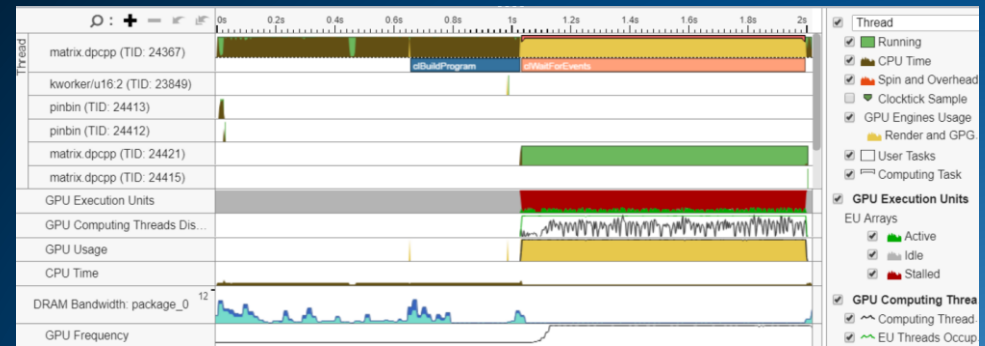
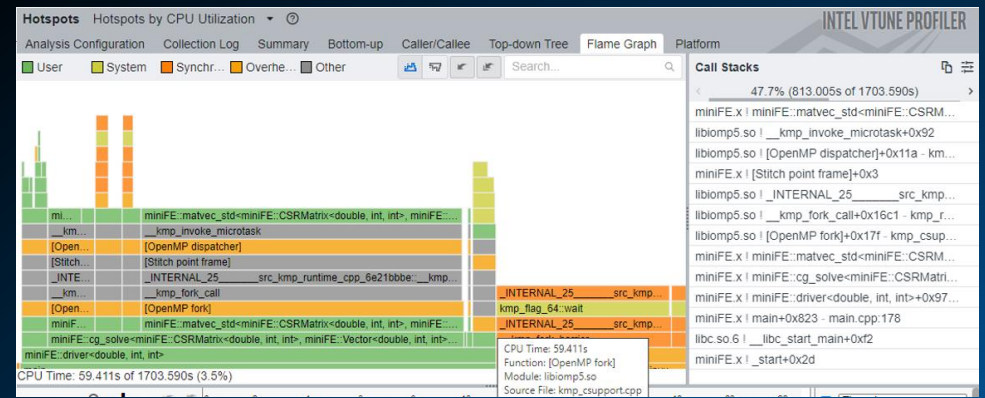
- A suite of profiling for CPU, GPU, FPGA, threading, memory, cache, storage, offload, power...
- Application or system-wide analysis
- DPC++, C, C++, Fortran, Python*, Go*, Java*, or a mix
- Linux, Windows, FreeBSD, Android, Yocto and more
- Containers and VMs

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

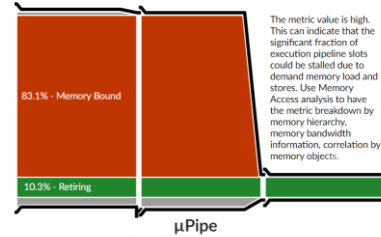
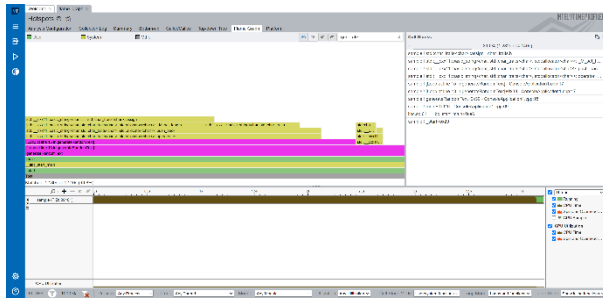
Work Your Way

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



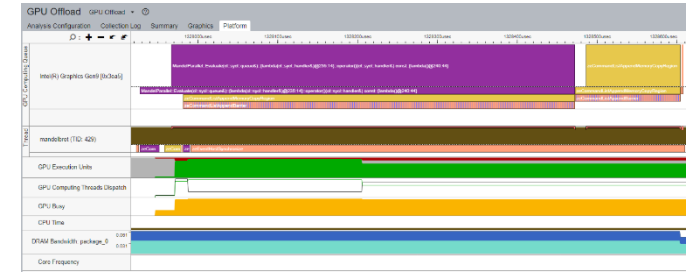
Rich Set of Profiling Capabilities

Intel® VTune™ Profiler



This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.

The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores. Use Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by memory objects.



Algorithm Optimization

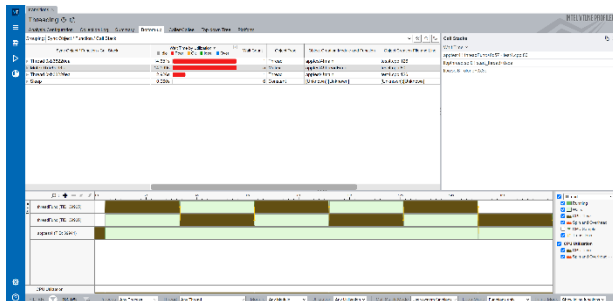
- ✓ Hotspots
- ✓ Anomaly Detection
- ✓ Memory Consumption

Microarch.&Memory Bottlenecks

- ✓ Microarchitecture Exploration
- ✓ Memory Access

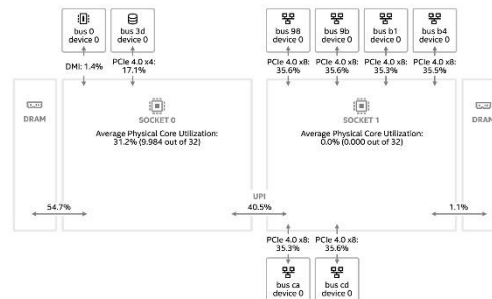
Accelerators / xPU

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



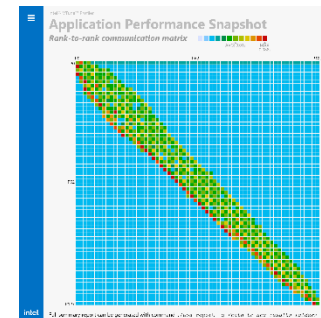
Parallelism

- ✓ 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

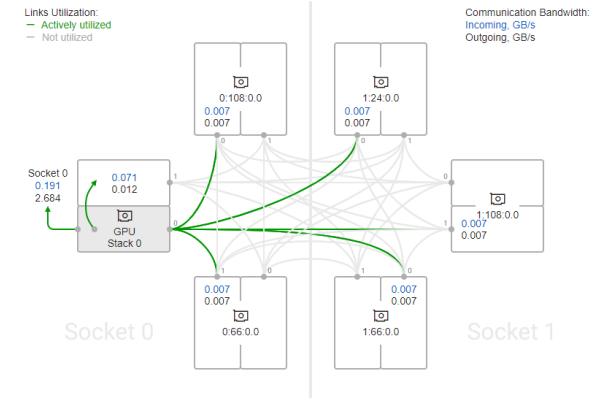
- Identify and optimize performance hotspots of Python code, now supporting Python 3.9.*.

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.

GPU Topology Diagram

Use this topology diagram to examine the GPU interconnect (Xe Link) and identify stack-stack, GPU-socket, and GPU-GPU bandwidths. Hover over a GPU stack to see bandwidth metrics.



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

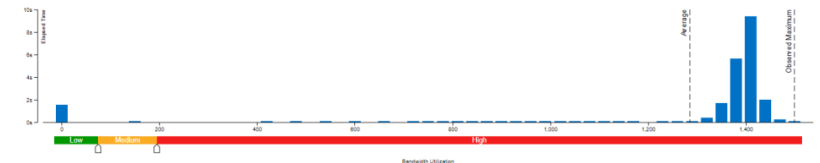
Bandwidth Utilization Histogram

Explore bandwidth utilization over time using the histogram and identify memory objects or functions with maximum contribution to the high bandwidth utilization.

Bandwidth Domain: HBM, GB/sec

Bandwidth Utilization Histogram

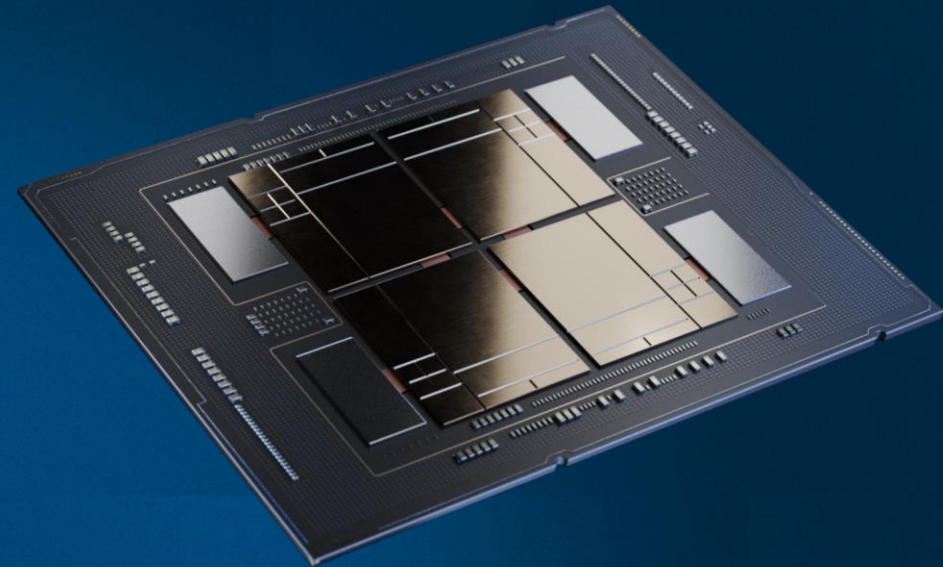
This histogram displays the wait time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and Interconnect bandwidth.



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



Only x86 CPU with High Bandwidth Memory



 HBM	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

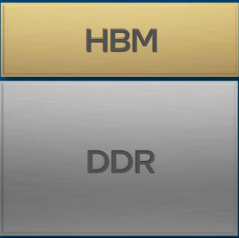
System boots and operates with HBM only



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



High Bandwidth Memory (HBM) Utilization

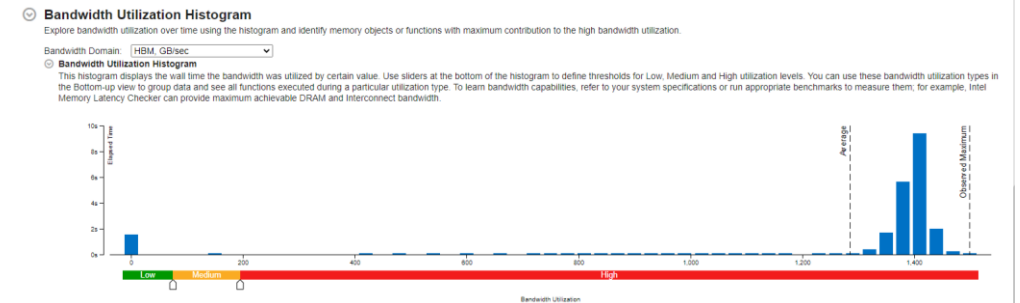
Intel® VTune™ Profiler

Understand HBM memory usage

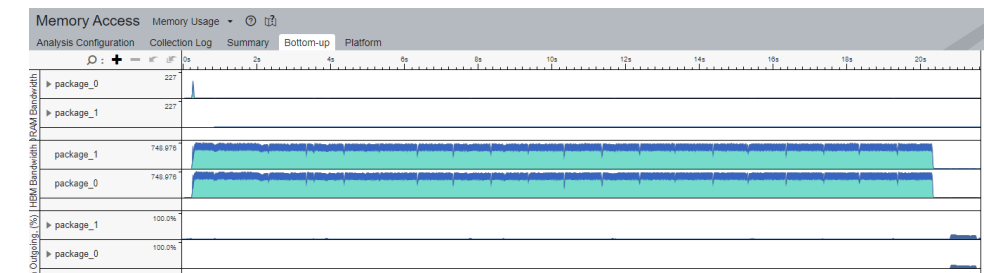
- Is the application performance affected by HBM utilization?
- How is the bandwidth distributed between DRAM vs. HBM?

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.



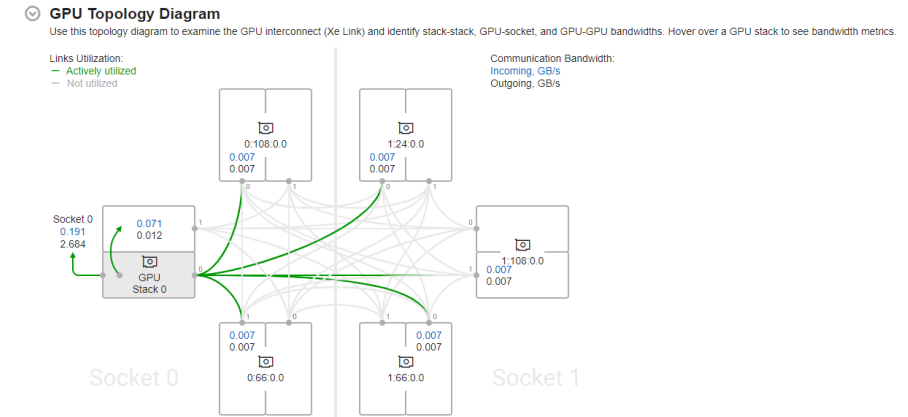
The workload performance in various HBM modes can be evaluated by running the collection in each mode and analyzing the bandwidth as described above.

Get Visibility into Xe Link Cross-card Traffic

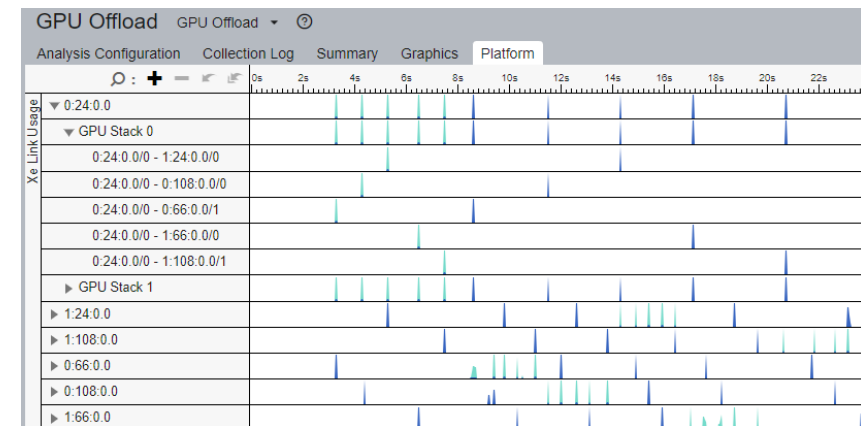
Intel® VTune™ Profiler

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

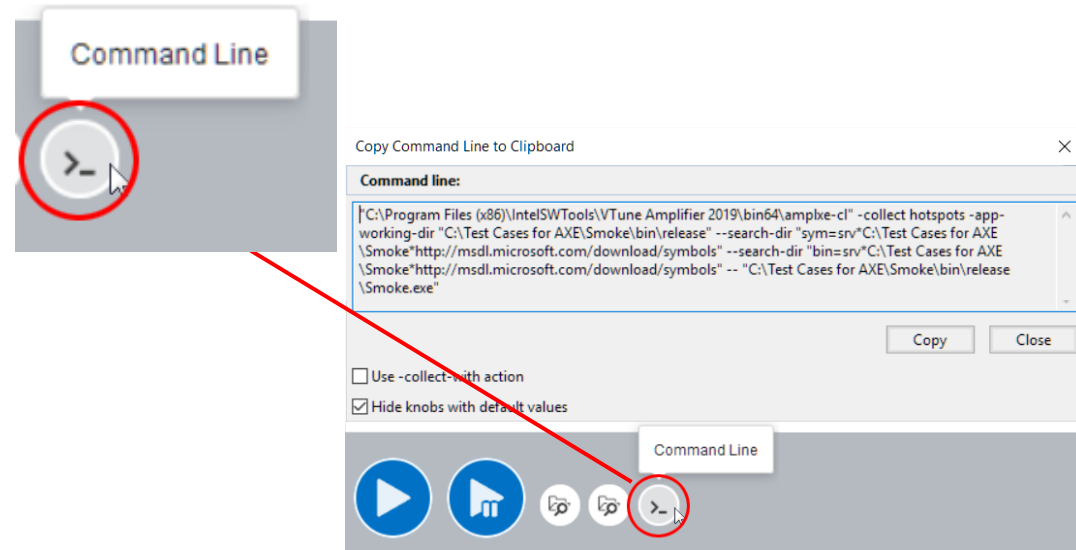
Help:

`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

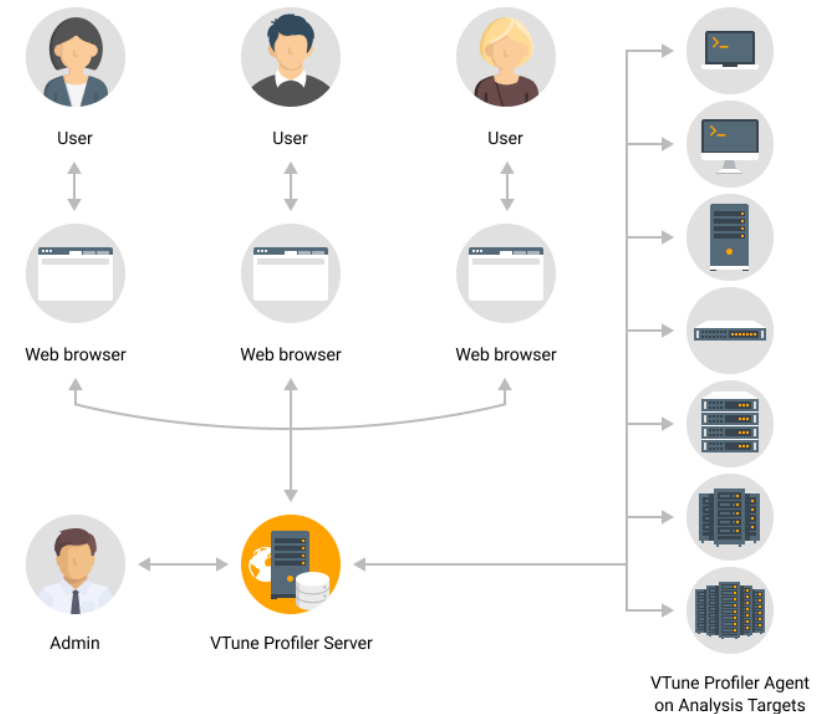
Does your development move to Cloud? VTune is ready to follow!

- VTune server for remote development

The screenshot shows the Intel VTune Profiler Server interface. On the left, a terminal window displays the command `bash-4.$ vtune-backend` and the output `Serving GUI at https://127.0.0.1:45361/`. The main window shows the 'Microarchitecture Exploration' view for a C++ program named 'multiply.c'. The source code is displayed, and the analysis results table shows the following data:

Source	Clockticks	CPI Rate
50 // This leads to bad cache reuse and significant memory sta		
51 // Use Microarchitecture and Memory access analysis to estim		
52 // See the 'multiply2' function implementation to overcome		
53 for(i=tidx; i<msize; i=i+numt) {		
54 for(j=0; j<msize; j++) {		
55 for(k=0; k<msize; k++) {		
56 c[i][j] = c[i][j] + a[i][j]	4.086	5.647
57 }		
58 }		

A tooltip is visible over the CPI Rate column, stating: "The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high CPI."

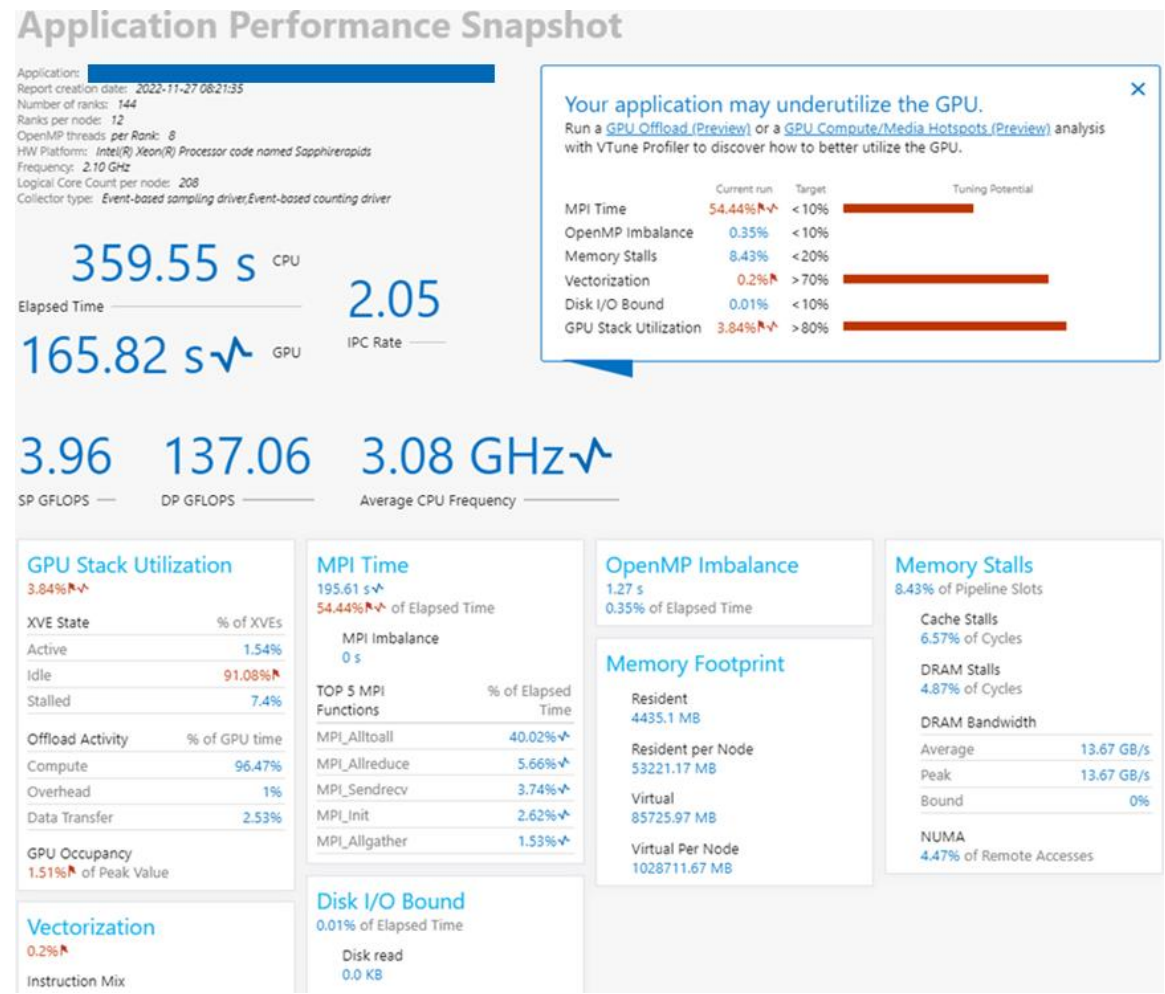


- VTune server for teams
 - Easy onboarding
 - Data sharing & collaboration

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

HPC Performance Characterization

Analysis Configuration | Collection Log | Summary | Bottom-up

Platform Diagram

Effective Physical Core Utilization: 40.3% (19,329 out of 48)

Effective Logical Core Utilization: 39.0% (37,439 out of 96)

Serial Time (outside parallel regions): 8.300s (54.6%)

Parallel Region Time: 6.902s (45.4%)

Estimated Ideal Time: 5.960s (39.2%)

OpenMP Potential Gain: 0.942s (6.2%)

Top OpenMP Regions by Potential Gain

OpenMP Region	OpenMP Potential Gain (%)	OpenMP Region Time
_Z22Iso3dfdVerifyIterationPIS_S_S_iiimm.DIR.OMP.PARALLEL.2\$omp\$parallel:96@/home/gta/iso3dfd_omp_offload/src/iso3dfd_verify.cpp:25:25	0.942s 6.2%	6.902s

Memory Bound: 44.1% of Pipeline Slots

Cache Bound: 14.8% of Clockticks

DRAM Bound: 40.7% of Clockticks

DRAM Bandwidth Bound: 39.6% of Elapsed Time

NUMA: % of Remote Accesses: 65.0%

Bandwidth Utilization Histogram

Vectorization: 99.9% of Packed FP Operations

Instruction Mix:

- SP FLOPs: 17.9% of uOps
 - Packed: 99.9% from SP FP
 - 128-bit: 99.9% from SP FP
 - 256-bit: 0.0% from SP FP
 - 512-bit: 0.0% from SP FP
 - Scalar: 0.1% from SP FP
- DP FLOPs: 0.0% of uOps
 - Packed: 0.0% from DP FP
 - Scalar: 100.0% from DP FP
- x87 FLOPs: 0.0% of uOps
- Non-FP: 82.1% of uOps

FP Arith/Mem Rd Instr. Ratio: 0.562

FP Arith/Mem Wr Instr. Ratio: 3.945

Top Loops/Functions with FP Usage by CPU Time

This section provides information for the most time consuming loops/functions with floating point operations.

Function	CPU Time	% of FP Ops	FP Ops: Packed	FP Ops: Scalar	Vector Instruction Set
[Loop at line 38 in _Z22Iso3dfdVerifyIterationPIS_S_S_iiimm.DIR.OMP.PARALLEL.2]	532.905s	23.3%	100.0%	0.0%	SSE(128); SSE2(128)
[Loop at line 34 in _Z22Iso3dfdVerifyIterationPIS_S_S_iiimm.DIR.OMP.PARALLEL.2]	16.680s	2.5%	100.0%	0.0%	SSE(128); SSE2(128)
[Loop at line 103 in WithinEpsilon]	0.460s	9.5%	0.0%	100.0%	SSE(128)

GPU Utilization when Busy: 25.0%

EU State:

- Active: 25.0%
- Stalled: 71.9%
- Idle: 3.0%

Occupancy: 97.9% of peak value

Offload Time: 34.1% (5.190s) of elapsed time

- Compute: 84.4% (4.381s) of offload time
- Data Transfer: 14.0% (0.728s) of offload time
- Overhead: 1.6% (0.081s) of offload time

Top OpenMP Offload Regions

OpenMP Offload Region	Offload Time	Percentage of Elapsed Time	Data Transfer	Overhead	GPU Utilization when Busy
iso3dfdIteration\$omp\$target\$region:dvc=0@/home/gta/iso3dfd_omp_offload/src/iso3dfd.cpp:50	4.382s	28.8%	0s	0.000s	0.0%
iso3dfd\$omp\$target\$region:dvc=0@/home/gta/iso3dfd_omp_offload/src/iso3dfd.cpp:32	0.808s	5.3%	0.728s	0.081s	0.0%
[Outside any OpenMP Offload Region]		0.0%			25.0%

*NA is applied to non-summable metrics

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.

Elapsed Time [?]: 133.634s

- CPU Time** [?]: 472.871s
- Total Thread Count: 5
- Paused Time [?]: 0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time [?]
multiply1	matrix.exe	472.573s

Hotspots Insights
If you see significant hotspots in the Top Hotspots list, switch to the [Bottom-up](#) view for in-depth analysis per function. Otherwise, use the [Caller/Callee](#) view to track critical paths for these hotspots.

Hotspots [?]

Analysis Configuration Collection Log Summary **Bottom-up** Caller/Callee Top-down Tree Flame Graph Platform

Grouping: Function / Call Stack

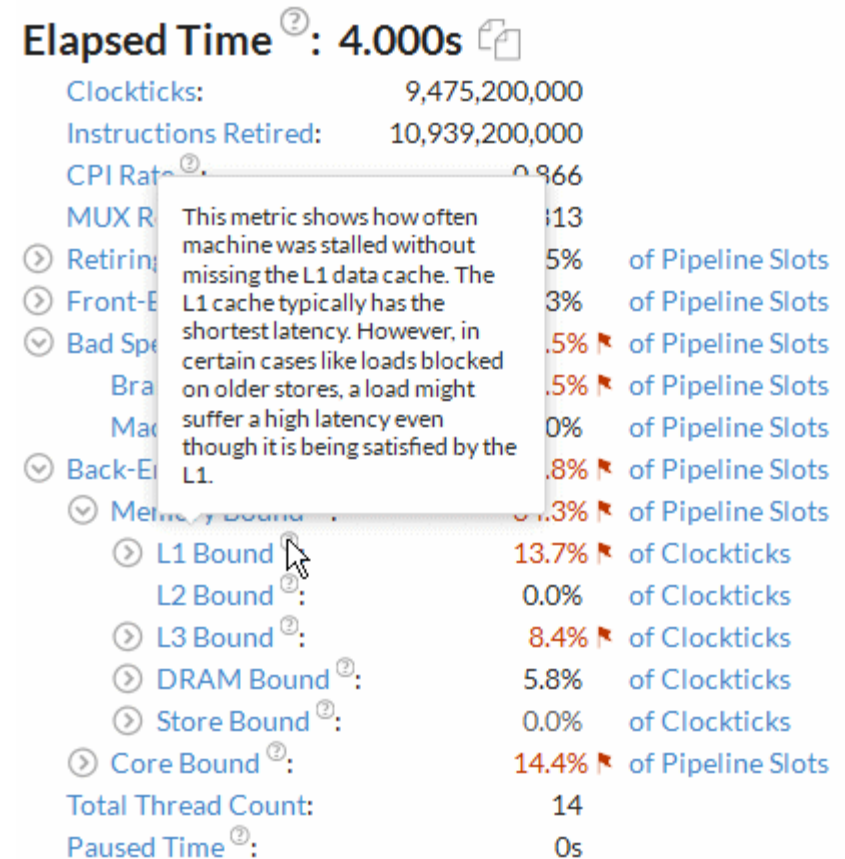
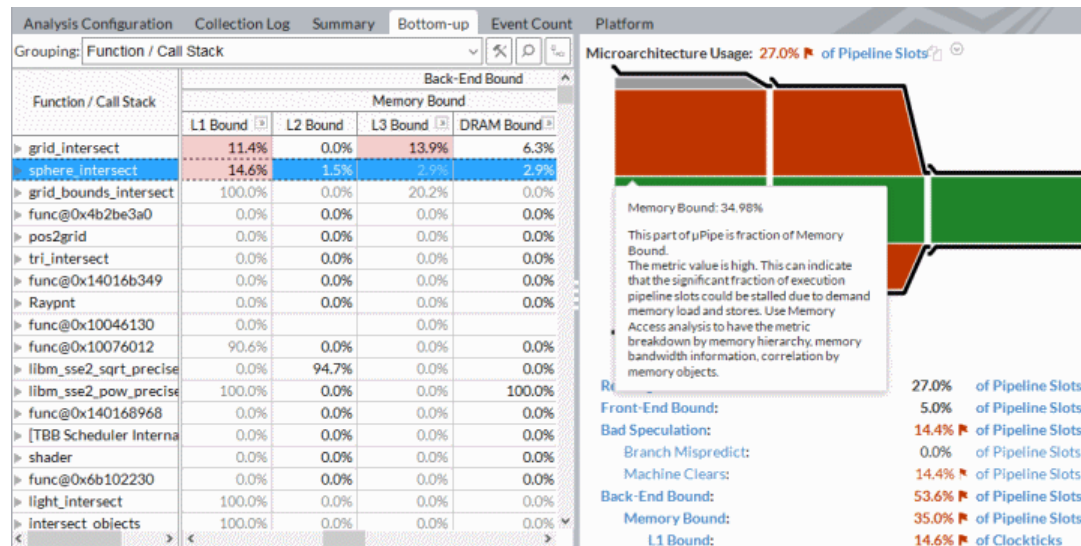
Function / Call Stack ▲	CPU Time			Module	Function (Full)	Source File
	Effective Time	Spin Time	Overhead Time			
▶ __intel_avx_rep_memset	0.012s	0s	0s	libintlc.so.5	__intel_avx_rep_memset	
▶ __printf	0.008s	0s	0s	libc.so.6	__printf	printf.c
▶ matrix_multiply	13.960s	0s	0s	MatrixMultiplication_icc	matrix_multiply	MatrixMultiplication...

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

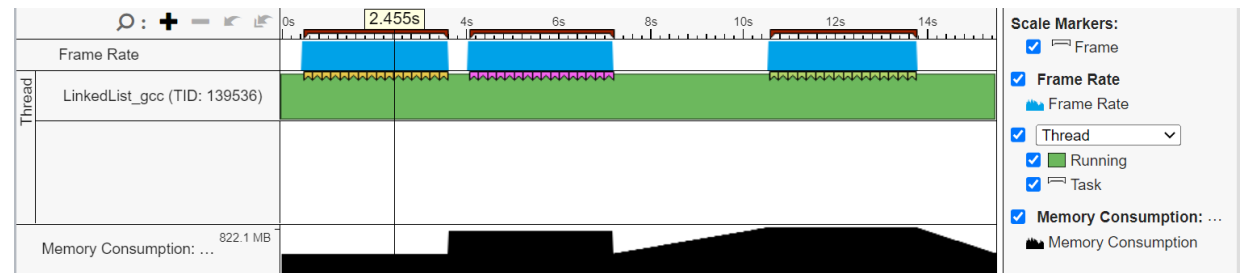
- Lists top memory consuming functions
- memory consumption distribution over time.
- View source to understand cause
- Filter by time using the memory consumption timeline
 - Focus on the peak values on the Timeline pane
- Introduce additional overhead due to instrumentation .

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*

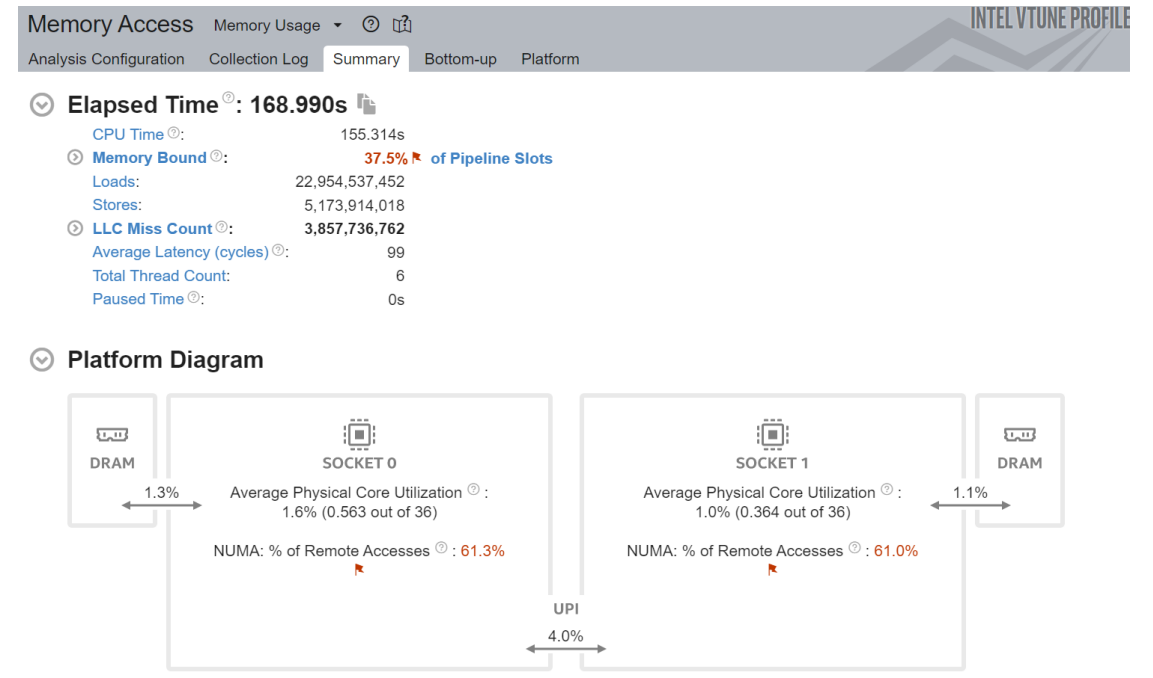
*N/A is applied to non-summable metrics.



Optimize Memory Access

Memory Access Analysis - Intel® VTune™ Profiler

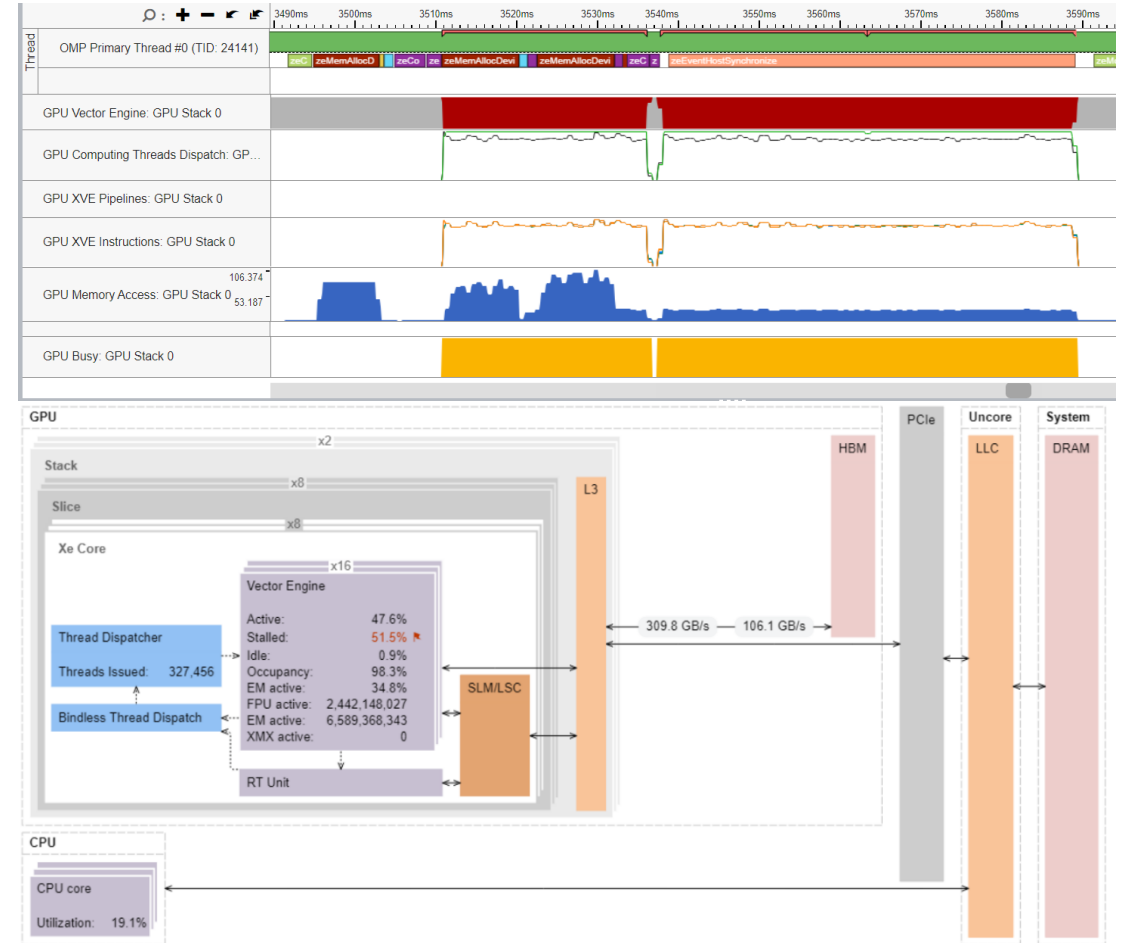
- Tune data structures for performance
 - Attribute cache misses to data structures (not just the code causing the miss)
 - Support for custom memory allocators
 - Shows average load latency in cycles
- Optimize NUMA latency & scalability
 - Auto detect max system bandwidth
 - Detects inter-socket bandwidth



Intel® VTune™ Profiler

Profile GPU Performance

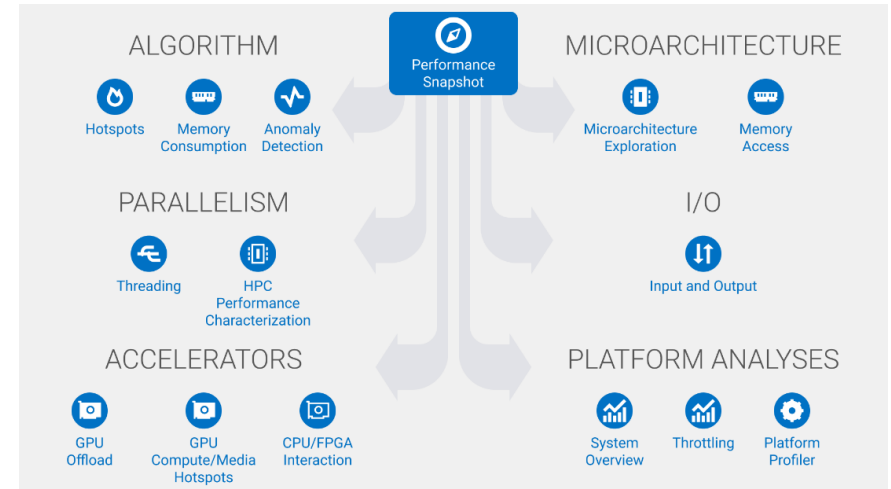
- Explicit support of DPC++, DirectX, Intel® Media SDK, OpenCL™, and OpenMP-offload software technology
- Multi-GPU systems analysis
- GPU Offload cost profiling
 - CPU vs GPU boundness
 - Offload overhead & host-to-device traffic, GPU compute vs data transfer
 - GPU utilization and software queues per 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



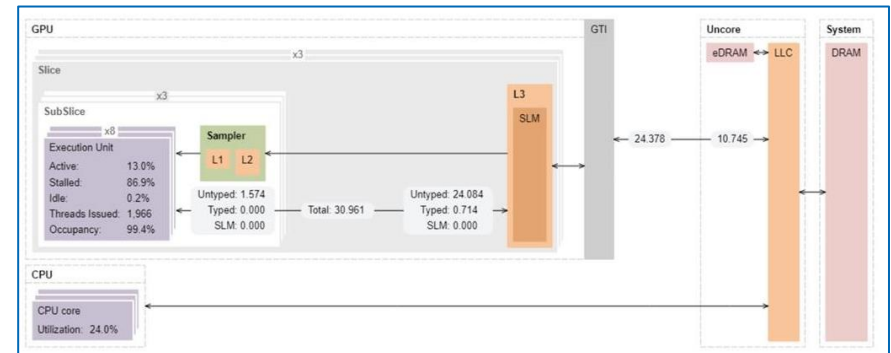
GPU Performance Problems

Addressing performance issues with dynamic analysis tools

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



Source	Assembly	GPU Instructions Executed by Instruction T...
158	<code>dx = ptr[j].pos[0] - ptr[i].pos[0]</code>	75,002,500
159	<code>dy = ptr[j].pos[1] - ptr[i].pos[1]</code>	12,500,000
160	<code>dz = ptr[j].pos[2] - ptr[i].pos[2]</code>	12,500,000



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

The screenshot displays the Intel VTune Profiler interface for HPC Performance Characterization. The main section shows GPU Stack Utilization at 30.8%. Below this, the EU State is detailed: Active at 28.8%, Stalled at 71.1%, and Idle at 0.1%. Occupancy is 99.2% of peak value. Offload Time is 31.9% (18.907s) of elapsed time, broken down into Compute (96.5%), Data Transfer (1.5%), and Overhead (2.0%). A table lists the top OpenMP Offload Regions.

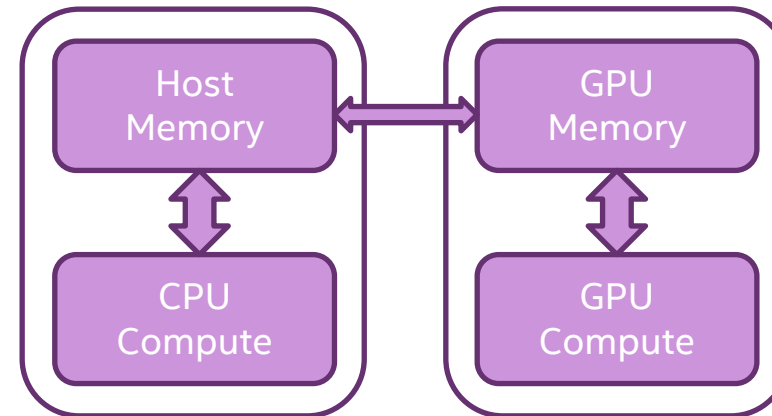
OpenMP Offload Region	Offload Time	Percentage of Elapsed Time	Data Transfer	Overhead	EU Array Active
Iso3dfdIteration\$omp\$target\$region:dvc=0@/home/intel/rroy/oneAPI-samples/DirectProgramming/C++/StructuredGrids/iso3dfd_omp_offload/src/iso3dfd.cpp:50	18.252s	30.8%	0s	0.001s	28.8%
Iso3dfd\$omp\$target\$region:dvc=0@/home/intel/rroy/oneAPI-samples/DirectProgramming/C++/StructuredGrids/iso3dfd_omp_offload/src/iso3dfd.cpp:332	0.655s	1.1%	0.281s	0.374s	0.0%
[Outside any OpenMP Offload Region]		0.0%			0.0%

*NA is applied to non-summable metrics.

Host and GPU Data Transferring

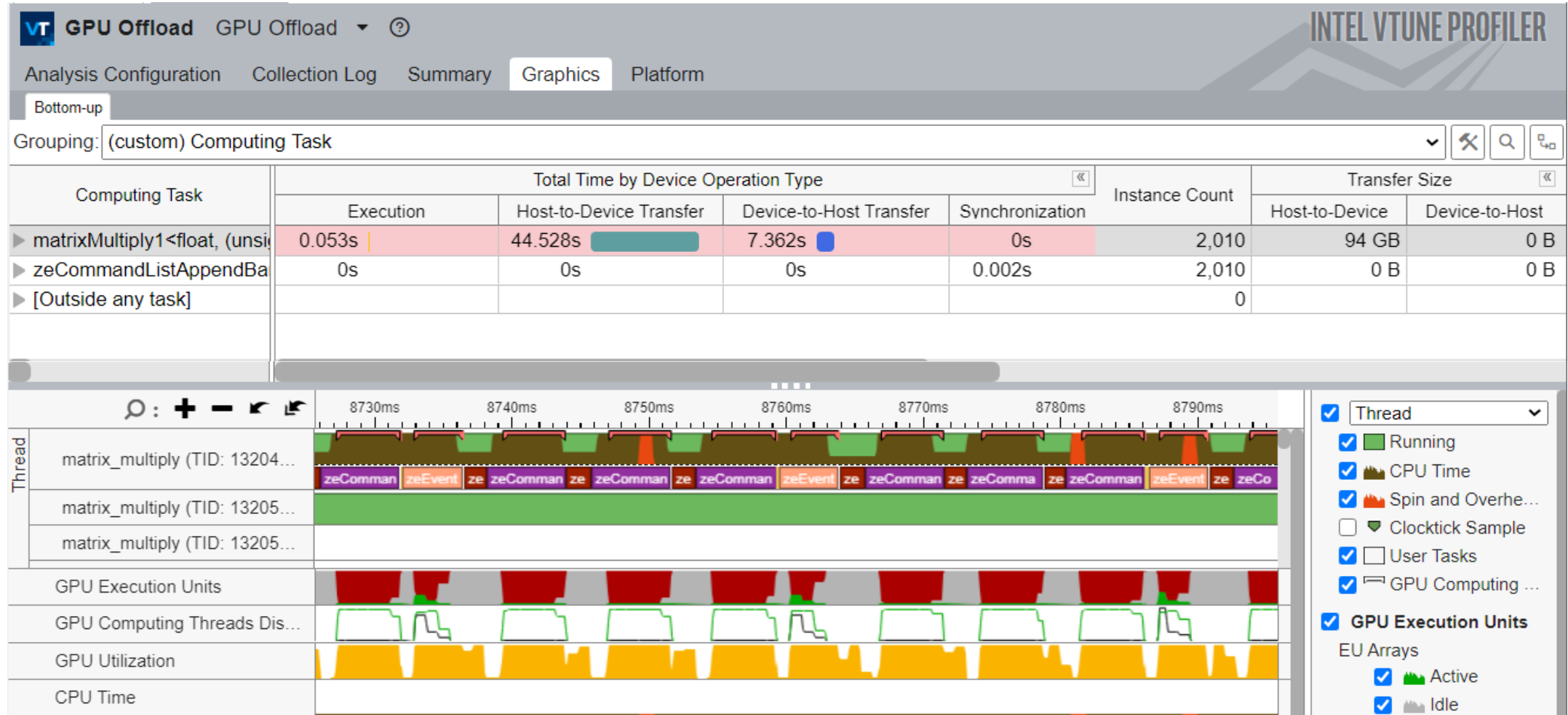
A commonly known problem of host-to-device transfer performance

- 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

Occupancy: 80.4%

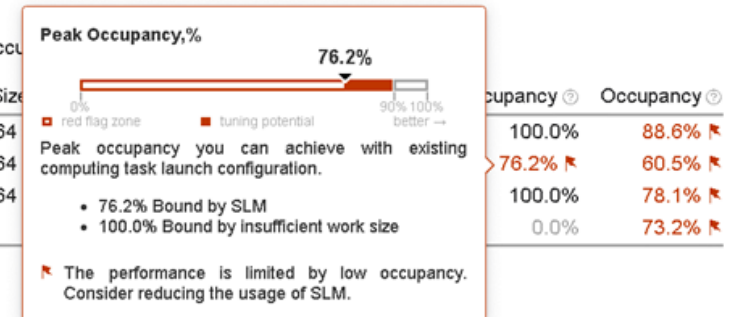
Identify too large or too small computing tasks with low occupancy that make the EU array idle while waiting for the scheduler. Note that frequent SLM accesses and barriers may affect the maximum possible occupancy.

Hottest GPU Computing Tasks with Low Occupancy

This section lists the most active computing tasks running on the GPU with a low Occu

Computing Task	Total Time	Global Size	Local Size
kernel_ocl_path_trace_shadow_blocked_dl	32.492s	128 x 185	64
kernel_ocl_path_trace_shader_sort	21.426s	128 x 185	64
kernel_ocl_path_trace_shader_eval	17.506s	128 x 185	64
[Others]	14.209s		

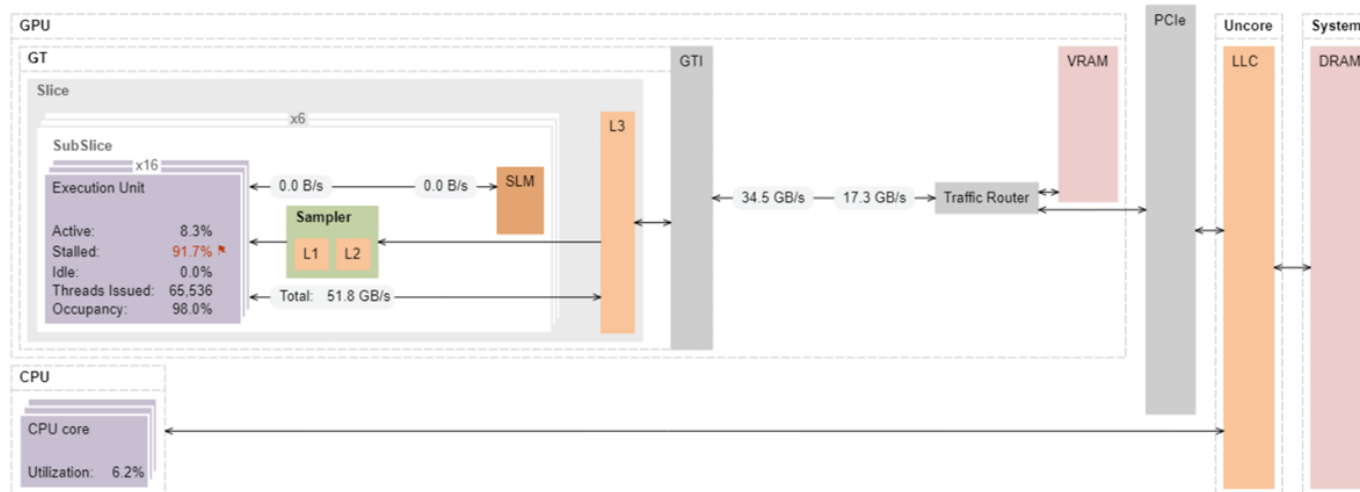
**N/A is applied to non-summable metrics.*



```
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

GPU Compute/Media Hotspots (preview) ©

Analysis Configuration Collection Log Summary Graphics iso3dfd_kernels.cpp

Source Assembly

Source Line	Source	Estimated GPU Cycles
428	for (auto iter = 0; iter < kHalfLength; iter++) {	
429	front[iter] = front[iter + 1];	
430	}	
432	// Only one new data-point read from global memory	
433	// in z-dimension (depth)	
434	front[kHalfLength] = prev[gid + kHalfLength * nxy];	8.573e+8
436	// Stencil code to update grid point at position given by global id (gid)	
437	float value = c[0] * front[0];	3.429e+8
438	#pragma unroll(kHalfLength)	
439	for (auto iter = 1; iter <= kHalfLength; iter++) {	
440	value += c[iter] * (front[iter] + back[iter - 1] + prev[gid + iter] +	1.367e+10
441	prev[gid - iter] + prev[gid + iter * nx] +	1.097e+10
442	prev[gid - iter * nx]);	2.358e+10
443	}	
444		
445	next[gid] = 2.0f * front[0] - next[gid] + value * vel[gid];	1.929e+9
446		
447	gid += nxy;	
448	begin_z++;	3.429e+8
449	}	
450	}	
451		
452	/*	
453	* Host-side SYCL Code	
454	*	
455	* Driver function for ISO3DFD SYCL code	
456	* Uses ptr_next and ptr_prev as ping-pong buffers to achieve	
457	* accelerated wave propagation	

Basic Block Latency

Welcome src-analysis_stall_1

GPU Compute/Media Hotspots (preview) ©

Analysis Configuration Collection Log Summary Graphics 3_GPU_linear.cpp

Source Assembly

Assembly grouping: Address

Source Line	Source	Stall Count by Stall Type					Add...	Sour...	Assembly	Stall Count by Stall Type				
		Pipe	Send	Dist or Acc	SBID	Syn				Pipe	Send	Dist or Acc	SBID	
42	sets to indices to ex						0x960	58	shl (16 M16) r114.0<					
43	n2 * n3;	1,407	0	41	0	0	0x968	58	send.ugm (32 M0) r11	0	0	37	0	
44	dx[0] + kHalfLength;	120	0	5	0	0	0x978	58	add (16 M0) r118.0<1					
45	dx[1] + kHalfLength;	125	0	1	0	0	0x980	58	add (16 M16) r120.0<					
46	dx[2] + kHalfLength;						0x988	58	send.ugm (32 M0) r1	0	0	154	0	
47							0x998	56	add (32 M0) acc0.0<1	3	0	0	57	
48	te linear index for ea						0x9a0	56	add (32 M0) acc0.0<1	3	0	814	829	
49	i * n2n3 + j * n3 + k;	3,222	0	198	0	0	0x9b0	57	add (32 M0) r40.0<1>	1	0	66	1,504	
50							0x9c0	57	(W) mul (1 M0) acc0.	40	0	201	0	
51	ce values for each cel						0x9d0	57	add (32 M0) acc2.0<1	2	0	0	4,250	
52	e = prev_acc[idx] * cd	844	0	23	0	0	0x9d8	56	sync.nop null {Compa	0	0	0	0	
53							0x9e0	56	add (32 M0) r35.0<1>	9	0	0	0	
54	= 1; x <= kHalfLength						0x9e8	57	(W) mul (1 M0) r61.0	83	0	0	51	
55		113	0	23	0	0	0x9f0	57	(W) mul (1 M0) r62.0	106	0	0	0	
56	f_acc[x] * (prev_acc[3,048	0	943	6,757	0	0xa00	57	(W) mach (1 M0) r66.	45	0	60	0	
57	prev_acc[20,038	0	1,457	23,052	0	0xa10	57	(W) shl (1 M0) r63.0	41	0	0	0	
58	prev_acc[11,350	0	824	13,521	0	0xa18	57	subb (16 M0) r84.0<1	53	0	47	0	
59							0xa28	57	sync.nop null {Compa	0	0	44	0	
60	dx] = 2.0f * prev_acc	174	0	289	9,676	0	0xa30	57	(W) add (1 M0) r90.0	60	0	0	0	
61	value * vel	7	0	58	0	0	0xa38	58	add (32 M0) r49.0<1>	5	0	0	5,428	
62	device code						0xa48	56	sync.nop null {Compa	0	0	0	0	
63							0xa50	56	add (32 M0) acc2.0<1	7	0	0	376	
64							0xa58	57	(W) mov (2 M0) r71.0	108	0	0	0	
65							0xa60	57	add3 (16 M0) r110.0<	109	0	65	0	
66							0xa70	57	subb (16 M16) r85.0<	0	0	0	0	

Stall Sampling

Custom Analysis with VTune Profiler

The image shows the Intel VTune Profiler interface with several red boxes and labels indicating the steps for creating a custom analysis:

- Step 1:** Selecting the **GPU Compute/Media Hotspots (preview)** analysis from the **ALGORITHM** category in the main menu.
- Step 2:** Clicking the **Customize a copy of the selected analysis.** button in the top right of the analysis preview pane.
- Step 3:** Enabling the **allow-multiple-runs** checkbox in the **GPU profiling mode** section.
- Step 3:** Configuring the **GPU events selection** table.

The **GPU events selection** table is as follows:

Event Name	Description
<input checked="" type="checkbox"/> L3_READ_L3BANK0 (L3_READ_L3BAN...	The number of L3 bank 0 read requests
<input checked="" type="checkbox"/> L3_READ_L3BANK1 (L3_READ_L3BAN...	The number of L3 bank 1 read requests
<input checked="" type="checkbox"/> L3_READ_L3BANK10 (L3_READ_L3BA...	The number of L3 bank 10 read requests
<input checked="" type="checkbox"/> L3_READ_L3BANK11 (L3_READ_L3BA...	The number of L3 bank 11 read requests
<input checked="" type="checkbox"/> L3_READ_L3BANK12 (L3_READ_L3BA...	The number of L3 bank 12 read requests
<input checked="" type="checkbox"/> L3_READ_L3BANK13 (L3_READ_L3BA...	The number of L3 bank 13 read requests
<input checked="" type="checkbox"/> L3_READ_L3BANK14 (L3_READ_L3BA...	The number of L3 bank 14 read requests
<input checked="" type="checkbox"/> L3_READ_L3BANK15 (L3_READ_L3BA...	The number of L3 bank 15 read requests
<input type="checkbox"/> ALU0 FLT16 Instructions Executed (EU...	The number of FLT16 instructions executed in ALU0 INT64 pipeline.
<input type="checkbox"/> ALU0 FLT32 Instructions Executed (EU...	The number of FLT32 instructions executed in ALU0 INT64 pipeline.
<input type="checkbox"/> ALU0 FLT64 Instructions Executed (EU...	The number of FLT64 instructions executed in ALU0 INT64 pipeline.
<input type="checkbox"/> ALU0 instructions executed by CCCS (...)	The number of instructions executed in EU ALU0 INT64 pipeline by co...

The **Computing task of interest** table is as follows:

Computing task of interest	Instance step
*	1
Enter computing task of interest	Default

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

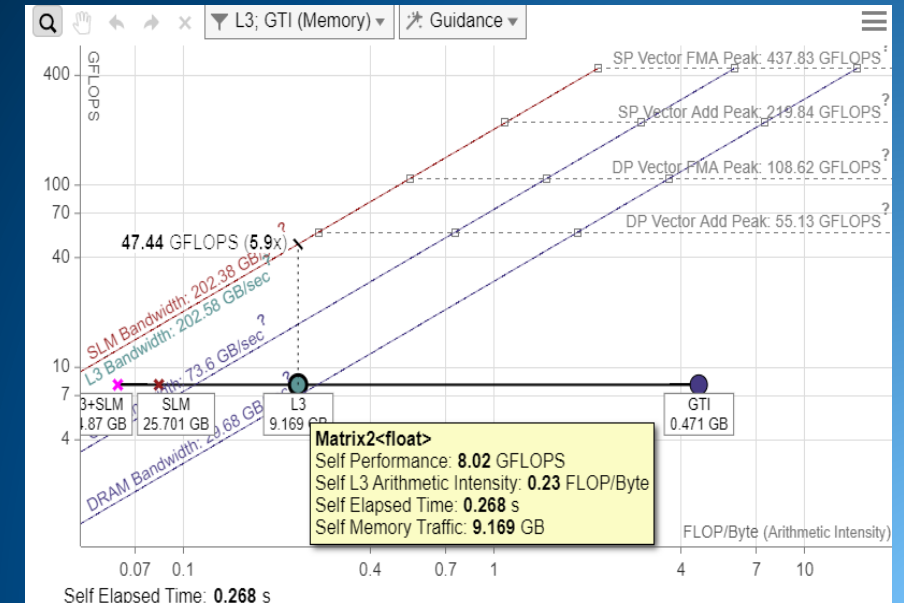
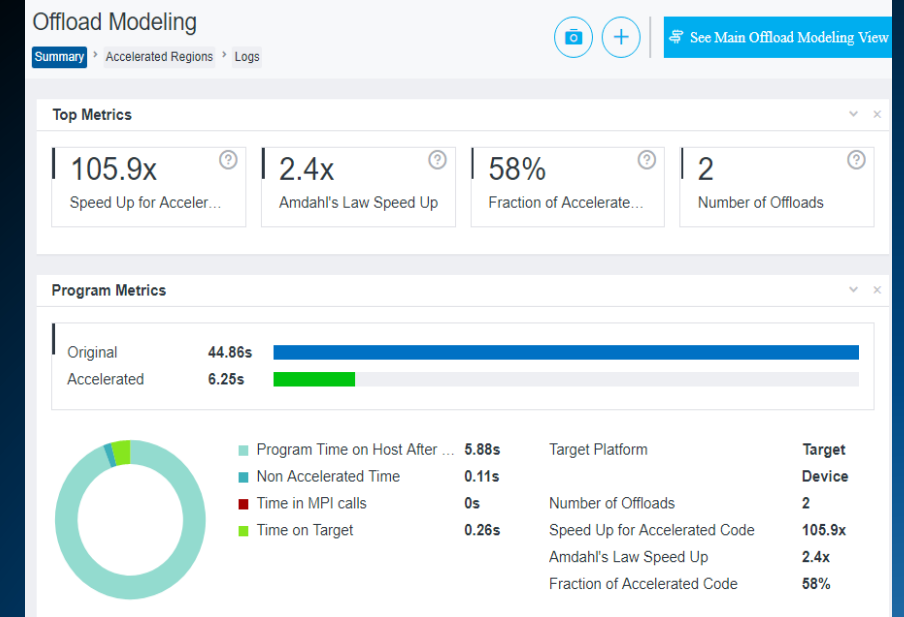
Thread Prototyping

- Add effective threading to unthreaded applications

Flow Graph Analyzer

- Create, visualize and analyze task and dependency computation graphs

[Learn more: Intel.com/advisor](https://www.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

The screenshot shows the Intel Advisor Vectorization Advisor interface. The top bar indicates 'Elapsed time: 0.50s' and 'Vectorized' status. The filter is set to 'mandelbrot.cpp' and 'Loops'. The table below shows performance data for various loops, including CPU Time, Type, Why No Vectorization?, and Vectorized Loops (Efficiency, Gain E..., VL).

Function Call Sites and Loops	Performance Issues	CPU Time		Type	Why No Vectorization?	Vectorized Loops					
		Total Time	Self Time			Vector ...	Efficiency	Gain E...	VL		
[loop in serial_mandelbrot at mandelbrot.cpp:70]		0.202s	35.3%	0.202s	27.9%	Scalar					
[loop in main\$omp\$parallel@164 at mandelbrot.cpp:181]		0.152s		0.152s		Scalar					
[loop in main\$omp\$parallel@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 main\$omp\$parallel@164 at mandelbrot.cpp:169]	1 Data type conv...	0.162s	28.3%	0.010s		Scalar					
[loop in serial_mandelbrot at mandelbrot.cpp:58]	1 Data type conv...	0.202s	35.3%	0.000s		Scalar					
[loop in serial_mandelbrot at mandelbrot.cpp:57]	1 Data type conv...	0.202s	35.3%	0.000s		Scalar					
[loop in simd_mandelbrot at mandelbrot.cpp:112]	1 Data type conv...	0.100s		0.000s		Scalar					
[loop in main\$omp\$parallel@164 at mandelbrot.cpp:164]	2 Assumed depe...	0.162s	28.3%	0.000s		Threaded (OpenMP)					

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

Intel.com/advisor

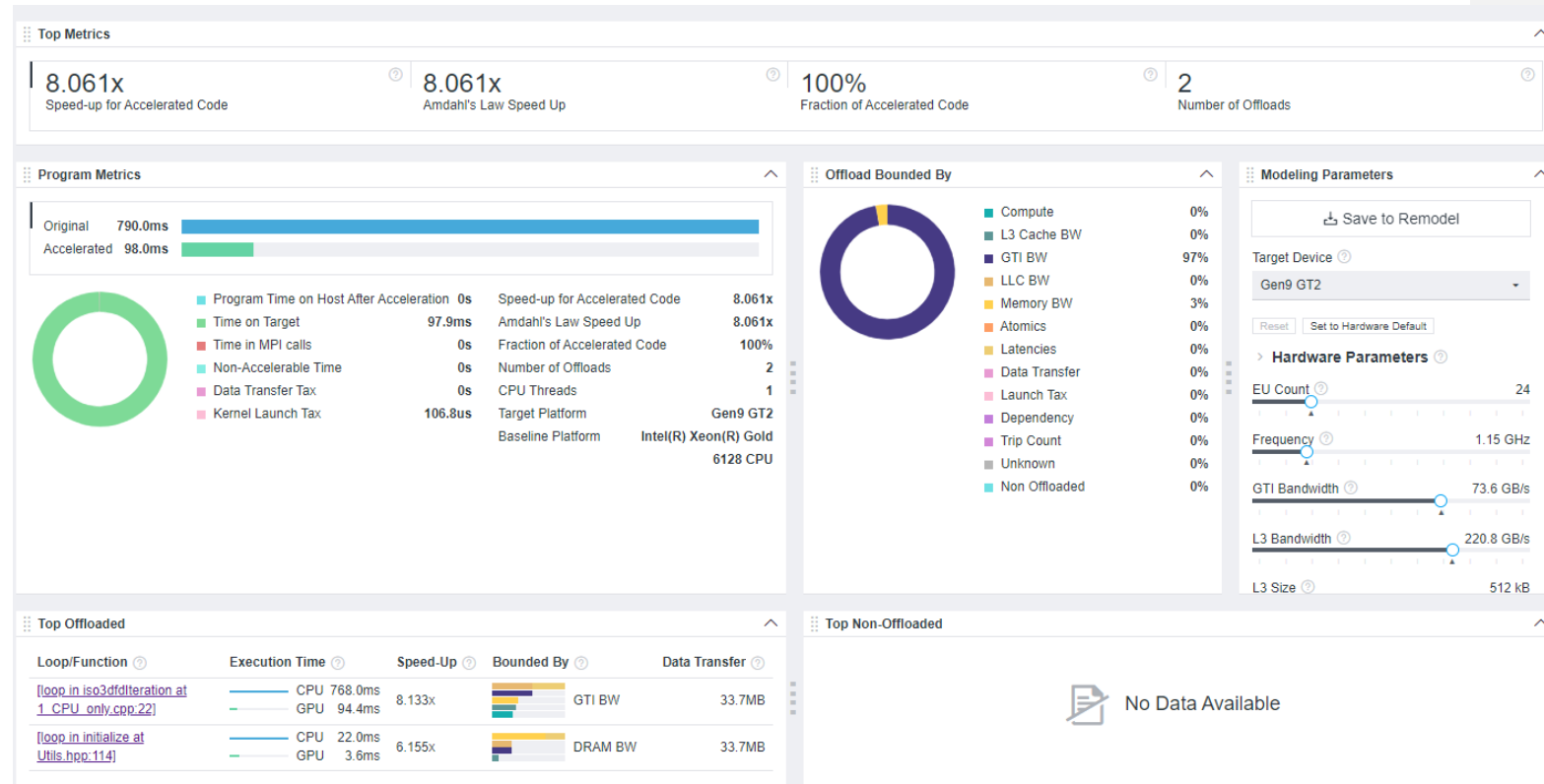
Design your code for efficient offload

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

Loop/Function	Performance Issues	Measured			Basic Estimated Metrics				
		Time	Region	Iteration Space	Speed-Up	Time	Bounded By	Offload Summary	W
▶ [loop in iso3dfdIteration at 1_CPU_only.cpp:22]	🔴 Code region is recommended for offloading	768.0ms		CC 20 TC 128	8.133x	94.4ms 96.3%	GTI BW	▶ Offloaded	
▶ [loop in initialize at Utils.hpp:114]	🔴 Code region is recommended for offloading	22.0ms		CC 1 TC 144	6.155x	3.6ms 3.7%	DRAM BW	▶ Offloaded	

Source × Top Down × Recommendations ×

① Possible offloading

Confidence level: high

Based on the **Offload Modeling** results, this code region is potentially profitable to offload.

💡 Code region is recommended for offloading

Confidence level: high

Based on the **Offload Modeling** results, this code region is potentially profitable to offload. Estimated relative speedup on the **Gen9 GT2** accelerator is **8.13** compared to the current platform. Consider using [Data Parallel C++ \(DPC++\)](#) or [OpenMP* Offload Programming Model](#) to offload this region to the target accelerator.

Example of using a DPC++ parallel for construct for offloading: ☺

```
...
    cgh.parallel_for<kernel>(N, [=](id<1> i) {
...

```

Example of using OpenMP target construct for offloading: ☺

```
...
#pragma omp target teams distribute parallel for map(to: matrixA, matrixB) map(from: matrixC) private(i, j, k)
...

```

Intel, and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.
© Intel Corporation

Details × Data Transfer Estimations > ^ ×

🔴 [loop in iso3dfdIteration at 1_CPU_onl...

ESTIMATED SPEED-UP: 8.133X

Estimated Time
94.4ms

Measured Time
768.0ms

BOUNDED BY: GTI BW

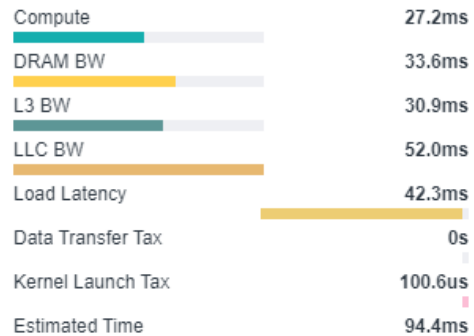


Table of Contents:

Possible offloading

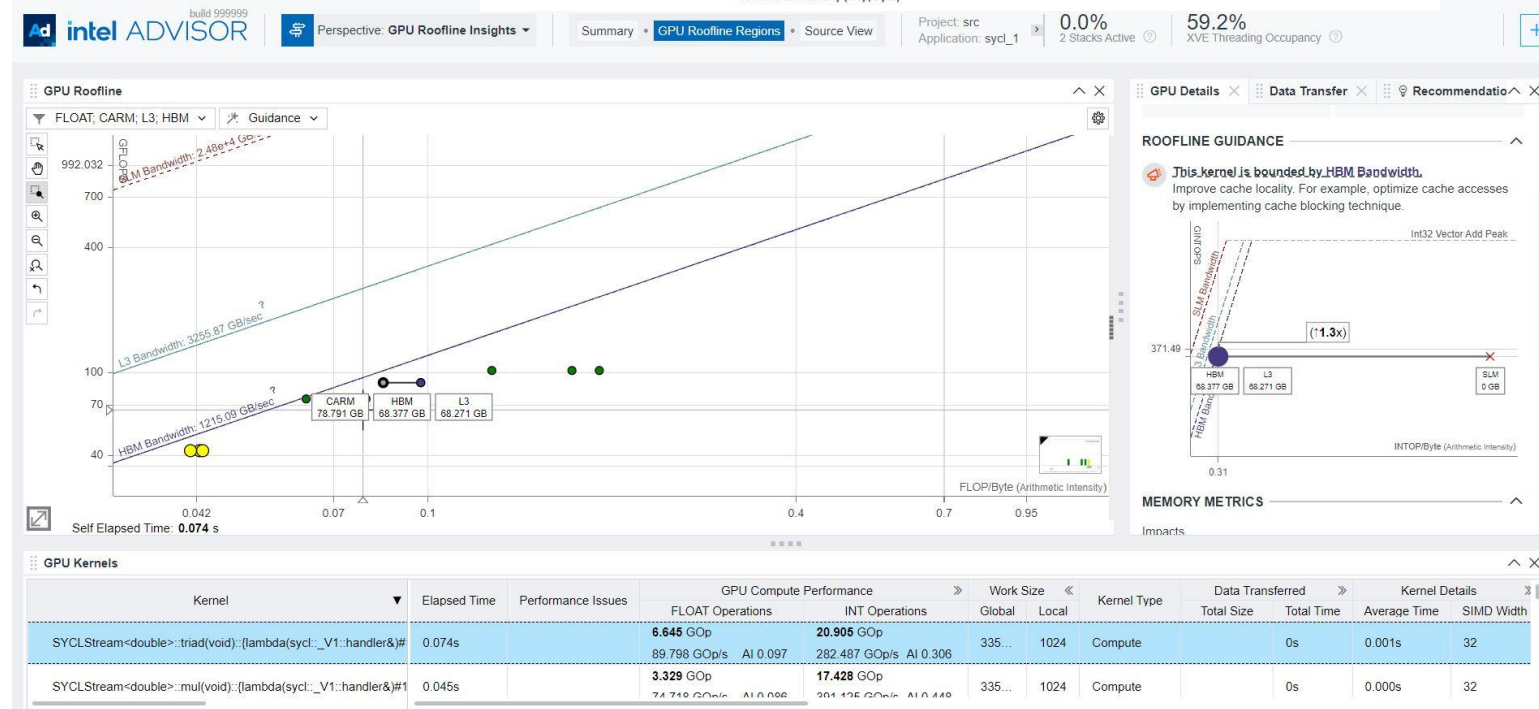
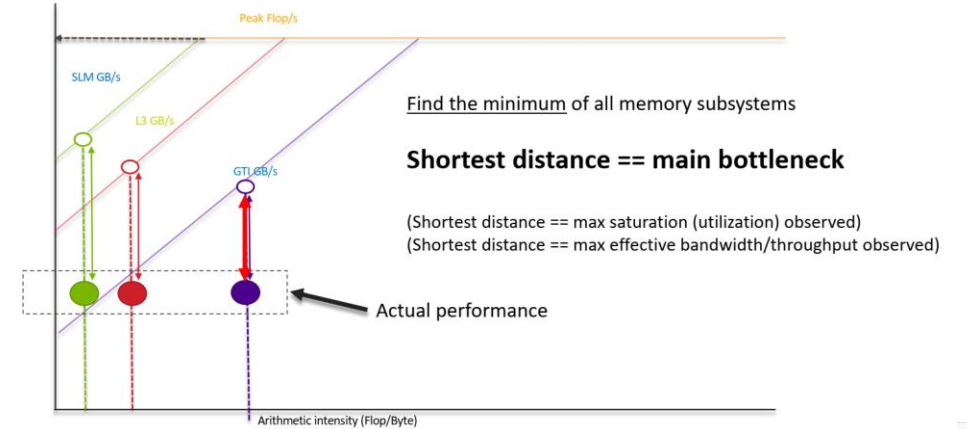
- Code region is recommended for offloading

Find Effective Optimization Strategies

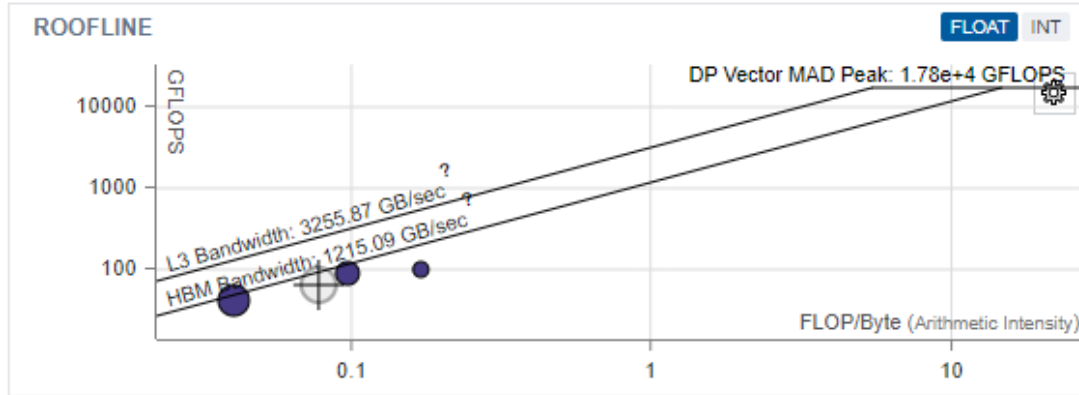
Intel® Advisor - Roofline Analysis on GPU

GPU Roofline Performance Insights

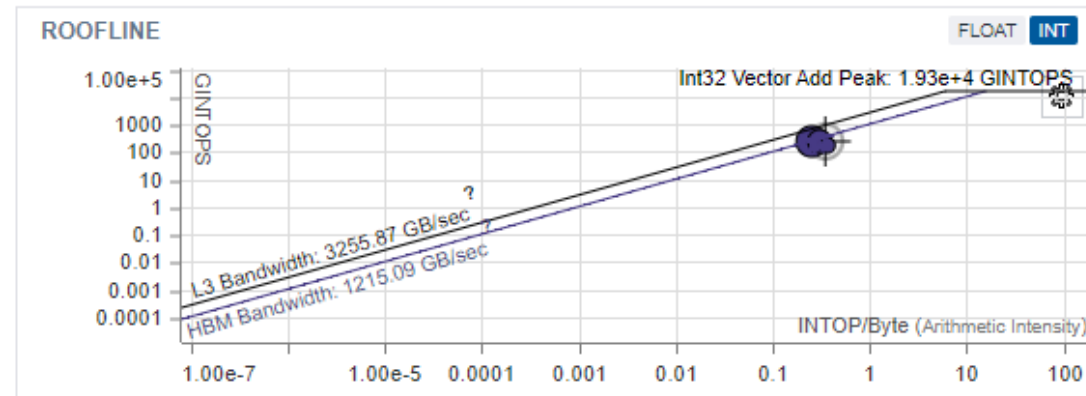
- Highlights poor performing kernels
- Shows performance 'headroom' for each kernels
 - Which can be improved
 - Which are worth improving
- Shows likely causes of bottlenecks
 - Memory bound vs. compute bound
- Suggests next optimization steps



GPU Roofline Analysis



This application is bounded by HBM Bandwidth: **844.67** 69% of 1215.09 GB/sec



This application is bounded by HBM Bandwidth: **844.67** 69% of 1215.09 GB/sec

intel ADVISOR build 999999 Perspective: GPU Roofline Insights Summary GPU Roofline Regions Source View Project: src Application: sycl_1

Program Metrics

10.03s Program Elapsed Time | 0.30s GPU Time | 0.02s Data Transfer Time | 9.73s CPU Time

GPU

GFLOPS: 65.88 | GINTOPS: 298.59 | HBM B... 844.67 GB/s

GFLOP: 20.05 FP AI (HBM): 0.08 | GINTOP: 90.88 INT AI (HBM): 0.35 | HBM Traffic: 257.09 GB

2 Stacks Active: 0.0% | XVE Threading Occupancy: 59.2%

CPU

GFLOPS: 0.02 | GINTOPS: 0.02

GFLOP: 0.20 FP AI: 0.06 | GINTOP: 0.21 INT AI: 0.07

Thread Count: 1

OP/S and Bandwidth

GPU

CPU

No Data Available

This application is bounded by HBM Bandwidth: **844.67** 69% of 1215.09 GB/sec

Top Hotspots

Kernel	Elapsed Ti...	GFLOPS	GINTOPS ...	Global/Lo...	Active/Sta...	Function Call S...	Self Elapsed Ti...	Self GFLOPS	Self GINTOPS ...
SYCLStrea...	0.08s	42.188	265.414	33554432/...	8.8/79.6/11.5	loop in intel...	0.08s		0.15729049530...
SYCLStrea...	0.07s	89.798	282.487	33554432/...	7.8/77.1/15.1	loop in piEngue...	0.02s		0.00056129438...
sycl::V1::...	0.07s	102.107	211.759	524288/1024	5.1/56.8/38.1	loop in func@0...	0.01s		
SYCLStrea...	0.04s	74.718	391.125	33554432/...	10.8/69.3/1...	loop in func@0...	0.01s		
SYCLStrea...	0.04s	0	437.821	33554432/...	8.3/59.3/32.4	loop in livm:Ba...	0.01s		

Platform Information

Performance Characteristics

GPU

- XVE Array Active: 8.0%
- XVE Array Stalled: 69.9%
- XVE Array Idle: 22.1%

Average GPU Vector Engine Utilization: 8.0%

Incoming GTI Bandwidth Bound: 35.2%

Outgoing GTI Bandwidth Bound: 17.8%

CPU

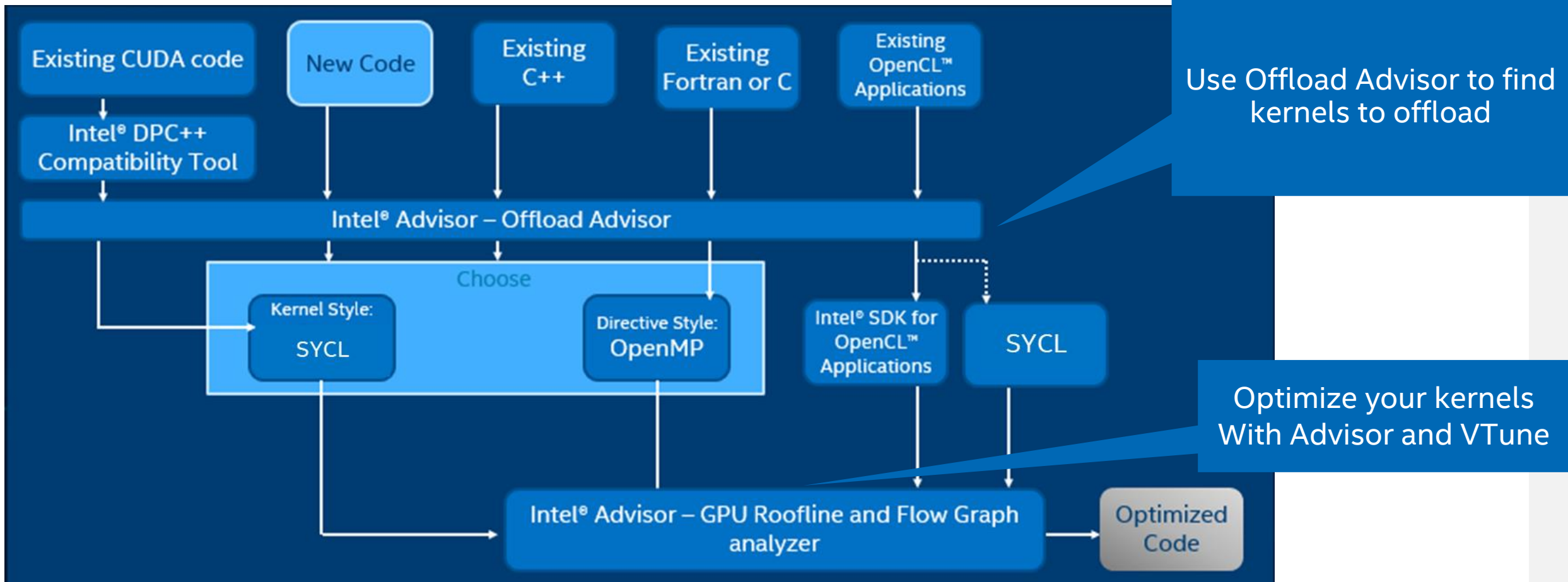
Total CPU Time: 3.58s 100%

Time in 9 Vectorized Loops: 0.46s 13%

Time in Scalar Code: 3.12s 87%

Recommended Workflow

Using Intel® Analyzers to increase performance



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

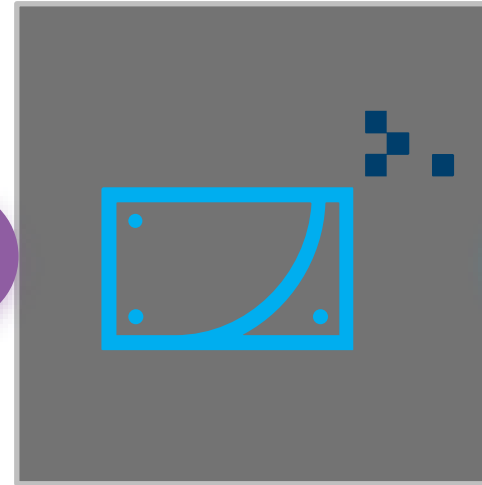
Workflow



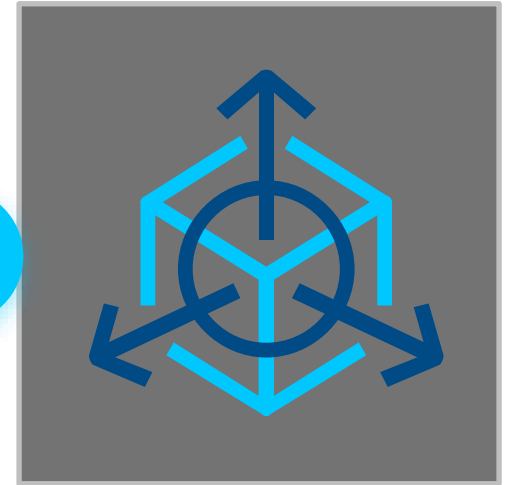
Log into an Intel®
DevCloud GPU
node and
configure the
MandelbrotOMP
sample



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 profiling tools in
commandline and view the results

DevCloud Setup



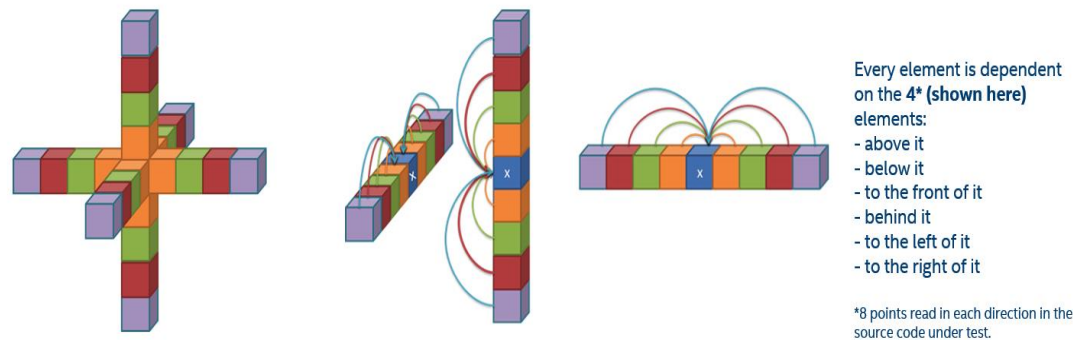
Intel DevCloud provides a free environment for testing the Intel CPUs and GPUs. Intel oneAPI toolkits are already installed and set up for use.

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/g  
uided_iso3dfd_GPUOptimization  
  
mkdir build  
  
cd build  
  
make ..  
  
make
```

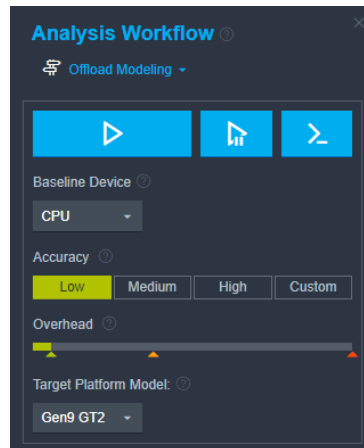
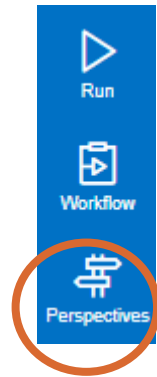
Intel[®] Advisor Exercise

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



Perspective Selector

Perspective is a predefined analysis workflow that consists of multiple profiling (collection) steps helping you understand certain aspects of your application performance. In the user interface, each perspective has its own set of panes and grids providing insights on application performance.

GPU Modeling

GPU Analysis

CPU Analysis and Modeling

Offload Modeling Perspective

- Identify high-impact opportunities to offload you code to an accelerator.
- Determine potential benefit and key bottlenecks even before running the code on the accelerator.
- Get reasons why certain regions are not recommended for offloading.

[Learn more](#)

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:

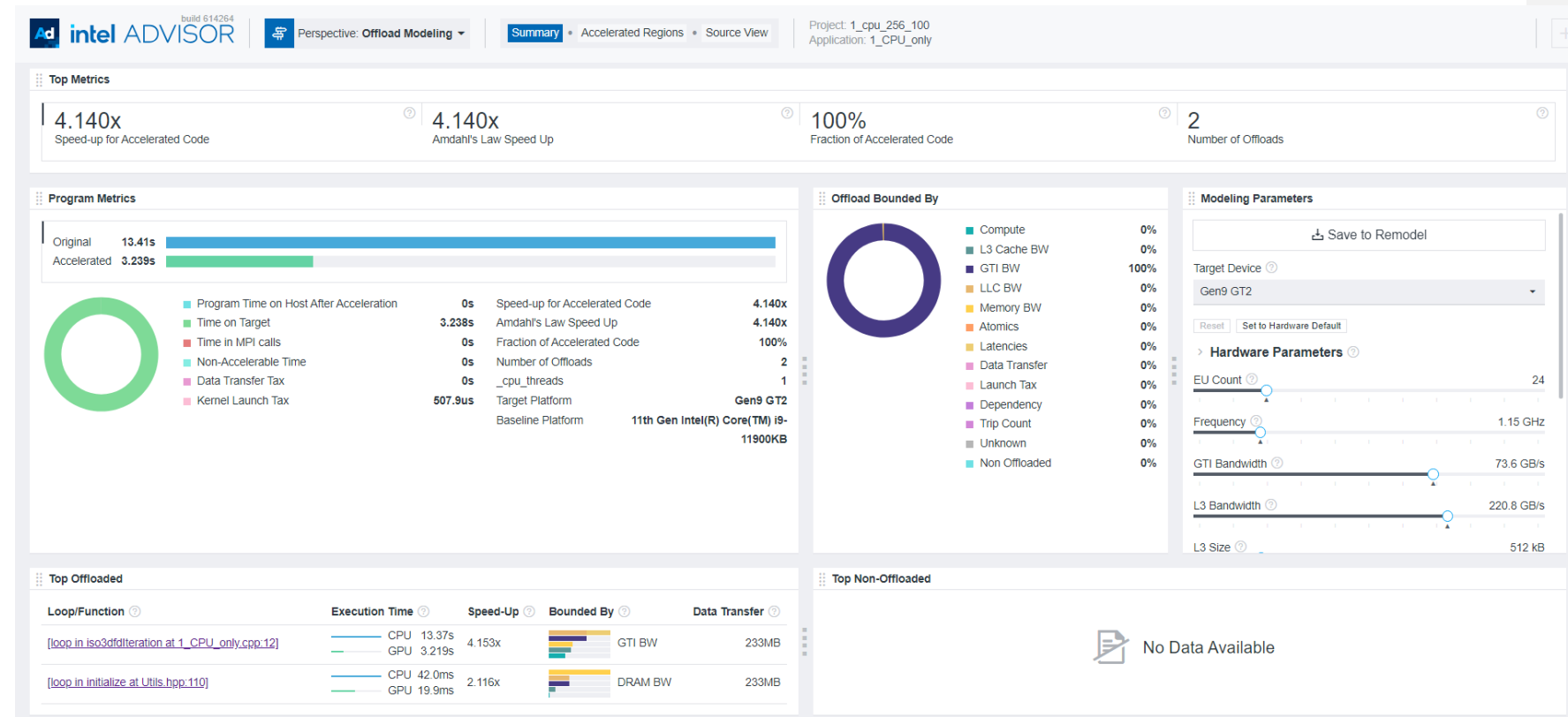
```
$ advisor --snapshot --project-dir=../../advisor/1_cpu --pack --cache-sources --cache-binaries -- ./offload_advisor_snapshot
```

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

build 614264

Perspective: **Offload Modeling**

Summary
Accelerated Regions
Source View

4.140x
4.140x
100%
2

Speed-up for Accelerated Cod...
Amdah's Law Speed U...
Fraction of Accelerated Cod...
Number of Offload...

Code Regions
Roofline

Loop/Function	Performance Issues	Measured			Basic Estimated Metrics				Estimated Bounded By			
		Time	Region	Iteration Space	Speed-Up	Time	Bounded By	Offload Summary	Why ...	Throughput	Taxes	Latencies
▼ [loop in iso3dfditeration at 1_CPU_only.cpp:12]	Code re...	13.37s		CC 100 TC 256	4.153x	3.219s 99.4%	GTI BW	Offloaded		GTI BW 1.956s LLC BW 1.956s	Launch Tax 502.5us All Taxes 502.5us	Load 1.263s
▼ [loop in iso3dfditeration at 1_CPU_only.cpp:12]		13.36s						Child loop				
▼ [loop in iso3dfditeration at 1_CPU_only.cpp:12]		13.34s						Child loop				
▶ [loop in initialize at Utils.hpp:110]	Code re...	42.0ms		CC 1 TC 272	2.116x	19.9ms 0.6%	DRAM BW	Offloaded		DRAM ... 19.8... LLC BW 6.6ms	Launch Tax 5.4us All Taxes 5.4us	Load 86.2us

Source
Top Down
Recommendations

Possible offloading
Confidence level: high
Based on the **Offload Modeling** results, this code region is potentially profitable to offload.

Code region is recommended for offloading
Confidence level: high
Based on the **Offload Modeling** results, this code region is potentially profitable to offload. Estimated relative speedup on the **Gen9 GT2** accelerator is **4.15** compared to the current platform. Consider using [Data Parallel C++ \(DPC++\)](#) or [OpenMP* Offload Programming Model](#) to offload this region to the target accelerator.

Example of using a DPC++ parallel for construct for offloading: ☺

```
...
    cgh.parallel_for<kernel>(N, [=](id<1> i) {
...

```

Example of using OpenMP target construct for offloading: ☺

```
...
#pragma omp target teams distribute parallel for map(to: matrixA, matrixB) map(from: matrixC) private(i, j, k)
...

```

Table of Contents:

Possible offloading

- Code region is recommended for offloading

Details
Data Transfer Esti

ESTIMATED SPEED-UP: **4.153X**

Estimated Time

3.219s

Measured Time

13.37s

BOUNDED BY: GTI BW

Compute	877.3ms
DRAM BW	1.236s
L3 BW	1.166s
LLC BW	1.956s
Load Latency	1.263s
Data Transfer Tax	0s
Kernel Launch Tax	502.5us
Estimated Time	3.219s

Intel, and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of their owners.

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

Top Metrics

94.193x
Speed-up for Accelerated Code

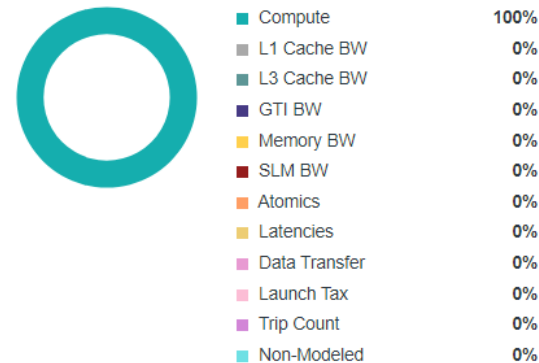
1
Number of Offloads

Program Metrics



Estimated Time on GPU	117.4ms	Speed-up for Accelerated Code	94.193x
Data Transfer Tax	18.6ms	Number of Offloads	1
Kernel Launch Tax	500.0us	Target Platform	XeHPC XT 512
		Baseline Platform	Intel (R) HD Graphics P630

Offload Bounded By



Modeling Parameters

Save to Remodel

Target Device: XeHPC XT 512

Reset Set to Hardware Default

Hardware Parameters

Frequency: 1.7 GHz

GTI Bandwidth: 1.74 TB/s

L1 Bandwidth: 55.71 TB/s

L1 Size: 32 MB

L3 Bandwidth: 6.96 TB/s

Top Offloaded

Kernel	Execution Time	Speed-Up	Bounded By	Data Transfer
iso3dfd(sycl: _V1::queue&.float*.float*.float*...	Baseline 12.86s Target 136.5ms	94.193x	Compute	483MB

Top Non-Offloaded

No Data Available

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
```

intel ADVISOR build 614264 Perspective: GPU Roofline Insights Summary GPU Roofline Regions Source View Project: 9_gpu_roofline_basic Application: 2_GPU_basic

24.31s Program Elapsed Time 12.86s GPU Time 0.02s Data Transfer Time 11.45s CPU Time

GPU

GFLOPS: 8.09 GINTOPS: 84.00 GTI Bandwidth: 11.39 GB/s
 GFLOP: 104.02 FP AI (GTI): 0.71 GINTOP: 1,080.61 INT AI (GTI): 7.38 GTI Traffic: 146.49 GB
 FPU Utilization: 65.7% EU Threading Occupancy: 98.2% EU IPC Rate: 1.67

CPU

GFLOPS: <0.01 GINTOPS: <0.01
 GFLOP: <0.01 FP AI: <0.01 GINTOP: 0.01 INT AI: 0.04
 Thread Count: 1

OP/S and Bandwidth

GPU ROOFLINE

Roofline parameters: Int32 Vector Add Peak: 328.18 GINTOPS, GTI Bandwidth: 76 GB/sec.

CPU ROOFLINE

Roofline parameters: Int32 Vector Add Peak: 95.66 GINTOPS, DRAM Bandwidth: 24.17 GB/sec.

This application is bounded by Compute (GINTOPS): 84.00 25% of 328.18 GINTOPS Int32 Vector Add Peak

Top Hotspots

Kernel	Elapsed Time	GFLOPS	GINTOPS	Global/Local	Active/Stalled/Idle, %	Function Call Site...	Self Elapsed Time...	Self GFLOPS	Self GINTOPS
iso3dfd(sycl::V1::queue&,...	12.86s	8.086	83.998	256 x 256 x 256/256 x 1 x 1	97.4/2.6/<0.1	[loop in __intel_av...	0.06s		0.0306722439599...
						[loop in initialize_at...	0.05s		0.0968246677217...
						[loop in livm::String...	0.01s		
						[loop in khricdVend...	0s		
						[loop in khricdOsDI...	0s		

Performance Characteristics

GPU

- EU Array Active: 97.4%
- EU Array Stalled: 2.6%
- EU Array Idle: 0.0%

CPU

- FPU Utilization: 65.7%
- Average GPU Execution Unit Utilization: 97.4%
- GPU L3 Bandwidth Bound: 23.2%
- Incoming GTI Bandwidth Bound: 14.9%
- Outgoing GTI Bandwidth Bound: 1.5%

CPU Time Breakdown

- Total CPU Time: 16.62s (100%)
- Time in 3 Vectorized Loops: 0.22s (1%)
- Time in Scalar Code: 16.40s (99%)

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
```

intel ADVISOR build 614264 Perspective: GPU Roofline Insights Summary GPU Roofline Regions Source View Project: 9_gpu_roofline_opt_good Application: 5_GPU_optimized

Program Metrics

6.55s Program Elapsed Time | 3.15s GPU Time | 0.01s Data Transfer Time | 3.41s CPU Time

GPU

GFLOPS: 33.07 | GINTOPS: 14.08 | GTI Bandwidth: 16.42 GB/s | GFLOPS: <0.01 | GINTOPS: <0.01

GFLOP: 104.02 | FP AI (GTI): 2.01 | GINTOP: 44.28 | INT AI (GTI): 0.86 | GTI Traffic: 51.63 GB | GFLOP: <0.01 | FP AI: <0.01 | GINTOP: 0.01 | INT AI: 0.02

FPU Utilization: 13.1% | EU Threading Occupancy: 85.0% | EU IPC Rate: 1.36 | Thread Count: 1

OP/S and Bandwidth

GPU ROOFLINE

This application is bounded by SLM Bandwidth: 123.70 61% of 202.47 GB/sec

CPU ROOFLINE

Top Hotspots

Kernel	Elapsed Time	GFLOPS	GINTOPS	Global/Local	Active/Stalled/Idle, %
iso3dfd(sycl_..._V1::queue&...	3.15s	33.069	14.076	256 x 256 x 16/16 x 8 x 1	43.7/56.3/<0.1

Function Call Site...

Function Call Site...	Self Elapsed Time...	Self GFLOPS	Self GINTOPS
loop_in_intel_av...	0.05s		0.0343048415801...
loop_in_initialize_at...	0.02s		0.2515649025749...
loop_in_intel_av...	<0.01s		0.3858495015962...
loop_in_initialize_at...	<0.01s		0.1017125676606...
loop_in_initialize_at...	0s		

Performance Characteristics

GPU

- EU Array Active: 43.7%
- EU Array Stalled: 56.3%
- EU Array Idle: 0.0%

FPU Utilization: 13.1%

Average GPU Execution Unit Utilization: 43.7%

GPU L3 Bandwidth Bound: 16.1%

Incoming GTI Bandwidth Bound: 19.5%

Outgoing GTI Bandwidth Bound: 6.3%

CPU

Total CPU Time: 5.77s 100%

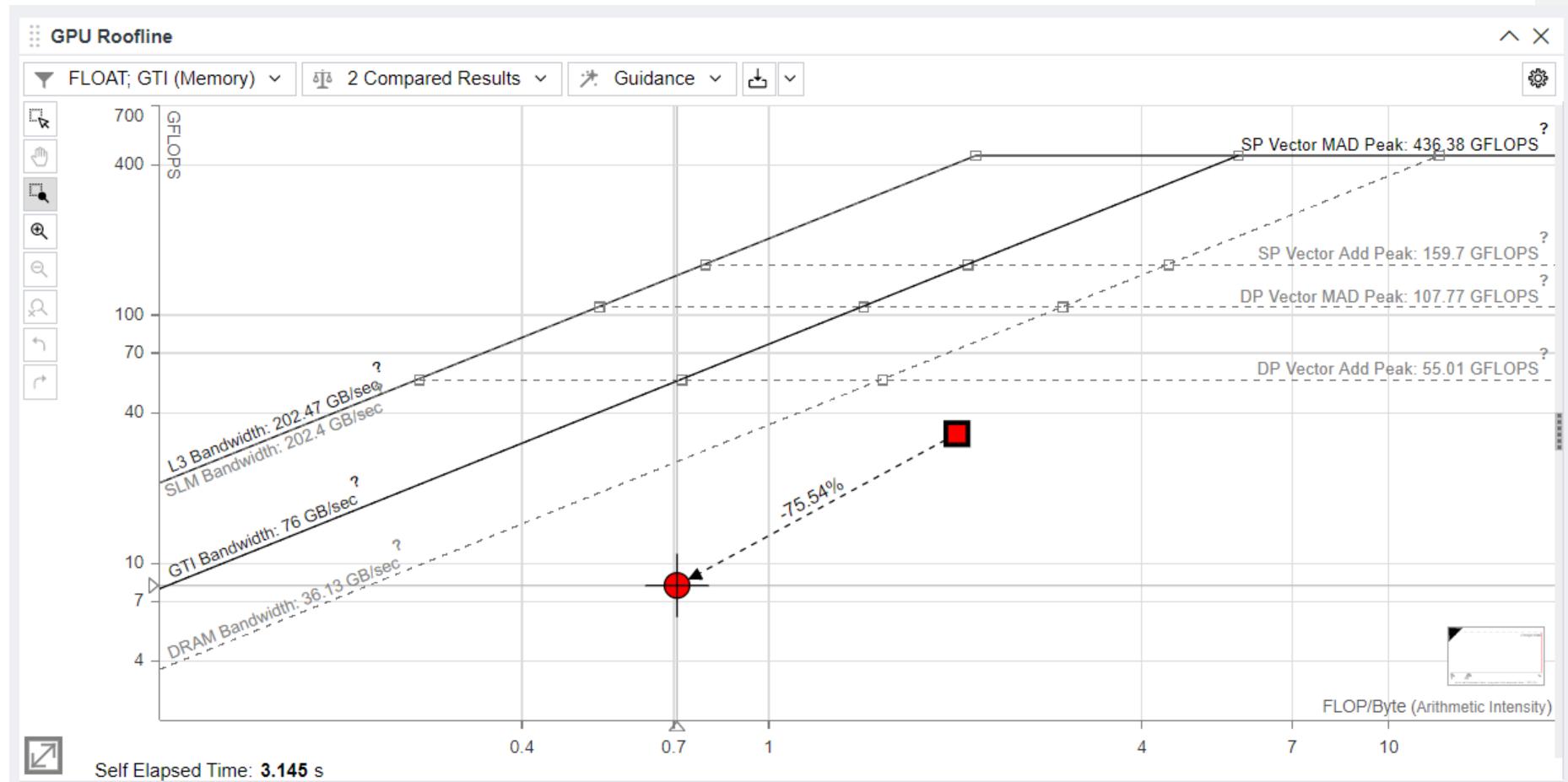
Time in 4 Vectorized Loops: 0.11s 2%

Time in Scalar Code: 5.66s 98%

Compare Rooflines

- Available in the client!

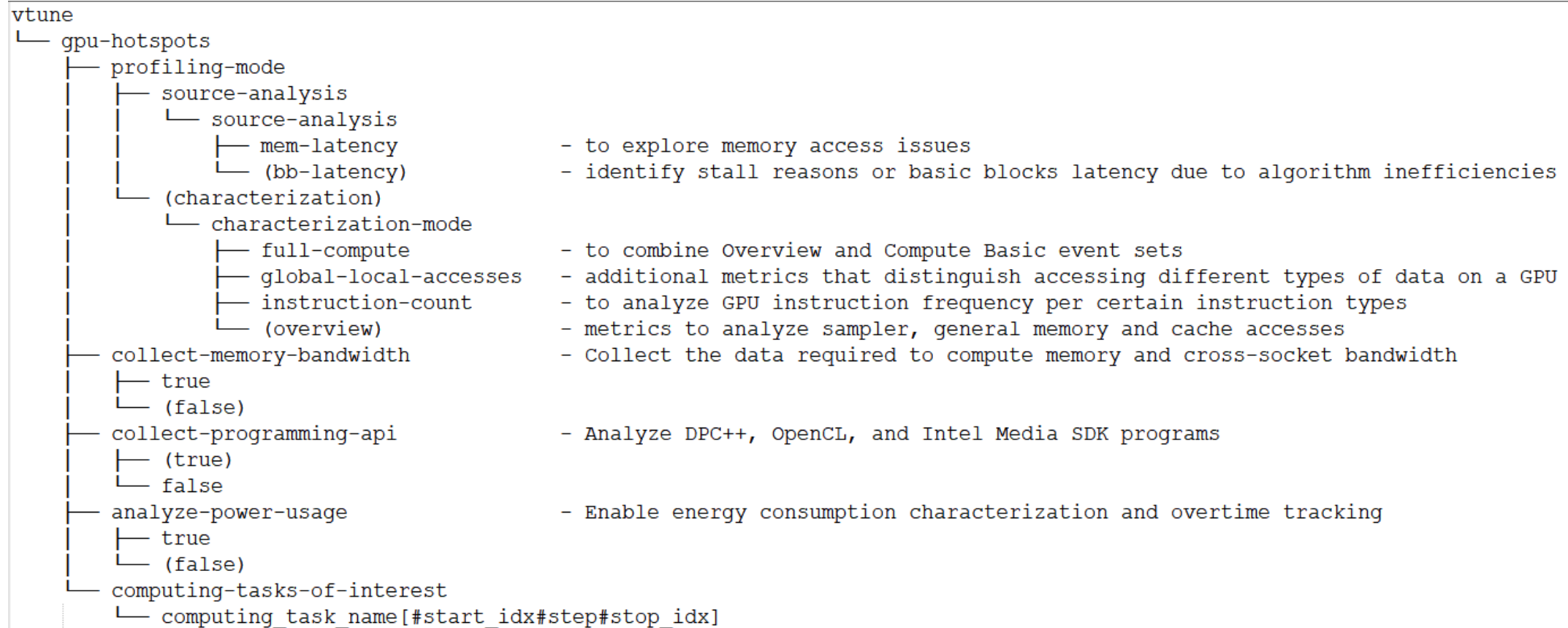
<https://www.intel.com/content/www/us/en/developer/articles/tool/oneapi-standalone-components.html#advisor>



Intel® VTune™ Profiler Exercise

HPC-Performance, GPU Offload, GPU Hotspots

Intel® VTune™ Profiler Analysis Types



+ 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`

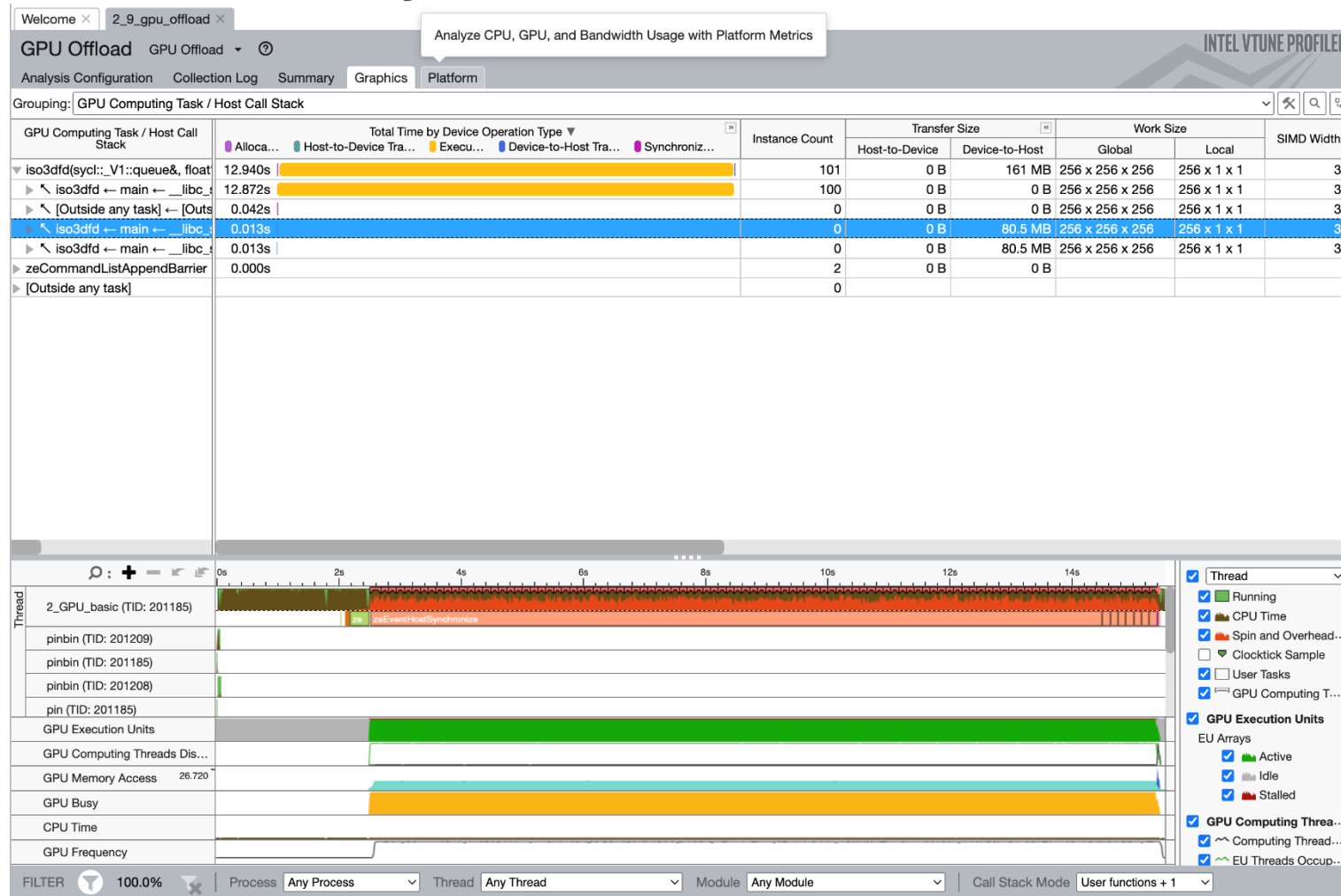
A URL will be printed by the above the command
3. Set up local port forwarding
 - Open another terminal to launch additional SSH sessions to enable port forwarding:
 - `ssh -L 8080:127.0.0.1:8080 devcloud`
 - `ssh -L 8080:127.0.0.1:8080 <compute-node from Step1>`
 - Copy the URL printed by Step 2 and 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

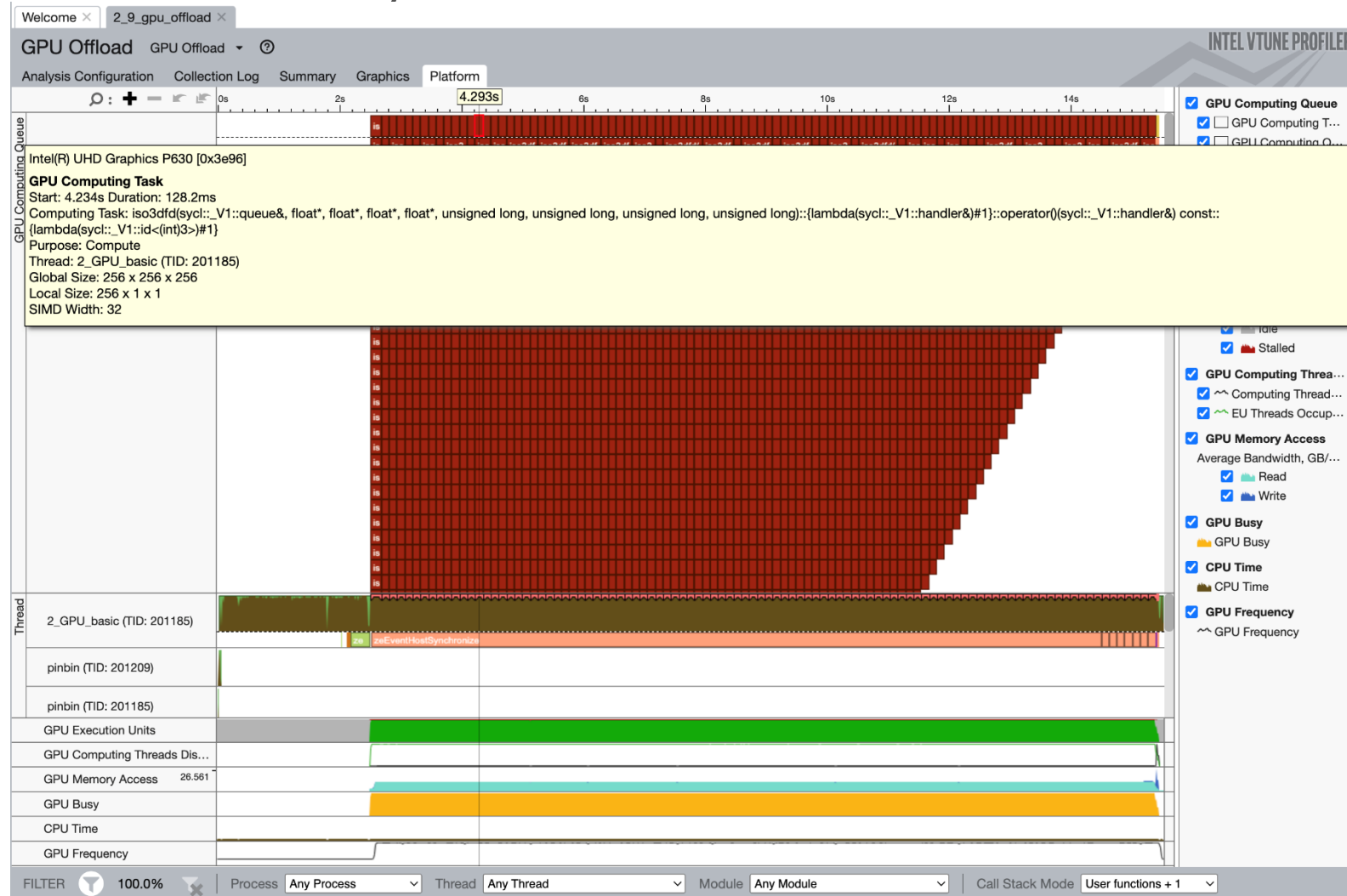
Intel® VTune™ Profiler GPU Offload Analysis

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



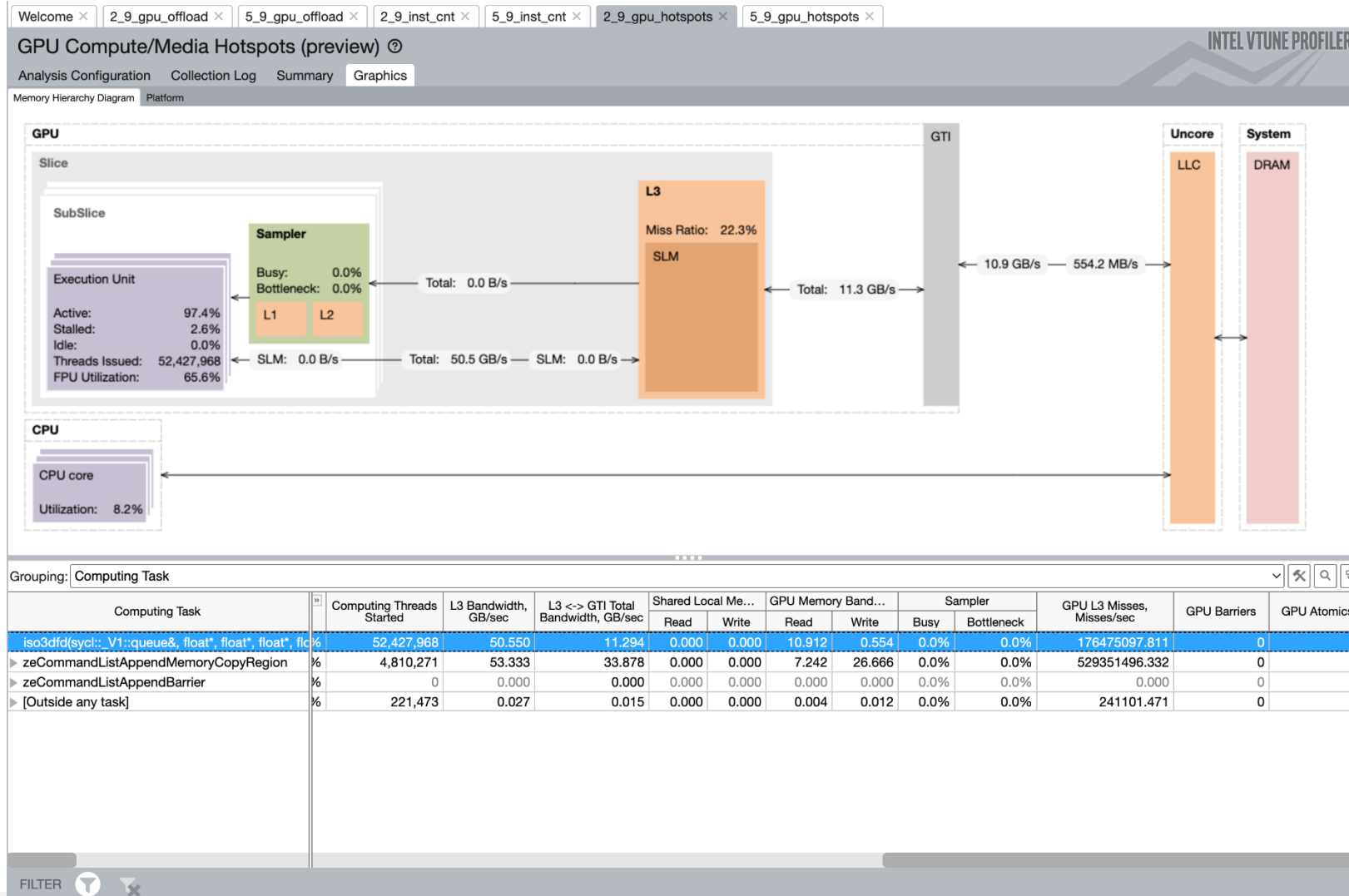
Intel® VTune™ Profiler GPU Offload Analysis

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



GPU Hotspots Analysis

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



Intel® VTune™ Profiler

GPU Hotspots Analysis

GPU Compute/Media Hotspots (preview) ⓘ

Analysis Configuration Collection Log Summary **Graphics**

Memory Hierarchy Diagram Platform



- Thread
- Running
- User Tasks
- Computing Task
- GPU Vector Engine
 - XVE Arrays
 - Active
 - Idle
 - Stalled
- GPU Computing Thre...
 - Computing Threa...
 - XVE Threads Occ...
- GPU XVE Pipelines: ...
 - FPU and EM Utiliz...
 - FPU and XMV Util...
- GPU XVE Instruction...
 - XVE Send pipelin...
 - XVE Send instruct...
 - XVE FPU instructi...

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*, fl...	12.836s	0.128s	100	0 B	2.373e+12
▶ iso3dfd(sycl::_V1::queue&, float*, float*, float*, fl...					1.370e+12
▶ _ZN4sycl3_V16detail13dim_loop_implJLm0EL...					6.711e+9
▶ _ZN4sycl3_V16detail13dim_loop_implJLm0EL...					9.555e+11
▶ _ZN4sycl3_V16detail13dim_loop_implJLm0EL...					3.355e+9
▶ _ZN4sycl3_V16detail13dim_loop_implJLm0EL...					3.691e+10
▶ zeCommandListAppendMemoryCopyRegion	0.023s	0.011s	2	161 MB	
▶ zeCommandListAppendBarrier	0.000s	0.000s	2	0 B	

FILTER | Call Stack Mode | Inline Mode

Source level in-kernel profiling

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

The screenshot shows the Intel VTune Profiler interface with the 'Source' view selected. The table below represents the data shown in the 'Estimated GPU Cycles' columns.

Source Line	Source	Estimated GPU Cycles: Total	Estimated GPU Cycles: Self
35	accessor vel_acc(vel_buf, h, read_only);		
36	accessor coeff_acc(coeff_buf, h, read_only);		
37			
38	// Send a SYCL kernel(lambda) to the device for parallel execution		
39	// Each kernel runs single cell		
40	h.parallel_for(kernel_range, [=](id<3> idx) {	0.0%	8.389e+8
41	// Start of device code		
42	// Add offsets to indices to exclude HALO		
43	int i = idx[0] + kHalfLength;	0.1%	2.517e+9
44	int j = idx[1] + kHalfLength;	0.1%	1.678e+9
45	int k = idx[2] + kHalfLength;	0.1%	1.678e+9
46			
47	// Calculate values for each cell		
48	float value = prev_acc[i][j][k] * coeff_acc[0];	1.7%	4.066e+10
49	#pragma unroll(8)		
50	for (int x = 1; x <= kHalfLength; x++) {		
51	value +=	0.6%	1.342e+10
52	coeff_acc[x] * (prev_acc[i][j][k + x] + prev_acc[i][j][k - x] +	10.3%	2.438e+11
53	prev_acc[i][j + x][k] + prev_acc[i][j - x][k] +	19.5%	4.620e+11
54	prev_acc[i + x][j][k] + prev_acc[i - x][j][k]);	20.0%	4.754e+11
55	}		
56	next_acc[i][j][k] = 2.0f * prev_acc[i][j][k] - next_acc[i][j][k] +	1.3%	3.081e+10
57	value * vel_acc[i][j][k];	1.1%	2.557e+10
58	// End of device code		
59	});		
60	});		
61			
62	// Swap the buffers for always having current values in prev buffer		
63	std::swap(next_buf, prev_buf);		
64	}		
65	};		
66			
67	int main(int argc, char* argv[]) {		
68	// Arrays used to update the wavefield		
69	float* prev;		
70	float* next;		
71	// Array to store wave velocity		
72	float* vel;		
73			
74	// Variables to store size of grids and number of simulation iterations		

Intel® VTune™ Profiler

Comparisons of two profiles

Welcome × 5_9_inst_cnt × 2_9_src_anal_mem × 5_9_src_anal_mem × 2_9_inst_cnt × 2_9_gpu_hotspots - 5_9_gpu_hotspots ×

GPU Compute/Media Hotspots (preview) ⓘ

Analysis Configuration Collection Log Summary

Elapsed Time ⓘ: 15.396s - 5.565s = 9.831s
GPU Time ⓘ: 12.903s - 3.186s = 9.717s

EU Array Stalled/Idle ⓘ: 2.6% - 56.5% = -53.9% of Elapsed time with GPU busy
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 ⓘ: 23.2% - 72.8% = -49.6% of peak value
- Sampler Busy ⓘ: Not changed, 0.0% of peak value

FPU Utilization ⓘ: 61.8% - 10.5% = 51.3% of Elapsed time with GPU busy
Identify computing tasks with high utilization of the floating point execution units.

- Hottest GPU Computing Tasks with High FPU Utilization**
This section lists the most active computing tasks that ran on the GPU heavily using the floating point execution units. Tasks in the table are sorted by the Total Time.
No data to show. The collected data is not sufficient.

Bandwidth Utilization Histogram
Explore bandwidth utilization over time using the histogram and identify memory objects or functions with maximum contribution to the high bandwidth utilization.

Bandwidth Domain:

- Bandwidth Utilization Histogram**
This histogram displays the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and Interconnect bandwidth.

Bandwidth Utilization (GB/sec)	Elapsed Time (s)
0	~2.5
~50	~12.0
160	~3.5

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>

intel®