

# Profiling Large-Scale Heterogeneous Applications with Intel® VTune™ Profiler

Xiao Zhu, Rupak Roy– Technical Consulting Engineer (Intel Corporation)

# Agenda

- VTune Profiling Capabilities
- Latest Features in Vtune
- Deep Dive into VTune Features
  - CPU Profiling
  - GPU Profiling
- GPU Roofline

# Optimize Performance

## Intel® VTune™ Profiler

### Get the Right Data to Find Bottlenecks

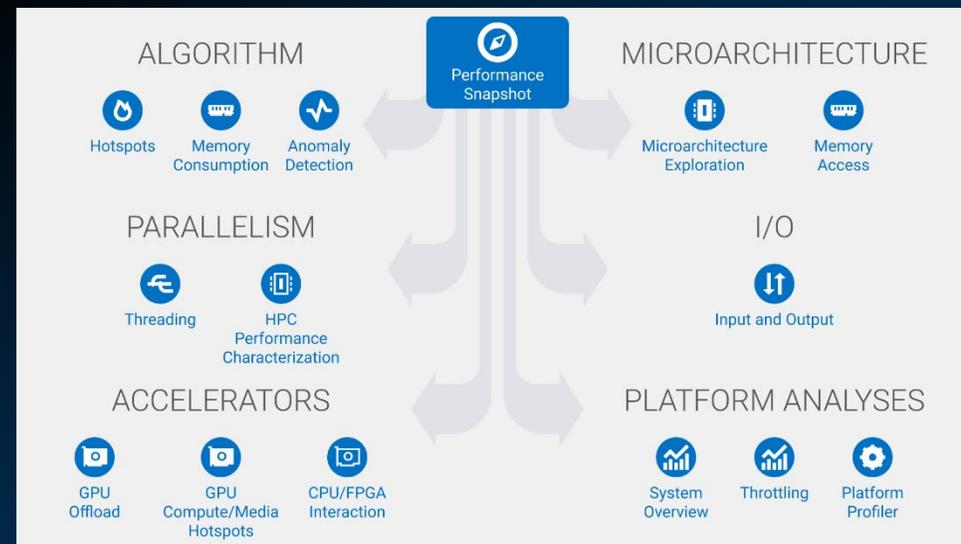
- A suite of profiling for CPU, GPU, FPGA,, memory, cache, storage, offload, power...
- Application or system-wide analysis
- DPC++, C, C++, Fortran, Python\*, Go\*, Java\*, or a mix
- Linux, Windows, FreeBSD, Android, Yocto and more
- Containers and VMs

### Analyze Data Faster

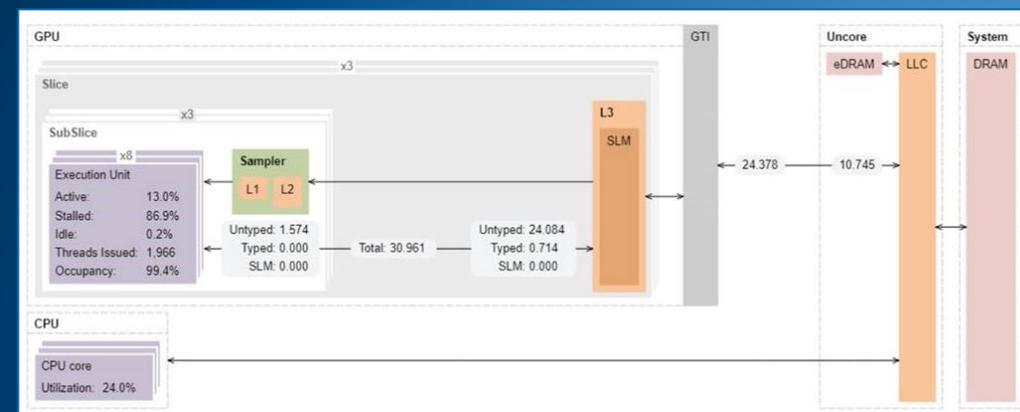
- Collect data HW/SW sampling and tracing w/o re-compilation
- See results on your source, in architecture diagrams, as a histogram, on a timeline...
- Filter and organize data to find answers

### Work Your Way

- User interface or command line
- Profile locally and remotely

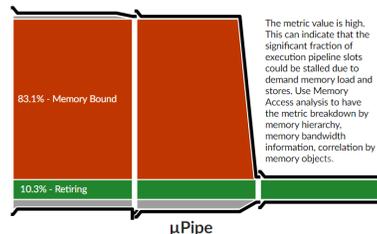
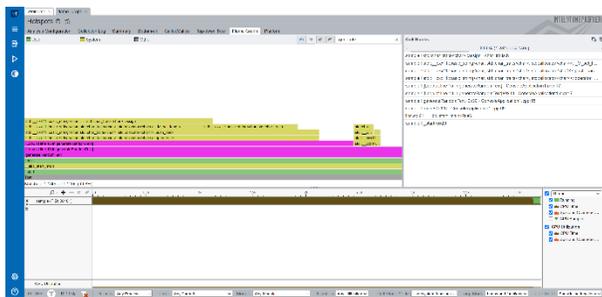


Source	Assembly	GPU Instructions Executed by Instruction T...
158	<code>dx = ptr[j].pos[0] - ptr[i].pos[0]</code>	75,002,500
159	<code>dy = ptr[j].pos[1] - ptr[i].pos[1]</code>	12,500,000
160	<code>dz = ptr[j].pos[2] - ptr[i].pos[2]</code>	12,500,000



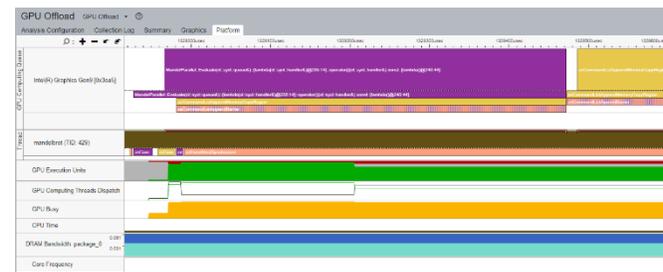
# Rich Set of Profiling Capabilities

## 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

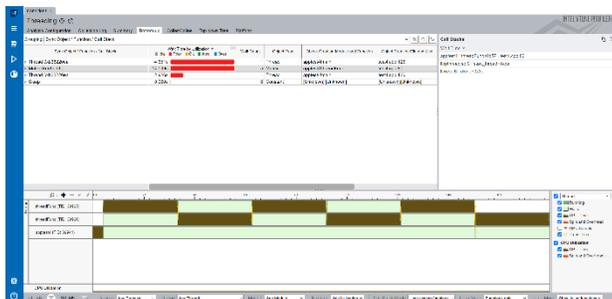
- ✓ Hotspots
- ✓ Anomaly Detection
- ✓ Memory Consumption

### Microarch.&Memory Bottlenecks

- ✓ Microarchitecture Exploration
- ✓ Memory Access

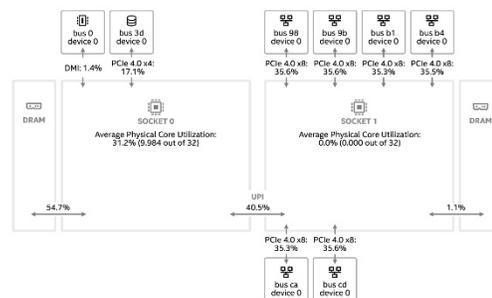
### Accelerators / xPU

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



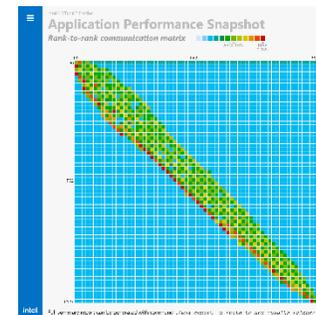
### Parallelism

- ✓ Threading
- ✓ HPC Performance Characterization



### Platform

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



### Multi-Node

- ✓ Application Performance Snapshot

# What's New in Intel® VTune™ Profiler

## 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

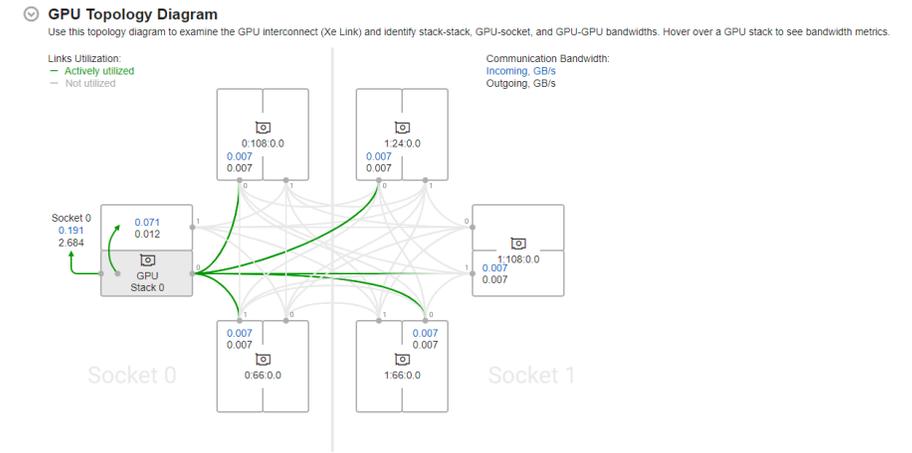
- Get visibility into XeLink cross-card traffic for issues such as stack-to-stack traffic, throughput and bandwidth bottlenecks. Identify imbalances of traffic between CPU and GPU through a GPU topology diagram.
- Identify the reasons of the stalls in Xe Vector Engines (XVEs), formerly known as Execution 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

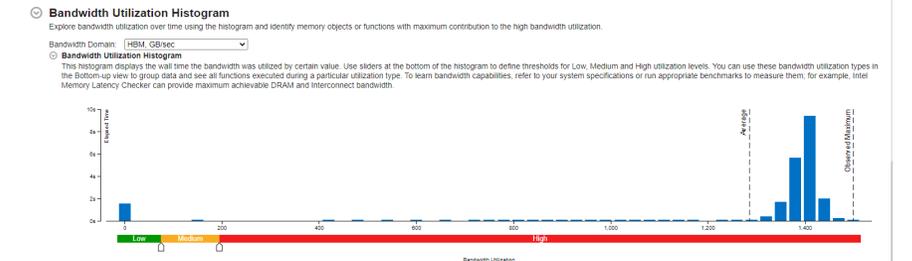
- Identify and optimize performance hotspots of Python code, now supporting Python 3.9.\*.

## 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.



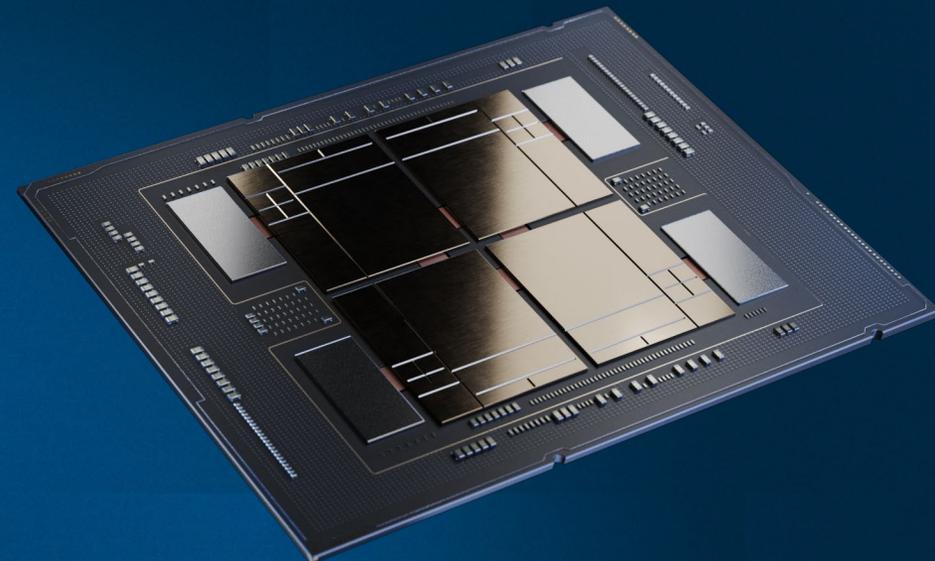
Cross-card, stack-to-stack, and card-to-socket bandwidth are presented on GPU Topology Diagram.



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



# Only x86 CPU with High Bandwidth Memory



## Memory modes

 HBM	<b>64GB</b> HBM2e	up to <b>112.5MB</b> shared LLC	<b>DDR5</b> 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

### 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

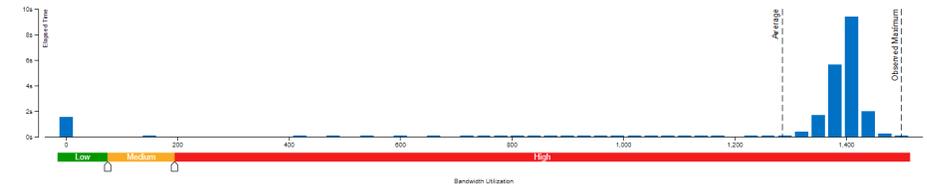
### Bandwidth Utilization Histogram

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

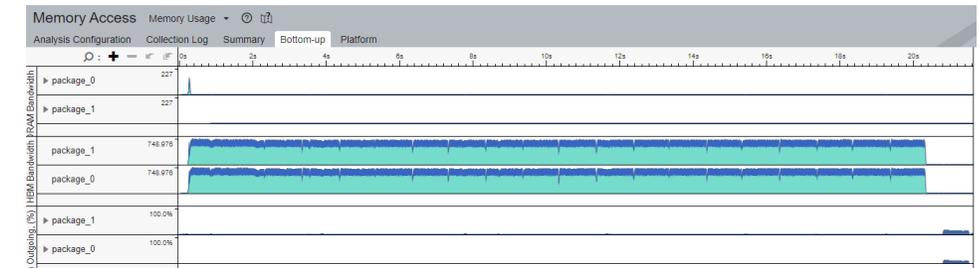
Bandwidth Domain:

#### 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.



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



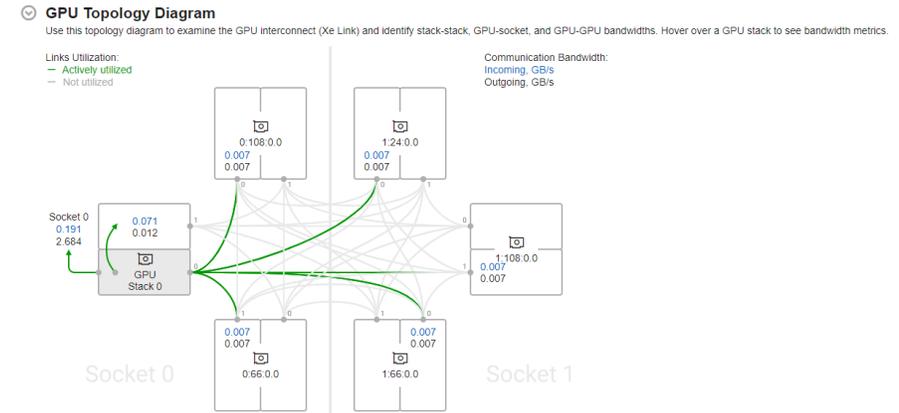
The workload performance in various HBM modes can be evaluated by running the collection in each mode and analyzing the bandwidth as described above.

# Get Visibility into Xe Link Cross-card Traffic

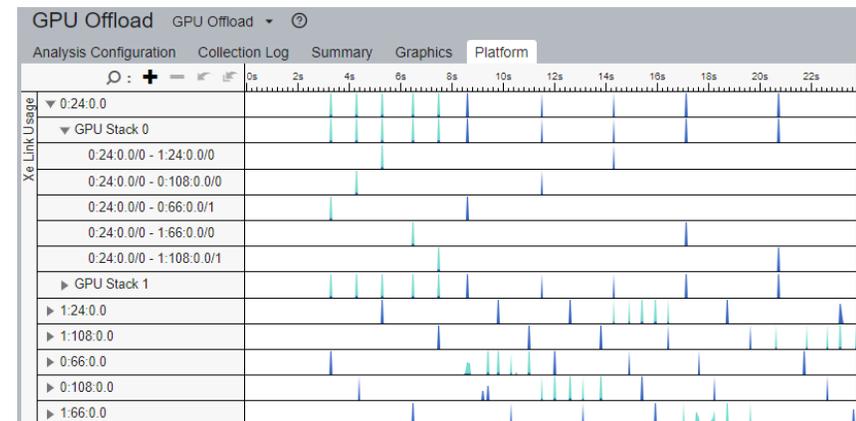
## Intel® VTune™ Profiler

### 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.

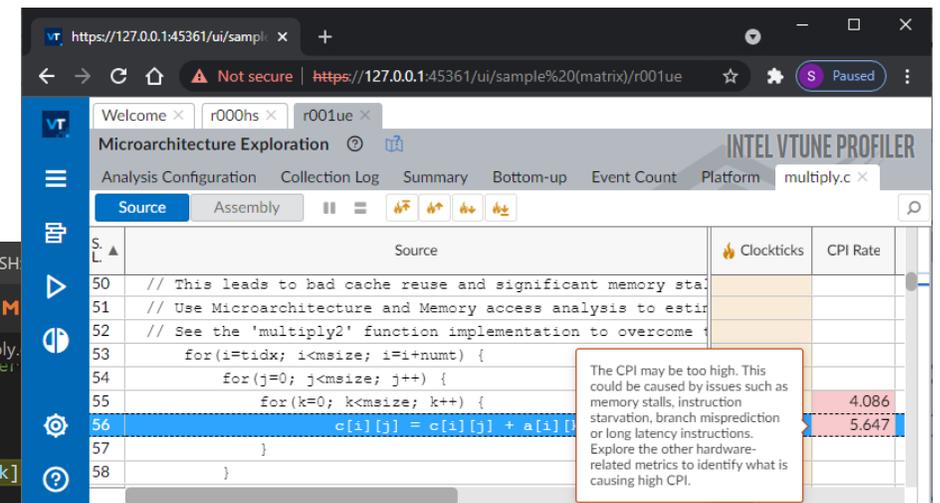
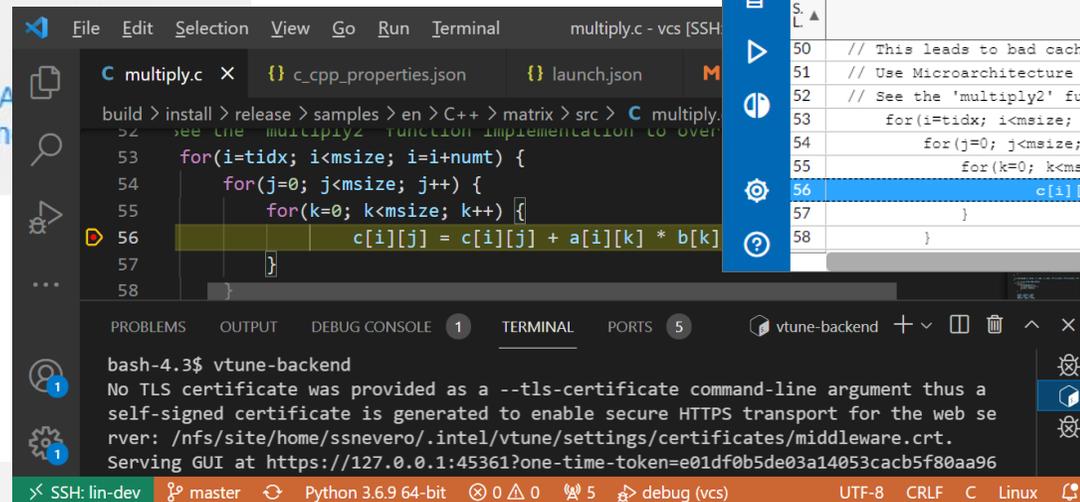
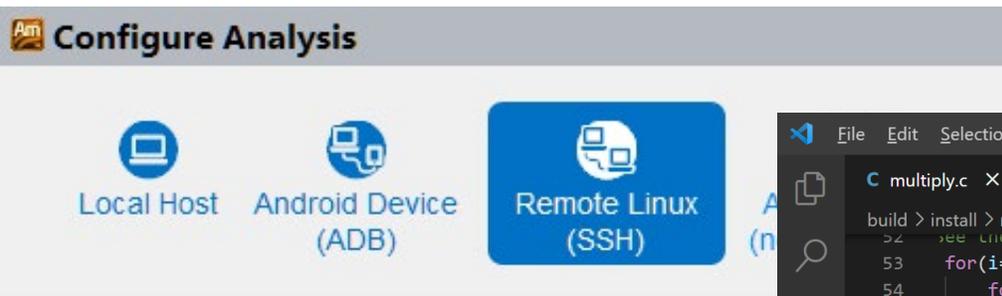
# Access Intel® VTune™ Profiler via web browser

## Interactive analysis

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

## Command line analysis

- 1) Run command line remotely on Linux\* target



# Command Line Interface

Automate analysis

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

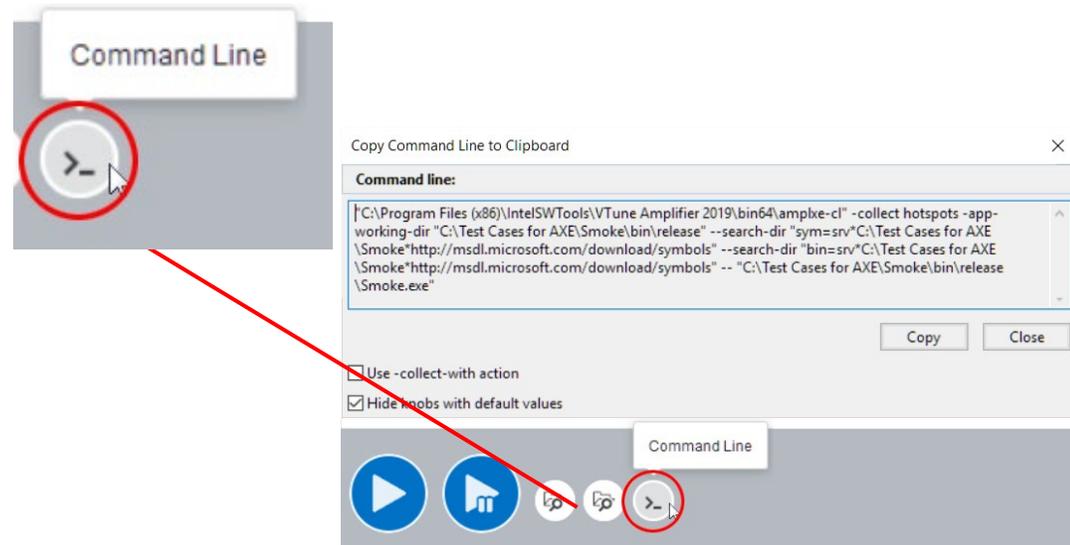
Help:

`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>
```

# Custom Analysis with VTune Profiler

The image shows the Intel VTune Profiler interface with three main sections highlighted by red boxes and labeled as Step 1, Step 2, and Step 3.

**Step 1:** The 'GPU Compute/Media Hotspots (preview)' analysis is selected in the 'ALGORITHM' section of the 'HOW' panel.

**Step 2:** The 'GPU Compute/Media Hotspots (preview)' analysis is selected in the 'HOW' panel, and the 'Customize a copy of the selected analysis.' button is visible.

**Step 3:** The 'GPU events selection' table is shown, with several events checked. The 'allow-multiple-runs' checkbox is also checked.

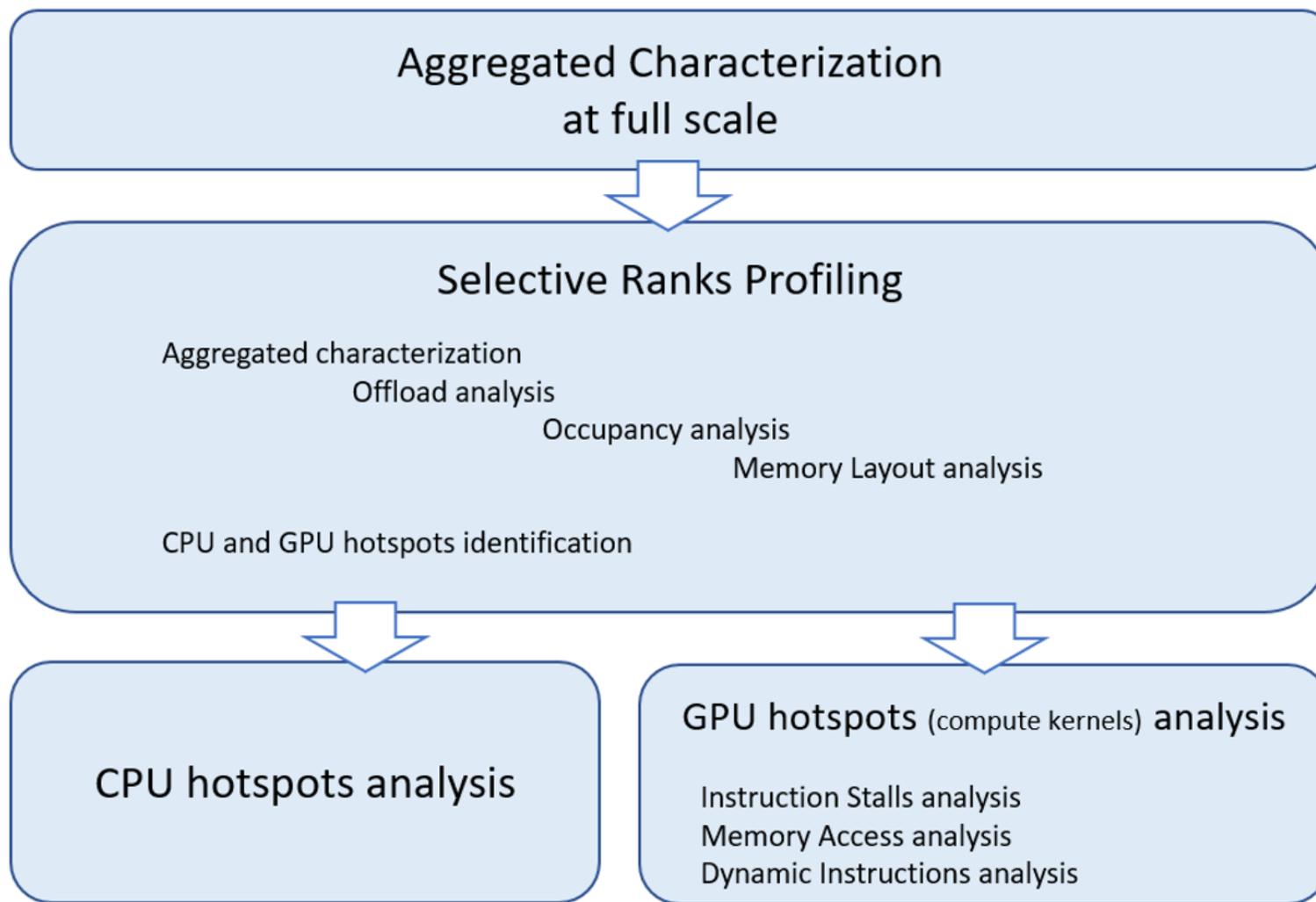
**GPU events selection table:**

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...

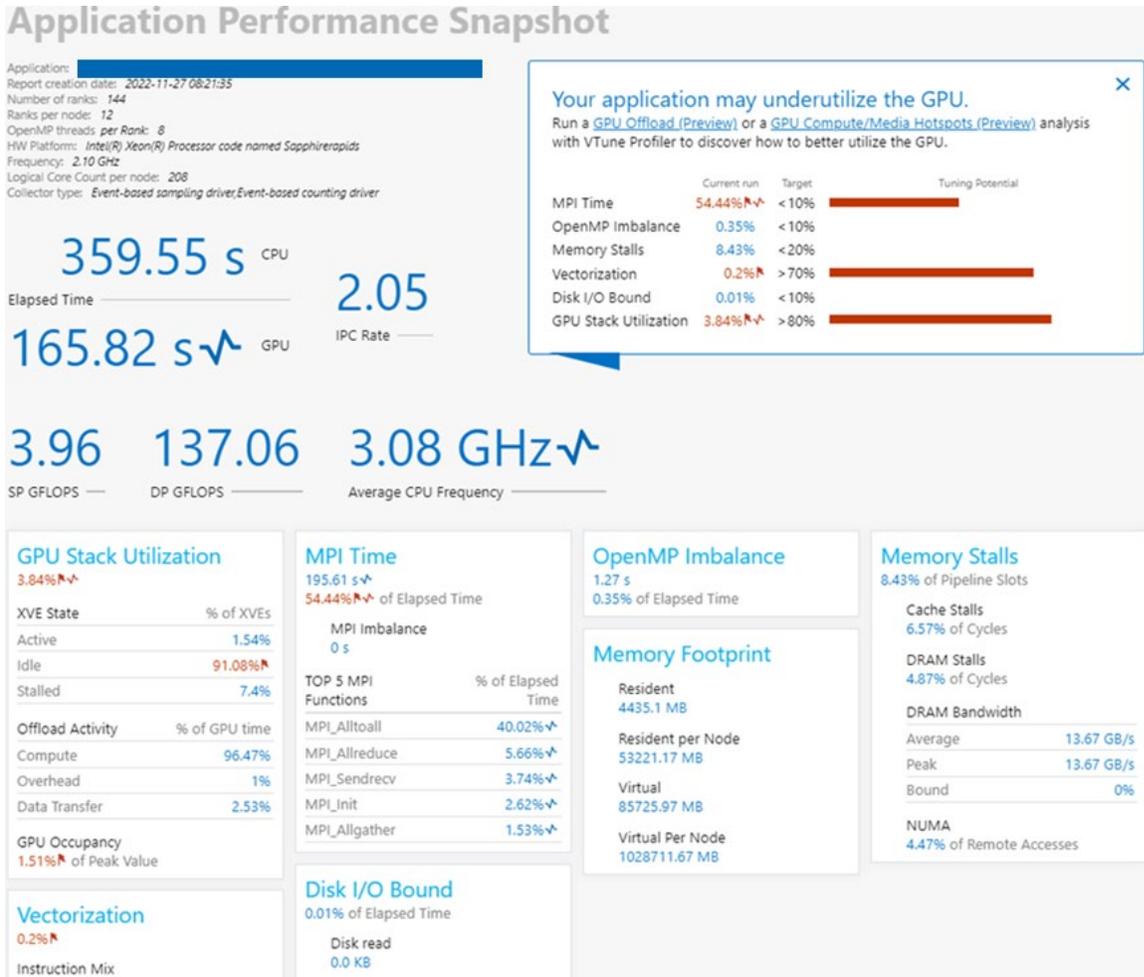
**Computing task of interest table:**

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

# General strategy



# 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>
```

# 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%)**
  - Top Serial Hotspots (outside parallel regions)
- 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

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was assumed to have no runtime overhead.

OpenMP Region	OpenMP Potential Gain (s)	OpenMP Potential Gain (%)	OpenMP Region Time (s)
[_Z22Iso3dfdVerifyIterationPIS_S_S_iiimmm.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 (s)	% of FP Ops	FP Ops: Packed	FP Ops: Scalar	Vector Instruction Set
[Loop at line 38 in _Z22Iso3dfdVerifyIterationPIS_S_S_iiimmm.DIR.OMP.PARALLEL.2]	532.905s	23.3%	100.0%	0.0%	SSE(128); SSE2(128)
[Loop at line 34 in _Z22Iso3dfdVerifyIterationPIS_S_S_iiimmm.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 (s)	Percentage of Elapsed Time	Data Transfer (s)	Overhead (s)	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>
```

# Selective Profiling

1. Use Multiple Program Multiple Data MPI run and apply VTune Profiler profiling for the ranks of interest.

```
$ export VTUNE_CL=vtune -collect hpc-performance -trace-mpi -result-dir my_result
```

```
$ mpirun -host myhost1 -ppn 8 -n 7 <app>: -n 1 $VTUNE_CL -- <app> :-n 7 <app>: -n 1 $VTUNE_CL -- <app>
```

2. And if you are interested in a particular rank (for example, an outlier rank defined by APS), it is recommended to write a launch script checking the rank number and pass the script to mpirun. For example, it can look like:

```
#!/bin/bash

if [ $PMI_RANK == 67 ]; then
    # Rank = 67 is chosen for vtune collection
    $VTUNE_CL -- <app>
else
    # all other ranks just run the application
    <app>
fi
```

# 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.

The screenshot displays the Intel VTune Profiler interface for Hotspots Analysis. The top navigation bar includes 'Analysis Configuration', 'Collection Log', 'Summary', 'Bottom-up', 'Caller/Callee', 'Top-down Tree', and 'Platform'. The 'Summary' view shows:

- Elapsed Time:** 133.634s
- CPU Time:** 472.871s
- Total Thread Count:** 5
- Paused Time:** 0s

The 'Top Hotspots' section lists the most active functions. The table below shows the top hotspot:

Function	Module	CPU Time
<a href="#">multiply1</a>	matrix.exe	472.573s

A 'Hotspots Insights' callout box provides guidance: '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.'

The bottom screenshot shows the 'Bottom-up' view of the 'multiply1' hotspot. The table below details the CPU time breakdown:

Function / Call Stack	CPU Time			Module	Function (Full)	Source File
	Effective Time	Spin Time	Overhead Time			
<a href="#">_intel_avx_rep_memset</a>	0.012s	0s	0s	libintlc.so.5	_intel_avx_rep_memset	
<a href="#">__printf</a>	0.008s	0s	0s	libc.so.6	__printf	printf.c
<a href="#">matrix_multiply</a>	13.960s	0s	0s	MatrixMultiplication_icc	matrix_multiply	MatrixMultiplication...

# What's Using All The Memory?

## Memory Consumption Analysis

### See What Is Allocating Memory

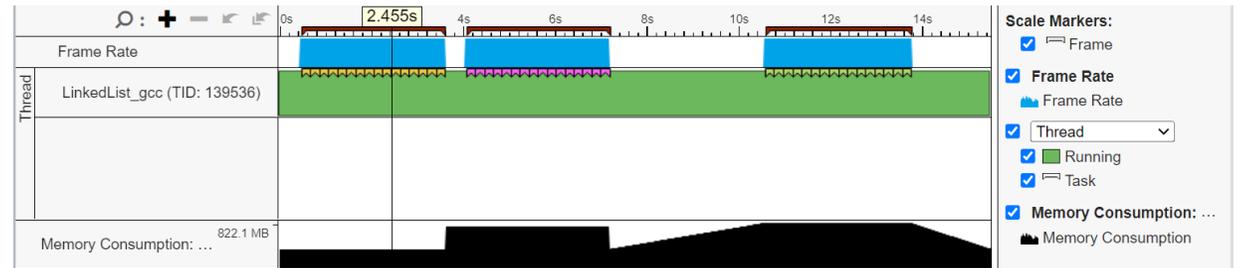
- Lists top memory consuming functions
- memory consumption distribution over time.
- View source to understand cause
- Filter by time using the memory consumption timeline
  - Focus on the peak values on the Timeline pane
- Introduce additional overhead due to instrumentation .

### Top Memory-Consuming Functions

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

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

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

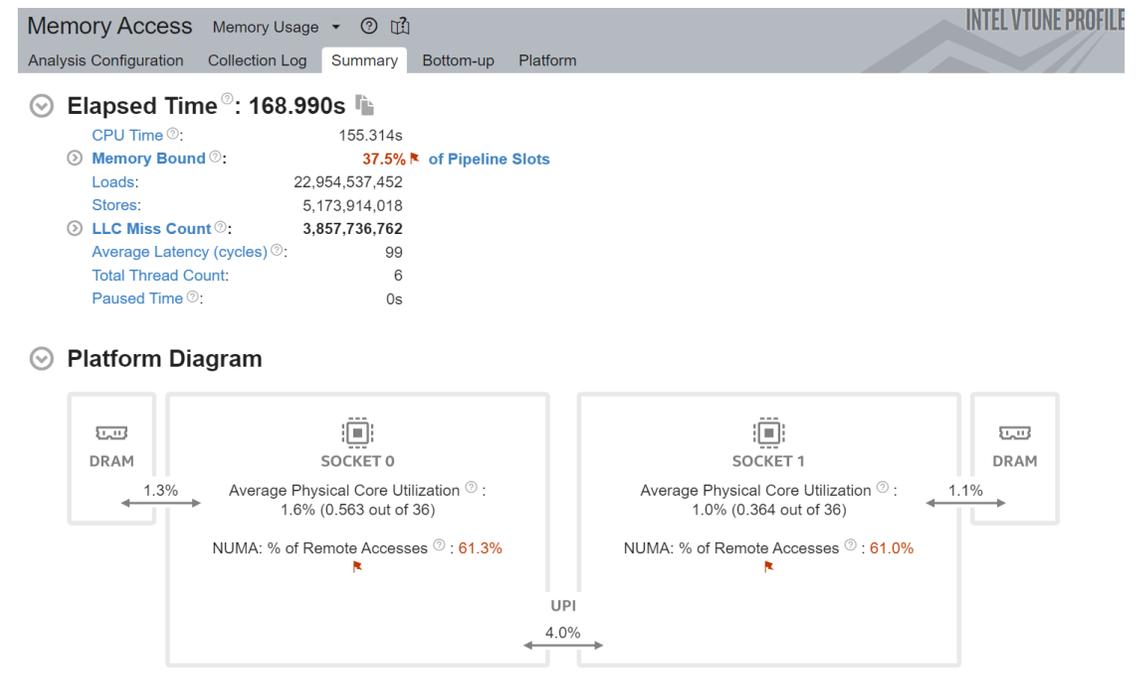


Native language support is not currently available for Windows\*

# Optimize Memory Access

## Memory Access Analysis

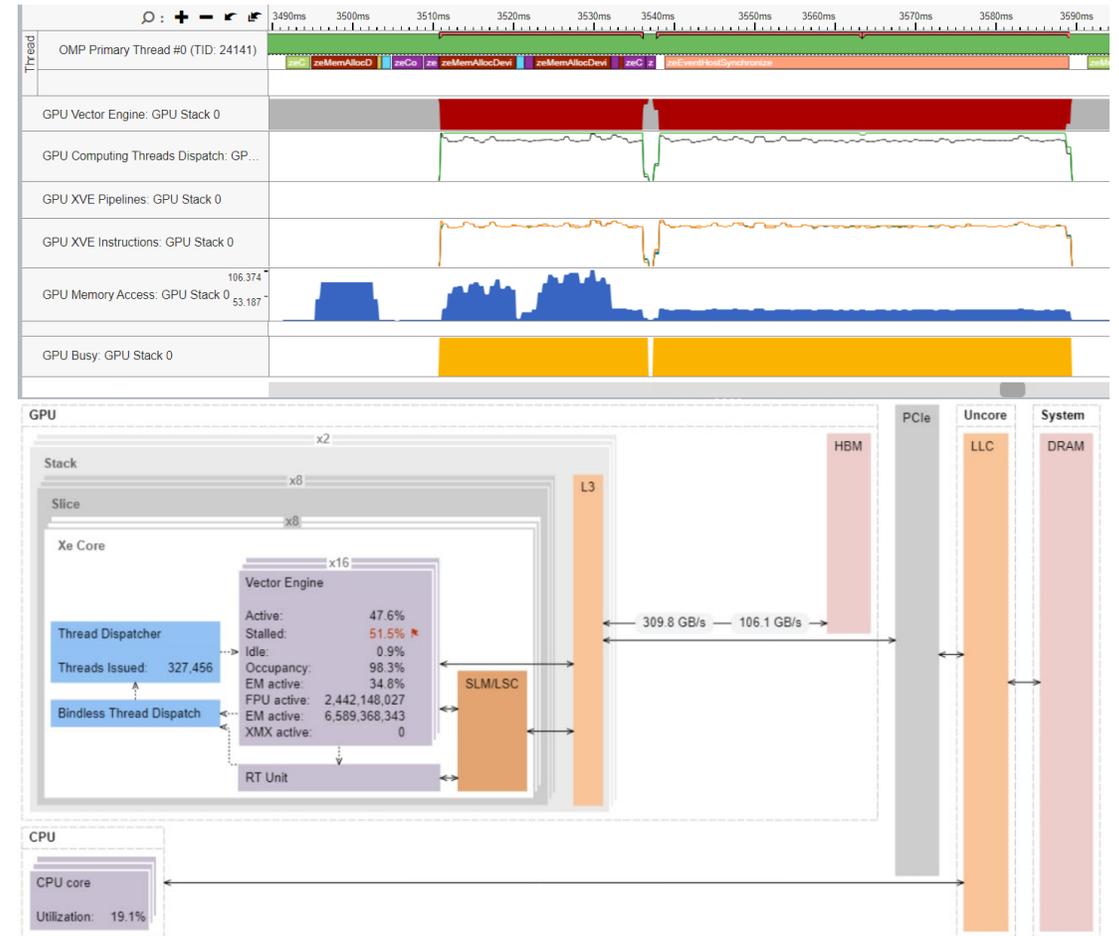
- Tune data structures for performance
  - Attribute cache misses to data structures (not just the code causing the miss)
  - Support for custom memory allocators
  - Shows average load latency in cycles
- Optimize NUMA latency & scalability
  - Auto detect max system bandwidth
  - Detects inter-socket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



# Intel® VTune™ Profiler

## Profile GPU Performance

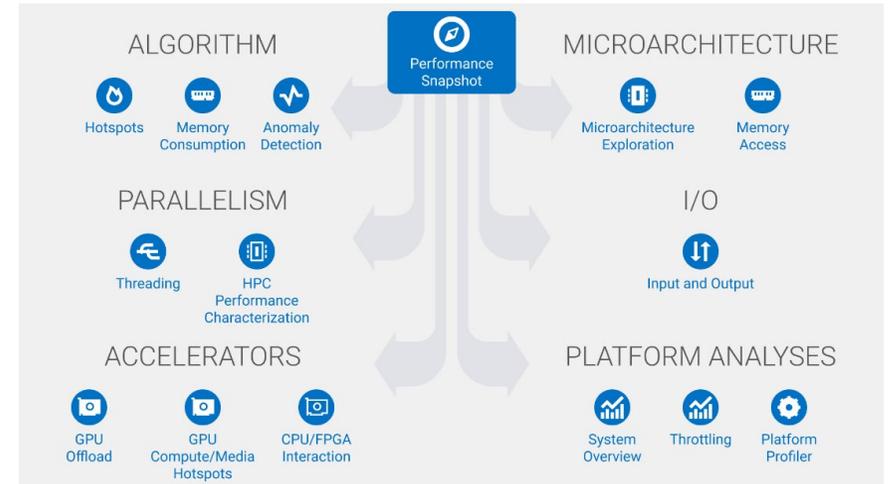
- Explicit support of **DPC++**, **DirectX**, **Intel® Media SDK**, **OpenCL™**, and **OpenMP-offload** software technology
- **Multi-GPU systems analysis**
- **GPU Offload cost profiling**
  - CPU vs GPU boundness
  - Offload overhead & host-to-device traffic, GPU compute vs data transfer
  - GPU utilization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

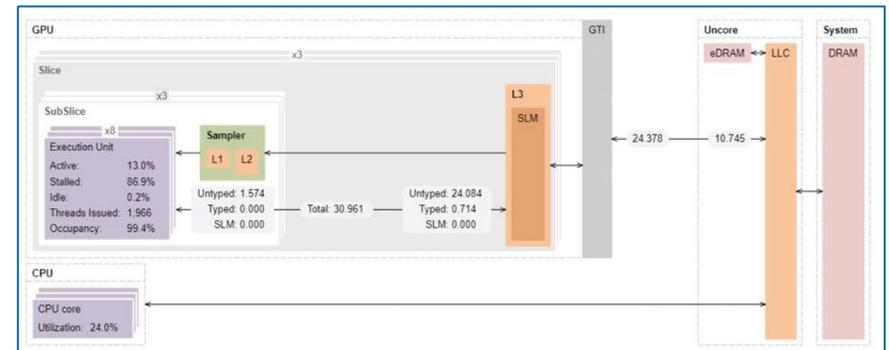


The screenshot displays a source code editor with the following code snippets:

```
158 dx = ptr[j].pos[0] - ptr[i].pos[0],
159 dy = ptr[j].pos[1] - ptr[i].pos[1],
160 dz = ptr[j].pos[2] - ptr[i].pos[2],
```

Execution statistics for these lines are shown in the table below:

Line	Source	GPU Instructions Executed by Instruction T...	Control Flow	Send & Wait	Int32 & SP Float	Int64 & DP Float	Other
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

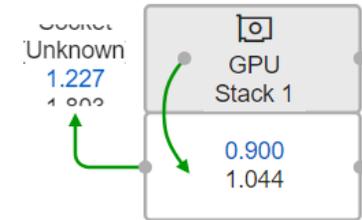
- CPU or GPU bound?
- GPU Utilization for OpenMP regions/SYCL kernels
- EU/XVEs efficiency (Active, Stalled, Idle)
- Offload Time characterization
  - Compute
  - Data Transfer
  - Overhead

### GPU Topology Diagram

Use this topology diagram to examine the GPU interconnect (Xe Link) and identify stack-stack

Links Utilization:  
 — Actively utilized  
 — Not utilized

Communication Bandwidth:  
 Incoming, GB/s  
 Outgoing, GB/s



Welcome x r000gh x r001go x

GPU Offload GPU Offload

Analysis Configuration Collection Log Summary Graphics

**Recommendations**

**GPU Time, % of Elapsed time: 6.7%**  
GPU utilization is low. Switch to the [Graphics view](#) for in-de

**XVE Array Stalled/Idle: 41.5%**  
GPU metrics detect some kernel issues. Use [GPU Comp](#)

**Elapsed Time<sup>Ⓢ</sup>: 2.495s**

**GPU Topology Diagram**

**Hottest Host Tasks**

**Hottest GPU Computing Tasks**  
This section lists the most active computing tasks running on the GPU, sorted by the Total Time. Focus on the computing tasks flagged as performance-critical.

Computing Task	Total Time <sup>Ⓢ</sup>	Execution Time <sup>Ⓢ</sup>	% of Total Time <sup>Ⓢ</sup>	SIMD Width <sup>Ⓢ</sup>	Peak XVE Threads Occupancy <sup>Ⓢ</sup>	XVE Threads Occupancy <sup>Ⓢ</sup>	SIMD Utilization <sup>Ⓢ</sup>
Iso3dfdDevice(sycl::_V1::queue&, float*, float*, float*, float*, unsigned long, unsigned long, unsigned long, unsigned long, unsigned long, unsigned long, unsigned int)::(lambda(#1))::operator()<sycl::_V1::handler>(, signed char) const::(lambda(#1))	0.437s	0.325s	74.3%	32	100.0%	94.1%	100.0%
Iso3dfdDevice(sycl::_V1::queue&, float*, float*, float*, float*, unsigned long, unsigned long, unsigned long, unsigned long, unsigned long, unsigned long, unsigned int)::(lambda(#1))::operator()<sycl::_V1::handler>(, signed char) const::(lambda(#2))	0.396s	0.325s	82.1%	32	100.0%	94.7%	100.0%

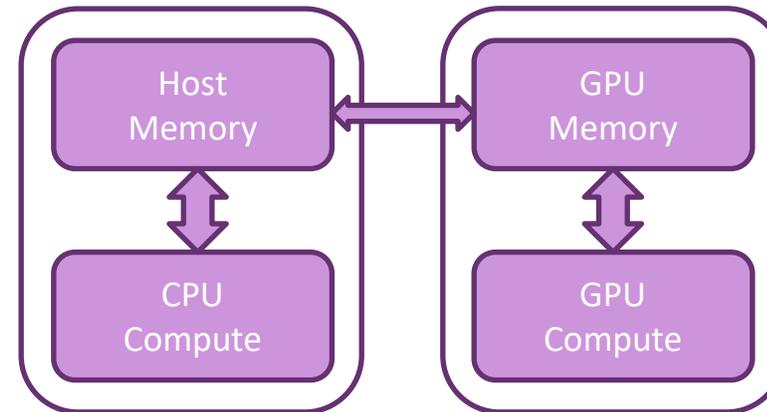
\*N/A is applied to non-summable metrics.

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

# Host and GPU Data Transferring

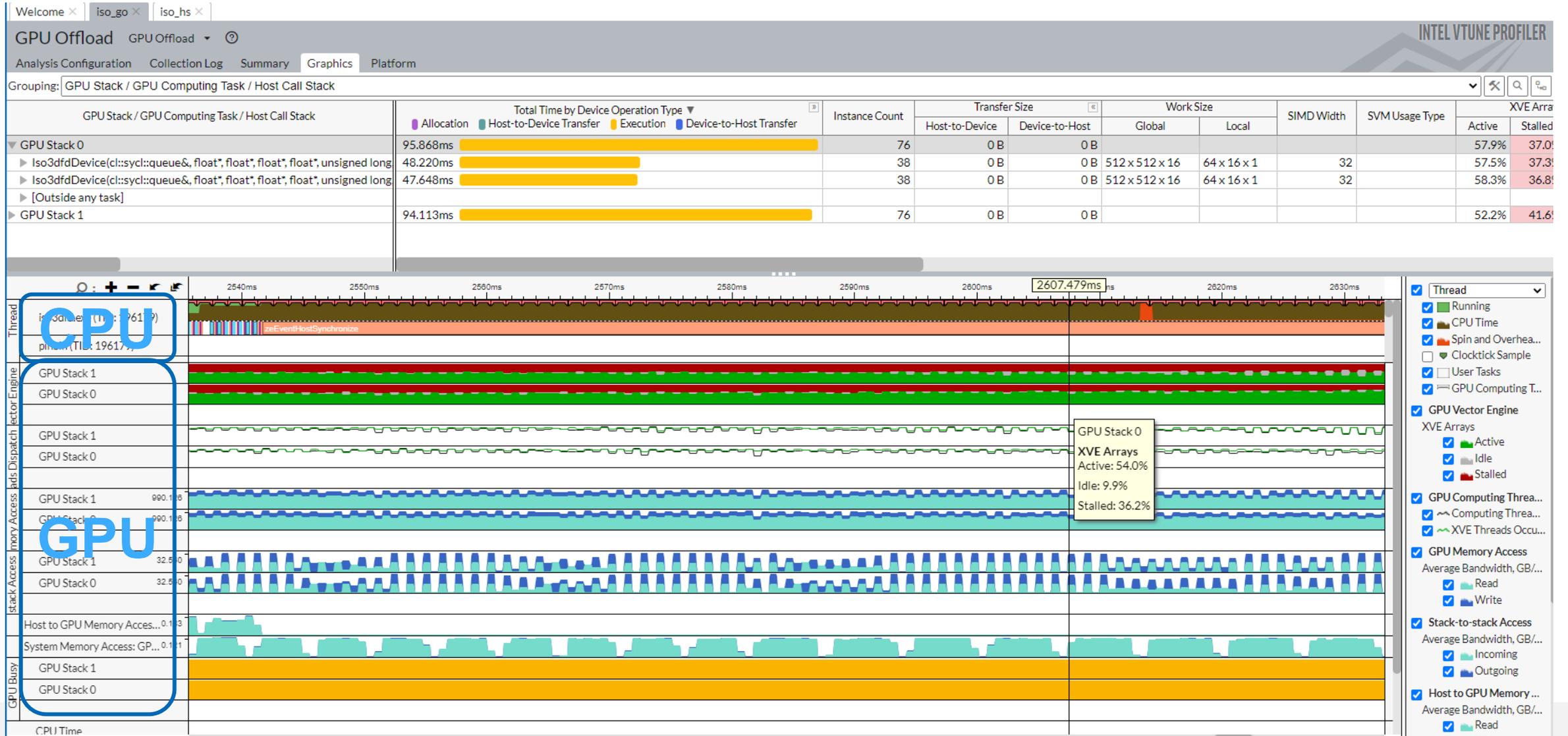
A commonly known problem of host-to-device transfer performance

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

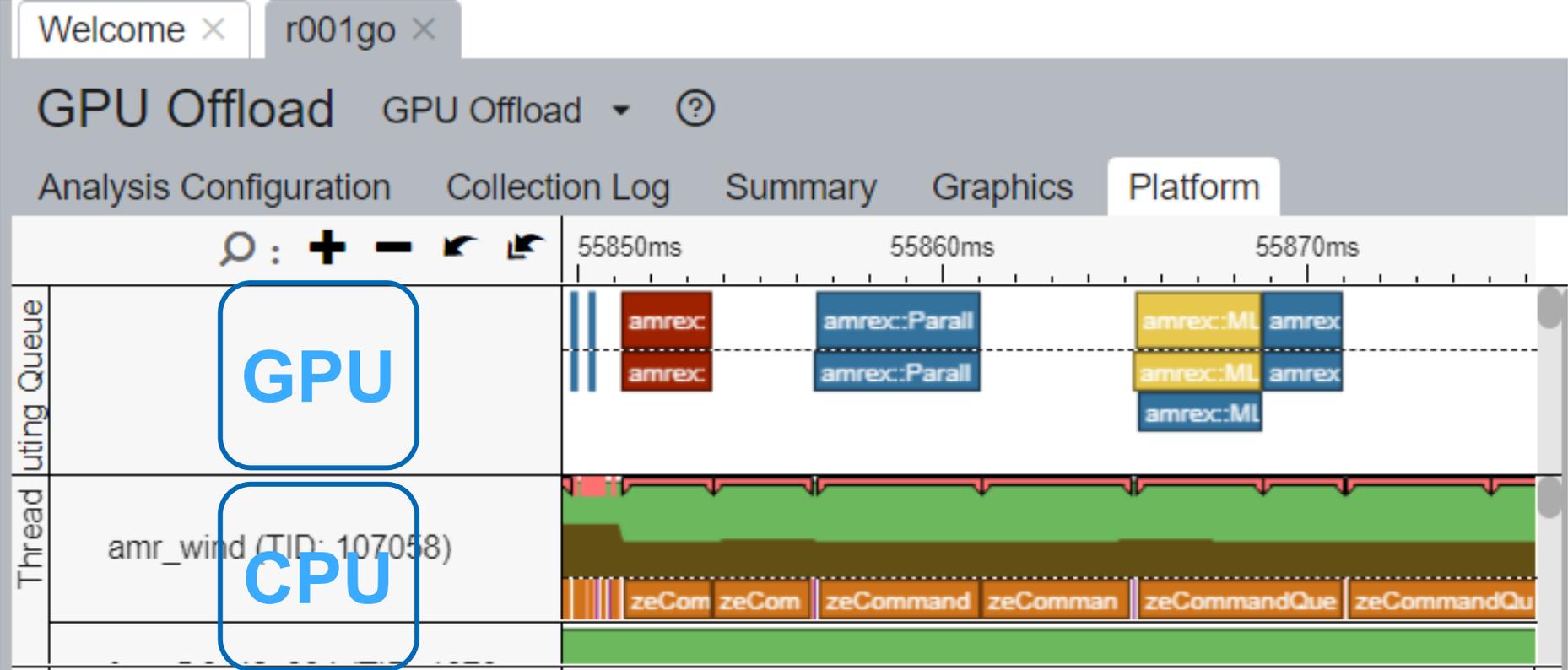


```
vtune -collect gpu-offload [-knob <knobName=knobValue>] [--] <app>
```

# Graphics View of GPU Offload



# Graphics View of GPU Offload

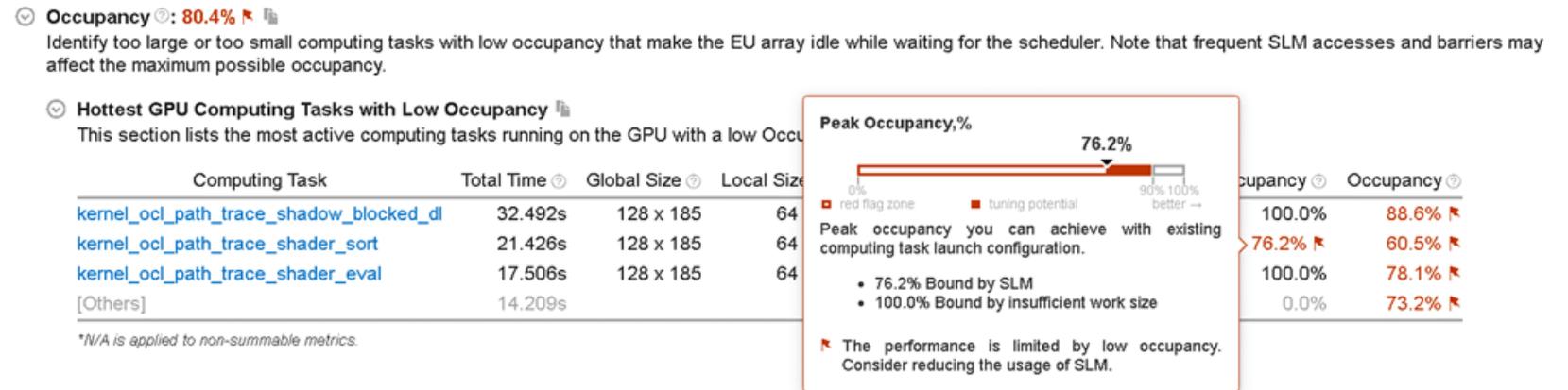


# Achieving High XVE Threads Occupancy

Occupancy analysis helps identifying problems with work mapping

- Detecting workgroups by global and local sizes
- Defining sub-groups by vector sizes or SIMD Width
- Specifying SLM size
- Barriers usage
- Tiny/huge kernels

scheduling issues



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

# Kernel code optimizations

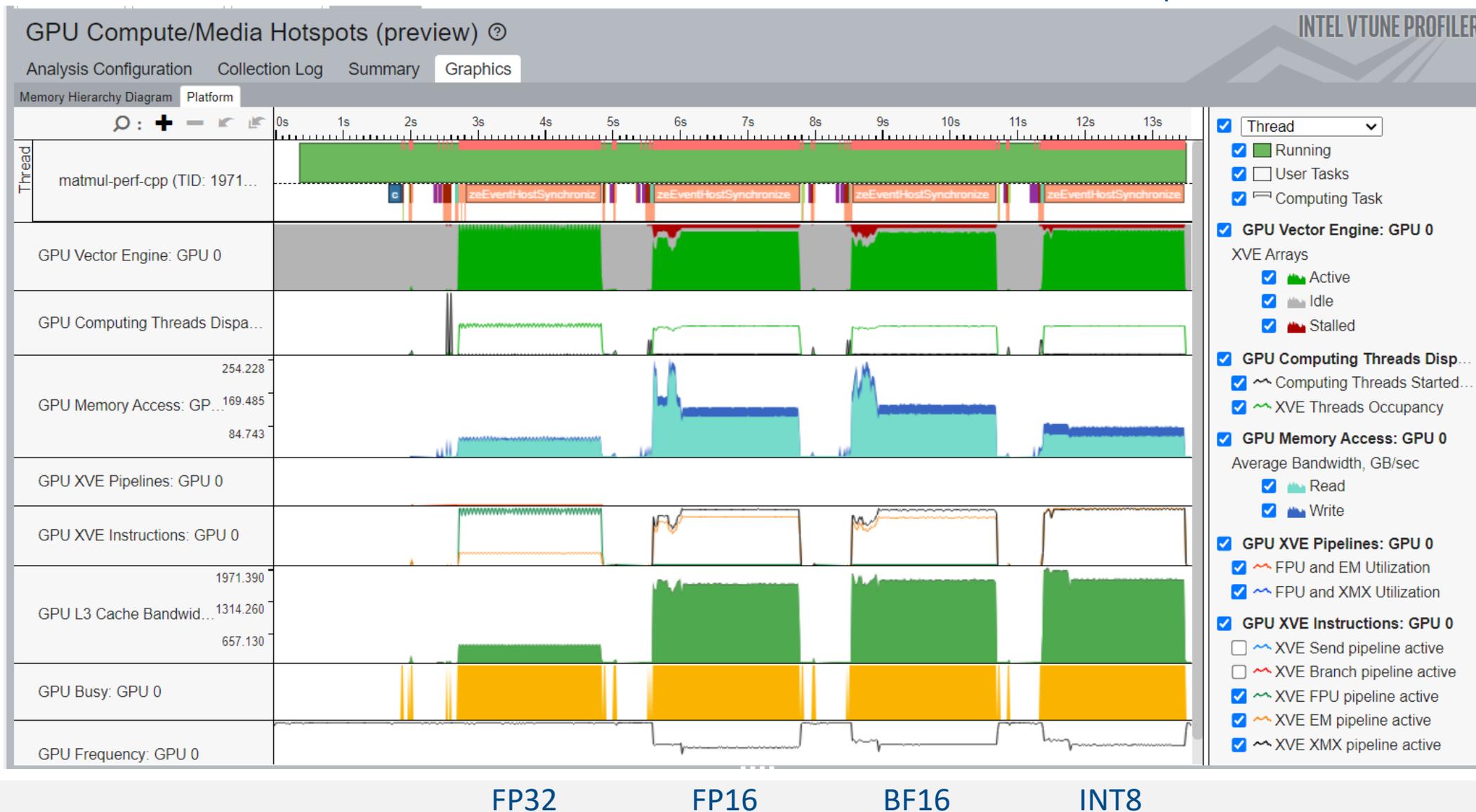
## Advanced code optimizations on kernel level

- Are FPUs and EM pipelines fully utilized?
- How are the systolic instructions used in AI application?
- Instructions counting profiles
- FPU and XMV pipeline Utilization



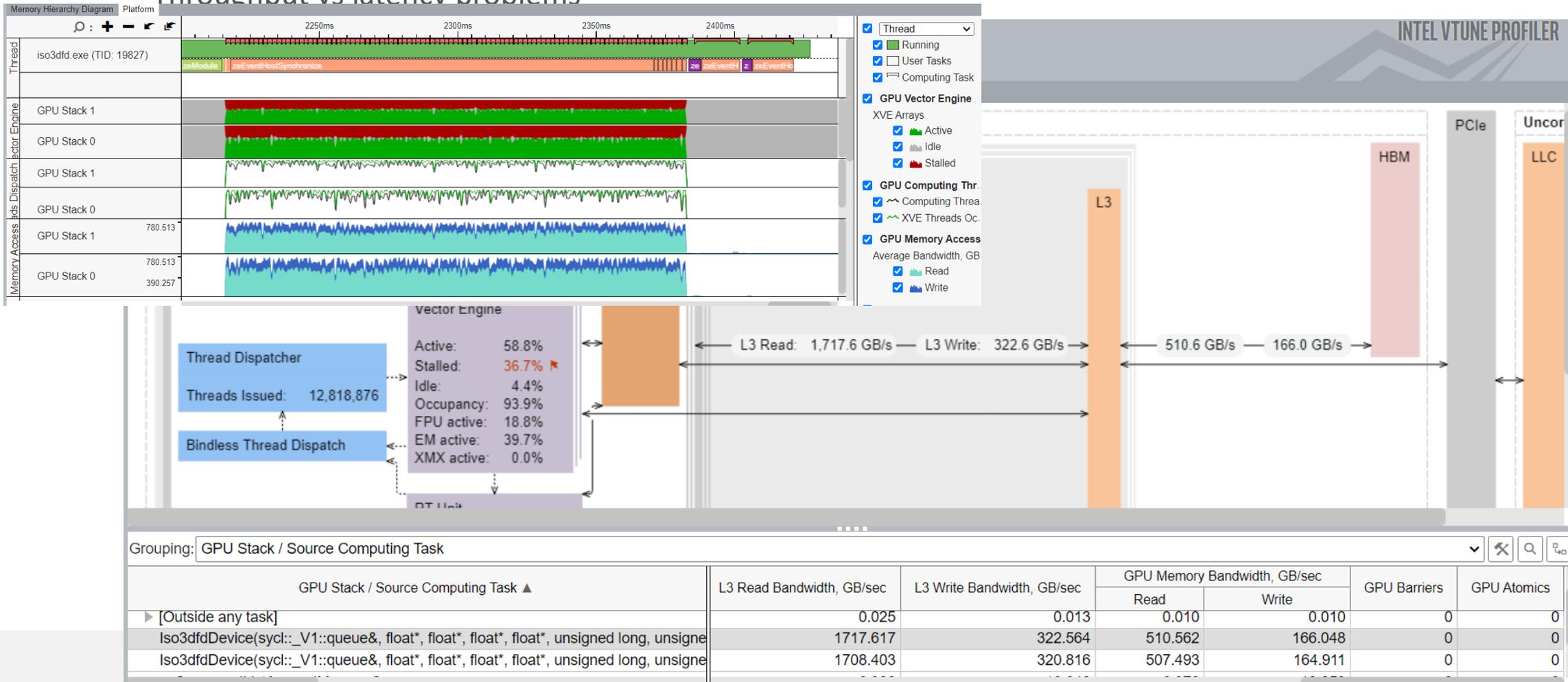
# oneDNN with Intel® XMV

## GEMM with different levels of precisions



# Memory Access problems

- Global memory access penalty
- Cache memory resource limit
- Throughput vs latency problems
- Which code is responsible for latency?
- Per Basic Block and latencies per individual instructions



# Source level in-kernel profiling

vtune -collect gpu-hotspots --knob profiling-mode=source-analysis -knob source-analysis=bb-latency/mem-latency -knob computing-task-of-interest=*\*pattern\*#start#step#end* [--] <app>

GPU Compute/Media Hotspots (preview) ©

Analysis Configuration Collection Log Summary Graphics iso3dfd\_kernels.cpp ×

Source Assembly

Source	Estimated GPU Cycles
428 for (auto iter = 0; iter < kHalfLength; iter++) {	
429 front[iter] = front[iter + 1];	
430 }	
431	
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
435	
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*

GPU Compute/Media Hotspots (preview) ©

Analysis Configuration Collection Log Summary Graphics iso3dfd\_kernels.cpp ×

Source Assembly

Source	Average Latency, Cycles	Estimated GPU Cycles
431		
432 // Only one new data-point read from global memory		
433 // in z-dimension (depth)		
434 front[kHalfLength] = prev[gid + kHalfLength * nxy];	856	1.061e+10
435		
436 // Stencil code to update grid point at position given by global id (gid)		
437 float value = c[0] * front[0];		
438 #pragma unroll(kHalfLength)		
439 for (auto iter = 1; iter <= kHalfLength; iter++) {		
440 value += c[iter] * (front[iter] + back[iter - 1] + prev[gid + iter] +	214	7.963e+9
441 prev[gid - iter] + prev[gid + iter * nx] +	186	2.302e+10
442 prev[gid - iter * nx]);	196	1.943e+10
443 }		
444		
445 next[gid] = 2.0f * front[0] - next[gid] + value * vel[gid];	875	2.169e+10
446		
447 gid += nxy;		
448 begin_z++;		
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		
458 *		
459 * This function uses SYCL buffers to facilitate host to device		
460 * buffer copies		

*Memory Latency*

# HW-Assisted Stall Sampling

- Provides detailed breakdown of stalls and reasons
- HW-assisted Stall Sampling technology designed for Intel® Data Center GPU Max Series (code-named Ponte Vecchio or PVC)
- Capabilities similar to instruction execution efficiency characterization of NVIDIA® Nsight™ Compute

```
vtune -collect gpu-hotspots --knob profiling-mode=source-analysis -knob source-analysis=stall-sampling [--] <app>
```

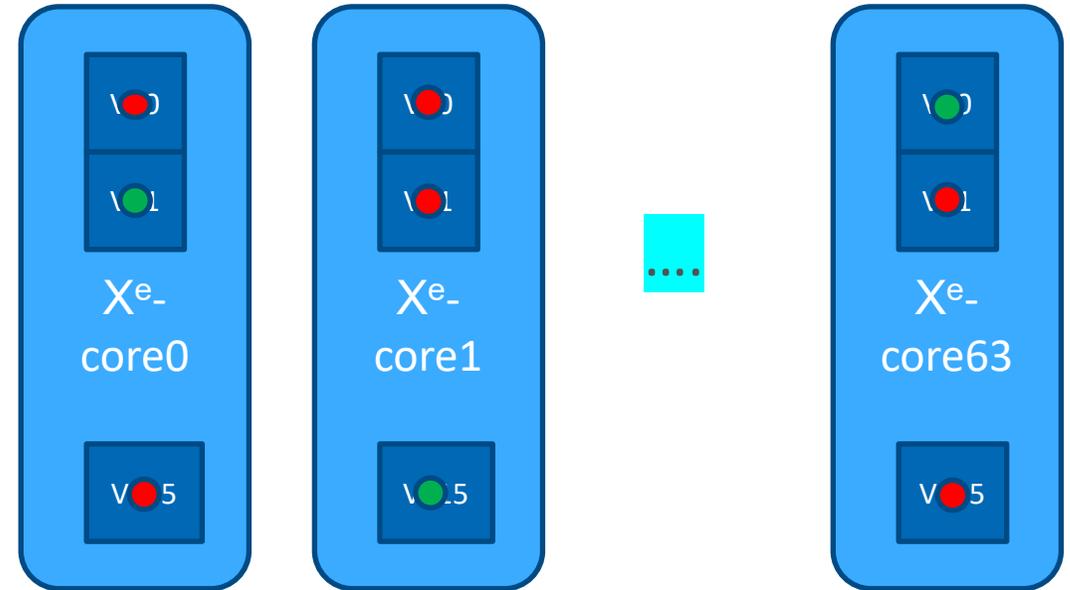
9	__kernel void spmv_jds_naive(__global float *ds	0.1%
10	__global int *d_index,	
11	__global float *x_vec, cor	
12	__constant int *jds_ptr_in	
13	__constant int *sh_zcnt_in	
14	{	
15	int ix = get_global_id(0);	
16		
17	if (ix < dim) {	0.0%
18	float sum = 0.0f;	
19	// 32 is warp size	
20	int bound=sh_zcnt_int[ix/32];	0.1%
21		
22	for(int k=0;k<bound;k++)	1.5%
23	{	
24	int j = jds_ptr_int[k] + ix;	4.5%
25	int in = d_index[j];	14.8% █
26		
27	float d = d_data[j];	0.6%
28	float t = x_vec[in];	42.7% █
29		
30	sum += d*t;	33.1% █
31	}	
32		
33	dst_vector[d_perm[ix]] = sum;	1.8%
34		
35	}	0.0%



Most stalling line

# High-Level Stall Sampling (Xe-core)

- Sample all Xe Vector Engines(XVE) statistically, one by one
  - Uses sampling interval large enough to make data manageable
  - Small enough to be representative
  - No assumption that all Xe Vector Engine are doing the same work
- Record both
  - Stall
  - Active
  - Idle (Ignore)



State	Meaning
Idle	No threads are loaded on the XVE. Do nothing.
Active	At least one pipeline is dispatching an instruction on the sampled cycle
Stall	One or more threads are loaded on the EU, but no instruction is being dispatched to any pipeline

# GPU Compute/Media Hotspots (preview) ?

Analysis Configuration Collection Log Summary **Graphics**

Memory Hierarchy Diagram Platform

Grouping: Computing Task / Function / Call Stack

Stall Count by Stall Type ▼

GPUActive	Other	Control	Pipe	Send	Dist or Acc	SBID	Synchronization	Instruction Fetch
2.3%	0.1%	1.1%	0.0%	5.7%	46.1%	10.7%	1.7%	
2.3%	0.1%	1.1%	0.0%	5.7%	46.1%	10.7%	1.7%	
2.3%	0.1%	1.1%	0.0%	5.7%	46.1%	10.7%	1.7%	
0.0%	0.0%	0.2%	0.0%	0.0%	20.8%	0.0%	0.0%	
0.0%	0.0%	0.2%	0.0%	0.0%	20.8%	0.0%	0.0%	
0.0%	0.0%	0.2%	0.0%	0.0%	20.8%	0.0%	0.0%	

Actively executing in at least one pipeline

Stalled Due to other reasons

Stalled due to branch

Stalled due to XVE pipeline

Stalled due to memory dependency or internal pipeline dependency for send

Stalled due to internal pipeline dependency

Stalled due to memory dependency or internal EU pipeline dependency

Stalled due to sync operation

Stalled due to an instruction fetch operation

# Source Analysis view of Stall Sampling

Intel VTune Profiler - GPU Compute/Media Hotspots (preview)

Analysis Configuration | Collection Log | Summary | Graphics | 3\_GPU\_linear.cpp

Source | Assembly

Source Line	Source	Stall Count by Stall Type								
		Active	Other	Control	Pipe	Send	Dist or Acc	SBID	Synchroniza...	Instruction Fetch
42	sets to indices to exclude HALO									
43	n2 * n3;	2,748	27	0	1,407	0	41	0	0	0
44	dx[0] + kHalfLength;	484	0	0	120	0	5	0	0	0
45	dx[1] + kHalfLength;	524	2	0	125	0	1	0	0	0
46	dx[2] + kHalfLength;									
47										
48	te linear index for each cell									
49	i * n2n3 + j * n3 + k;	5,867	79	0	3,222	0	198	0	0	0
50										
51	te values for each cell									
52	e = prev_acc[idx] * coeff_acc[0];	2,323	30	0	844	0	23	0	0	0
53										
54	= 1; x <= kHalfLength; x++) {									
55		2,180	97	0	113	0	23	0	0	0
56	f_acc[x] * (prev_acc[idx + x] + prev_acc[idx - x]);	10,713	2,421	0	3,048	0	943	6,757	0	0
57	prev_acc[idx + x * n3] + prev_acc[idx - x * n3];	38,436	2,059	0	20,038	0	1,457	23,052	0	0
58	prev_acc[idx + x * n2n3] + prev_acc[idx - x * n2n3];	24,060	6,599	0	11,350	0	824	13,521	0	0
59										
60	dx] = 2.0f * prev_acc[idx] - next_acc[idx] +	1,498	260	0	174	0	289	9,676	0	0
61	value * vel_acc[idx];	373	759	0	7	0	58	0	0	0
62	device code									
63										
64										
65										

Intel VTune Profiler - GPU Compute/Media Hotspots (preview)

Analysis Configuration | Collection Log | Summary | Graphics | 3\_GPU\_linear.cpp

Source | Assembly

Assembly grouping: Address

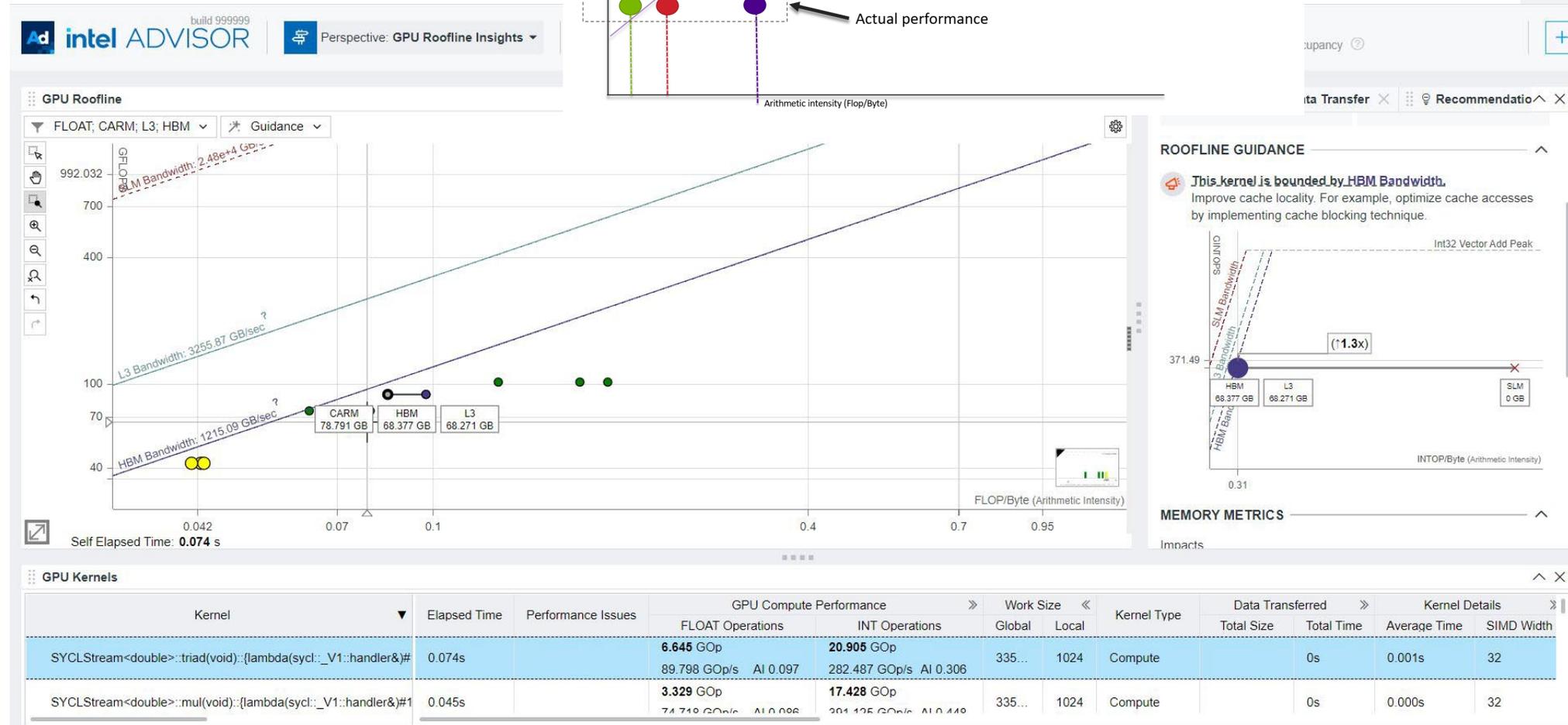
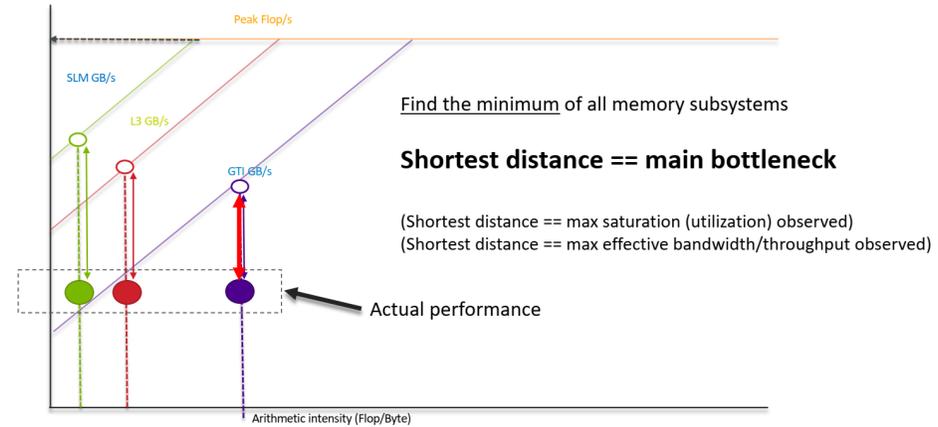
Source Line	Source	Stall Count by Stall Type				
		Pipe	Send	Dist or Acc	SBID	Syn
42	sets to indices to exclude HALO					
43	n2 * n3;	1,407	0	41	0	0
44	dx[0] + kHalfLength;	120	0	5	0	0
45	dx[1] + kHalfLength;	125	0	1	0	0
46	dx[2] + kHalfLength;					
47						
48	te linear index for each cell					
49	i * n2n3 + j * n3 + k;	3,222	0	198	0	0
50						
51	te values for each cell					
52	e = prev_acc[idx] * coeff_acc[0];	844	0	23	0	0
53						
54	= 1; x <= kHalfLength; x++) {					
55		113	0	23	0	0
56	f_acc[x] * (prev_acc[idx + x] + prev_acc[idx - x]);	3,048	0	943	6,757	0
57	prev_acc[idx + x * n3] + prev_acc[idx - x * n3];	20,038	0	1,457	23,052	0
58	prev_acc[idx + x * n2n3] + prev_acc[idx - x * n2n3];	11,350	0	824	13,521	0
59						
60	dx] = 2.0f * prev_acc[idx] - next_acc[idx] +	174	0	289	9,676	0
61	value * vel_acc[idx];	7	0	58	0	0
62	device code					
63						
64						
65						

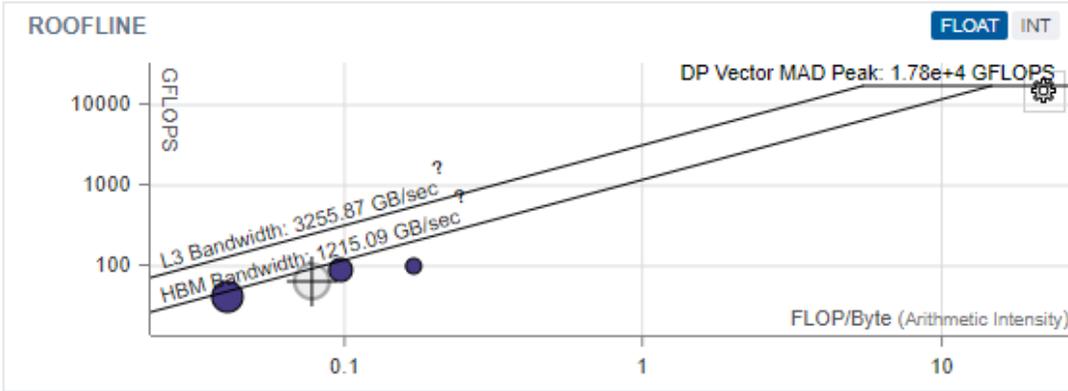
Address	Source Line	Assembly	Stall Count by Stall Type			
			Pipe	Send	Dist or Acc	SBID
0x960	58	shl (16 M16) r114,0<1>				
0x968	58	send.ugm (32 M0) r11	0	0	37	0
0x978	58	add (16 M0) r118,0<1>				
0x980	58	add (16 M16) r120,0<1>				
0x988	58	send.ugm (32 M0) r1	0	0	154	0
0x998	56	add (32 M0) acc0,0<1>	3	0	0	57
0x9a0	56	add (32 M0) acc0,0<1>	3	0	814	829
0x9b0	57	add (32 M0) r40,0<1>	1	0	66	1,504
0x9c0	57	(W) mul (1 M0) acc0,0<1>	40	0	201	0
0x9d0	57	add (32 M0) acc2,0<1>	2	0	0	4,250
0x9d8	56	sync.nop null {Compa	0	0	0	0
0x9e0	56	add (32 M0) r35,0<1>	9	0	0	0
0x9e8	57	(W) mul (1 M0) r61,0<1>	83	0	0	51
0x9f0	57	(W) mul (1 M0) r62,0<1>	106	0	0	0
0xa00	57	(W) mach (1 M0) r66,0<1>	45	0	60	0
0xa10	57	(W) shl (1 M0) r63,0<1>	41	0	0	0
0xa18	57	subb (16 M0) r84,0<1>	53	0	47	0
0xa28	57	sync.nop null {Compa	0	0	44	0
0xa30	57	(W) add (1 M0) r90,0<1>	60	0	0	0
0xa38	58	add (32 M0) r49,0<1>	5	0	0	5,428
0xa48	56	sync.nop null {Compa	0	0	0	0
0xa50	56	add (32 M0) acc2,0<1>	7	0	0	376
0xa58	57	(W) mov (2 M0) r71,0<1>	108	0	0	0
0xa60	57	add3 (16 M0) r110,0<1>	109	0	65	0
0xa70	57	subb (16 M16) r85,0<1>	0	0	0	0

# GPU Roofline Analysis

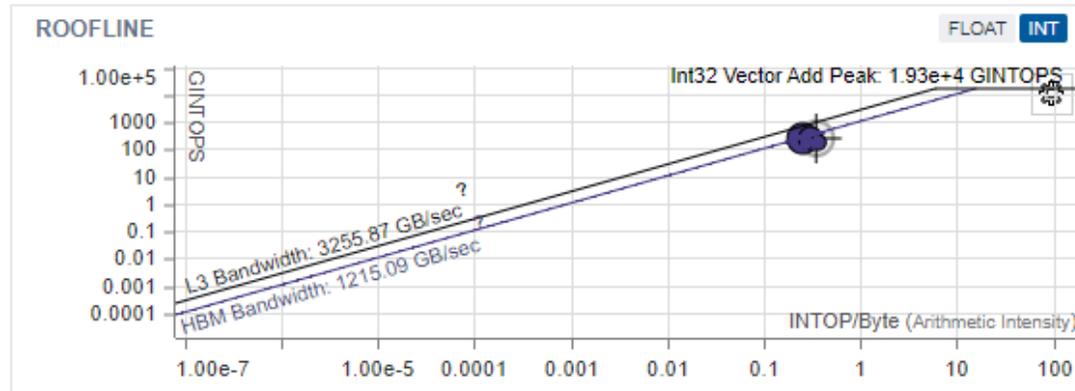
- kernels memory bound or compute bound?
- Compare the kernels performance to the capabilities of the GPU
- Memory Level Roofline - unambiguous bottleneck detection



# 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
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**

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	<a href="#">loop in intel...</a>	0.08s		0.15729049530...
SYCLStrea...	0.07s	89.798	282.487	33554432/...	7.8/77.1/15.1	<a href="#">loop in piEngue...</a>	0.02s		0.00056129438...
sycl:: V1:...	0.07s	102.107	211.759	524288/1024	5.1/56.8/38.1	<a href="#">loop in func@0...</a>	0.01s		
SYCLStrea...	0.04s	74.718	391.125	33554432/...	10.8/69.3/1...	<a href="#">loop in func@0...</a>	0.01s		
SYCLStrea...	0.04s	0	437.821	33554432/...	8.3/59.3/32.4	<a href="#">loop in livm:Ba...</a>	0.01s		

---

**Platform Information** | **Collection 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%

# GPU Roofline in Intel® Advisor

## 1<sup>st</sup> method: Run the shortcut command, simple

```
$ advisor --collect=roofline --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>
```

- 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
```

## 2<sup>nd</sup> 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>
```

intel®