# 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

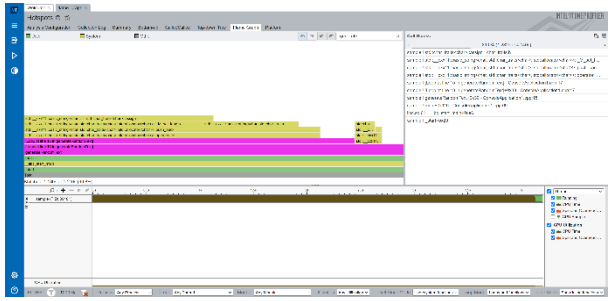
- Collect data HW/SW sampling and tracing w/o re-compilation
- See results on your source, in architecture diagrams, as a histogram, on a timeline…
- Filter and organize data to find answers

### Work Your Way

- User interface or command line
- Profile locally and remotely

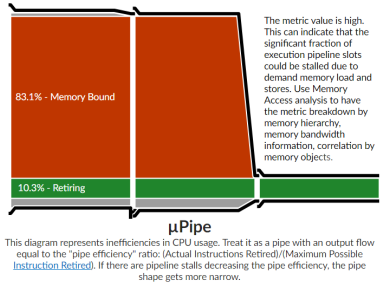
# Rich Set of Profiling Capabilities

## Intel® VTune™ Profiler



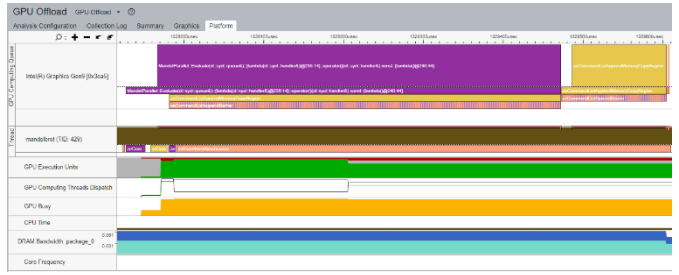
### Algorithm Optimization

- ✓ Hotspots
- ✓ Anomaly Detection
- ✓ Memory Consumption



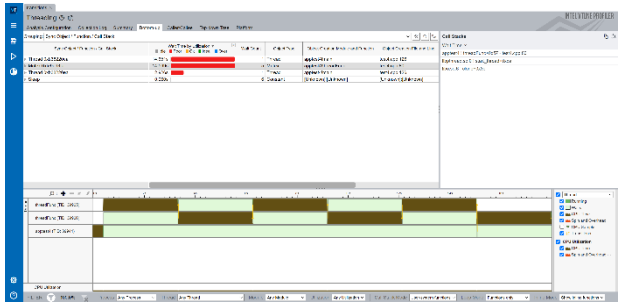

### Microarch.&Memory Bottlenecks

- ✓ Microarchitecture Exploration
- ✓ Memory Access



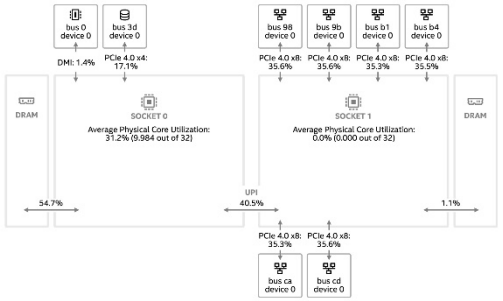

### Accelerators / xPU

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



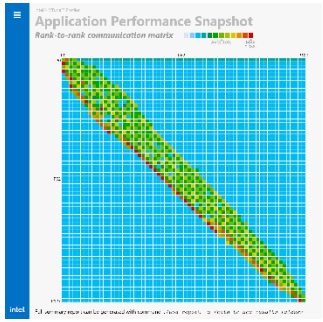
### Parallelism

- ✓ Threading
- ✓ HPC Performance Characterization




### Platform

- ✓ 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

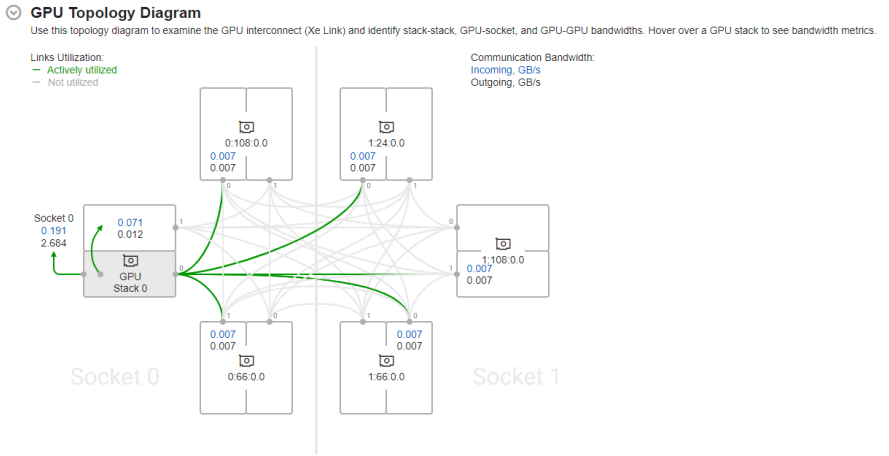
- Get visibility into XeLink cross-card traffic for issues such as stack-to-stack traffic, throughput and bandwidth bottlenecks. Identify imbalances of traffic between CPU and GPU through a GPU topology diagram.
- Identify the reasons of the stalls in Xe Vector Engines (XVEs), formerly known as Execution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

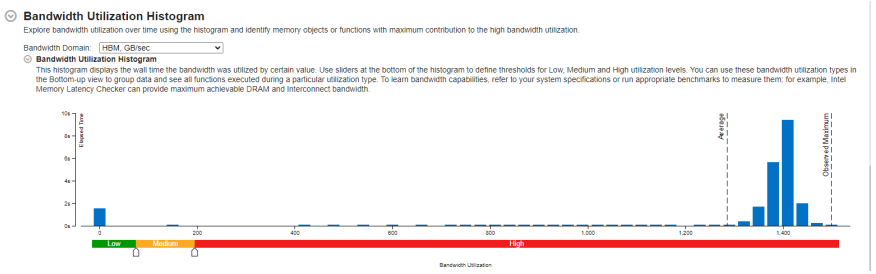
- Identify and optimize performance hotspots of Python code, now supporting Python 3.9.*.

## 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.

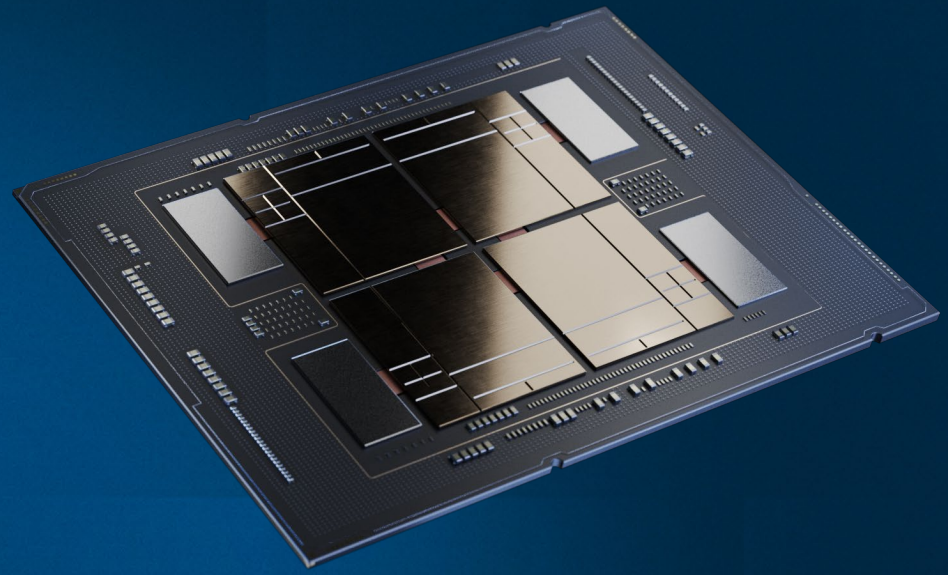



*Cross-card, stack-to-stack, and card-to-socket bandwidth are presented on GPU Topology Diagram.*




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

# Only x86 CPU with High Bandwidth Memory



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

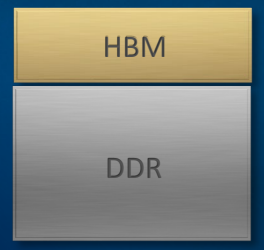
No code change
No DDR

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

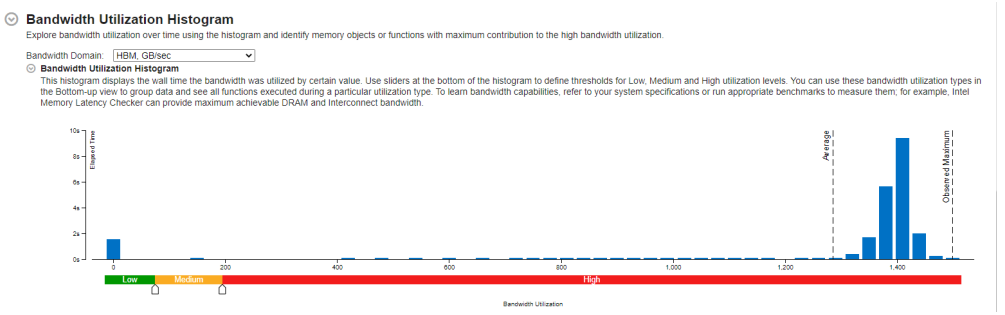
Intel® VTune™ Profiler

## Understand HBM memory usage

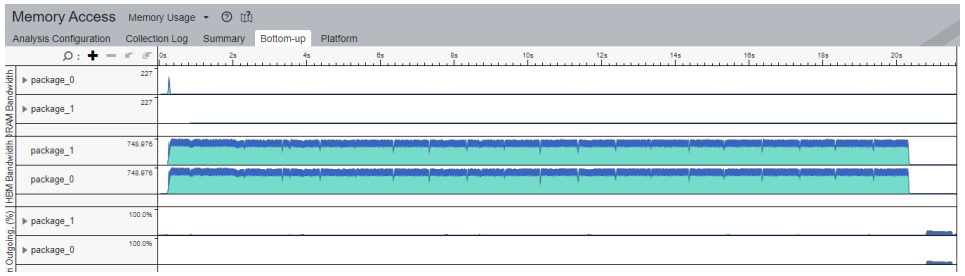
- Is the application performance affected by HBM utilization?
- How is the bandwidth distributed between DRAM vs. HBM?

## 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.*



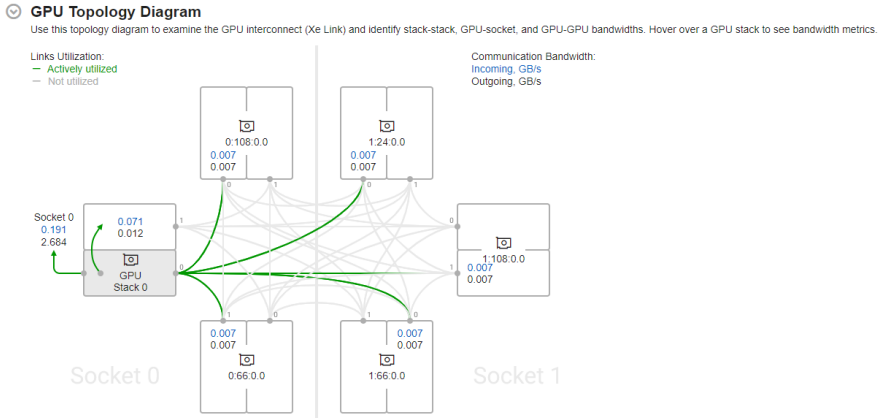

*The workload performance in various HBM modes can be evaluated by running the collection in each mode and analyzing the bandwidth as described above.*

# Get Visibility into Xe Link Cross-card Traffic

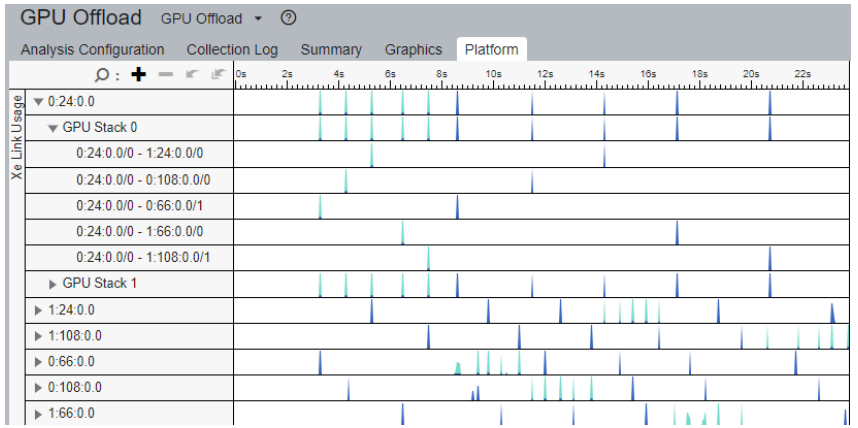
## Intel® VTune™ Profiler

### 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.*

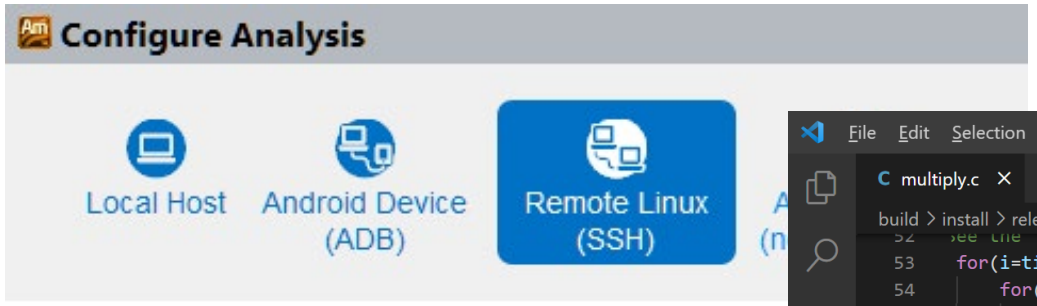
# Access Intel® VTune™ Profiler via web browser

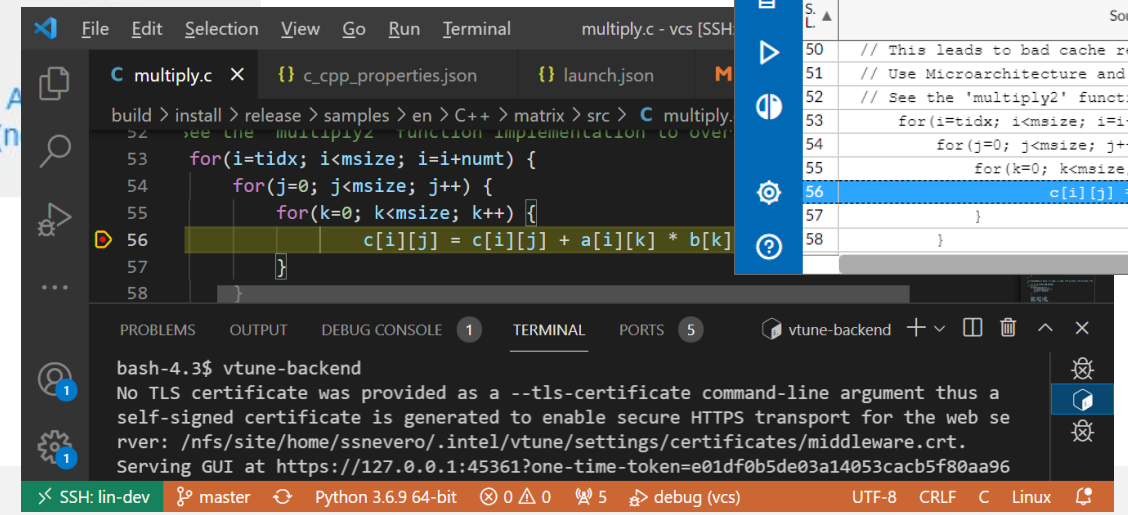
## Interactive analysis

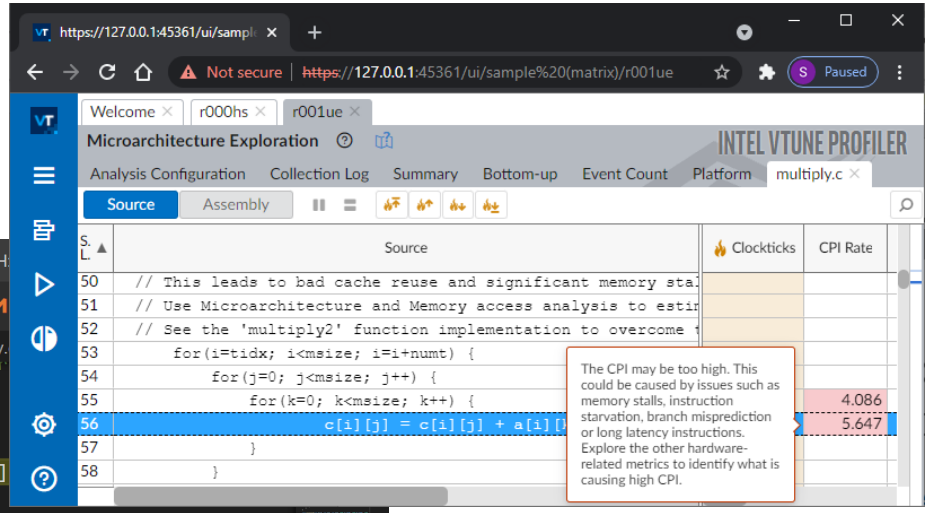
1) Configure SSH to a remote Linux* target
2) Choose and run analysis with the UI

## 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

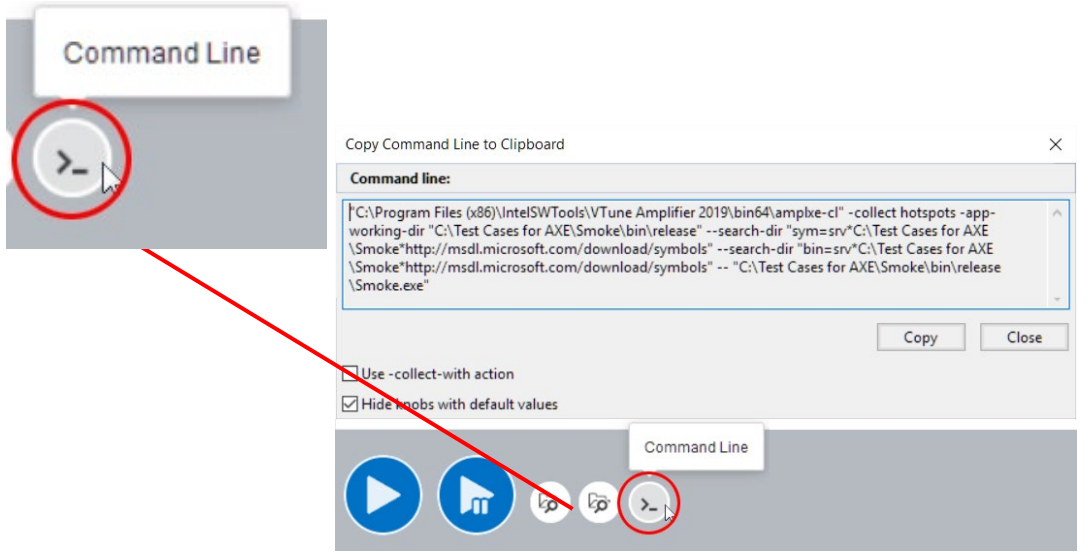
Help:

vtune –help

vtune –help collect hotspots

Use UI to setup

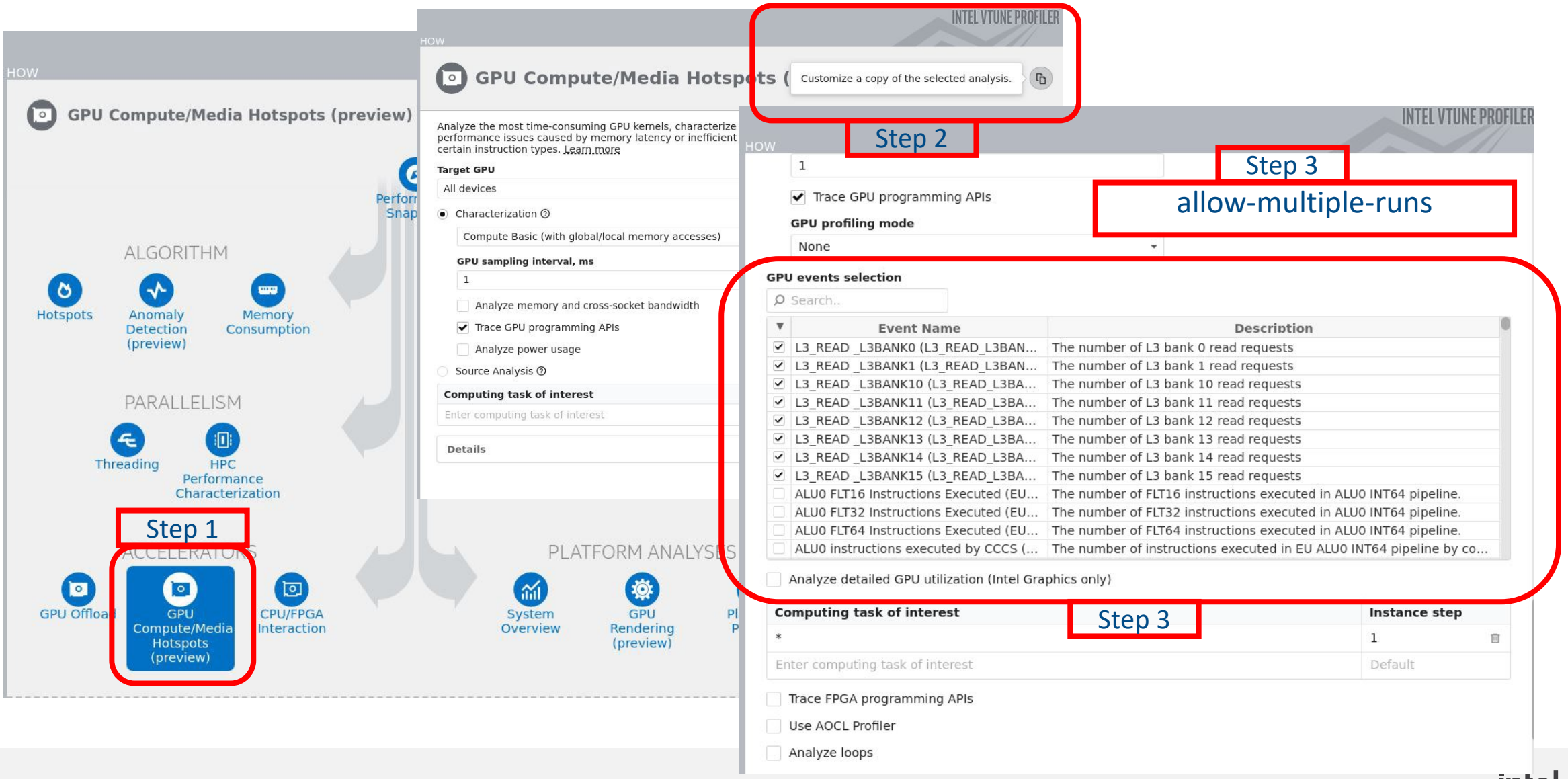
1) Configure analysis in UI
2) Press “Command Line…” button
3) Copy & paste command



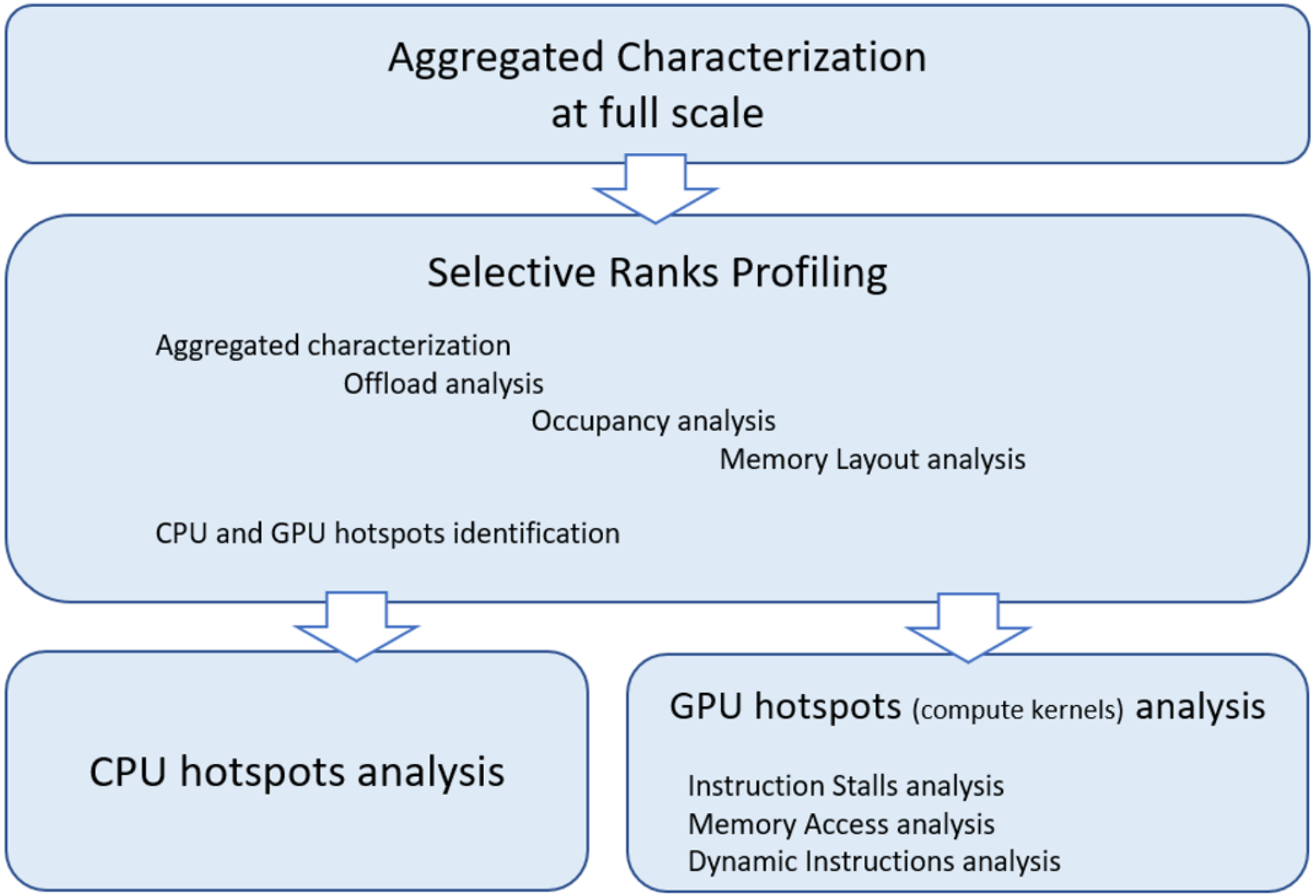

vtune -collect hpc-performance [-knob <knobName=knobValue>] [--] <app>

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

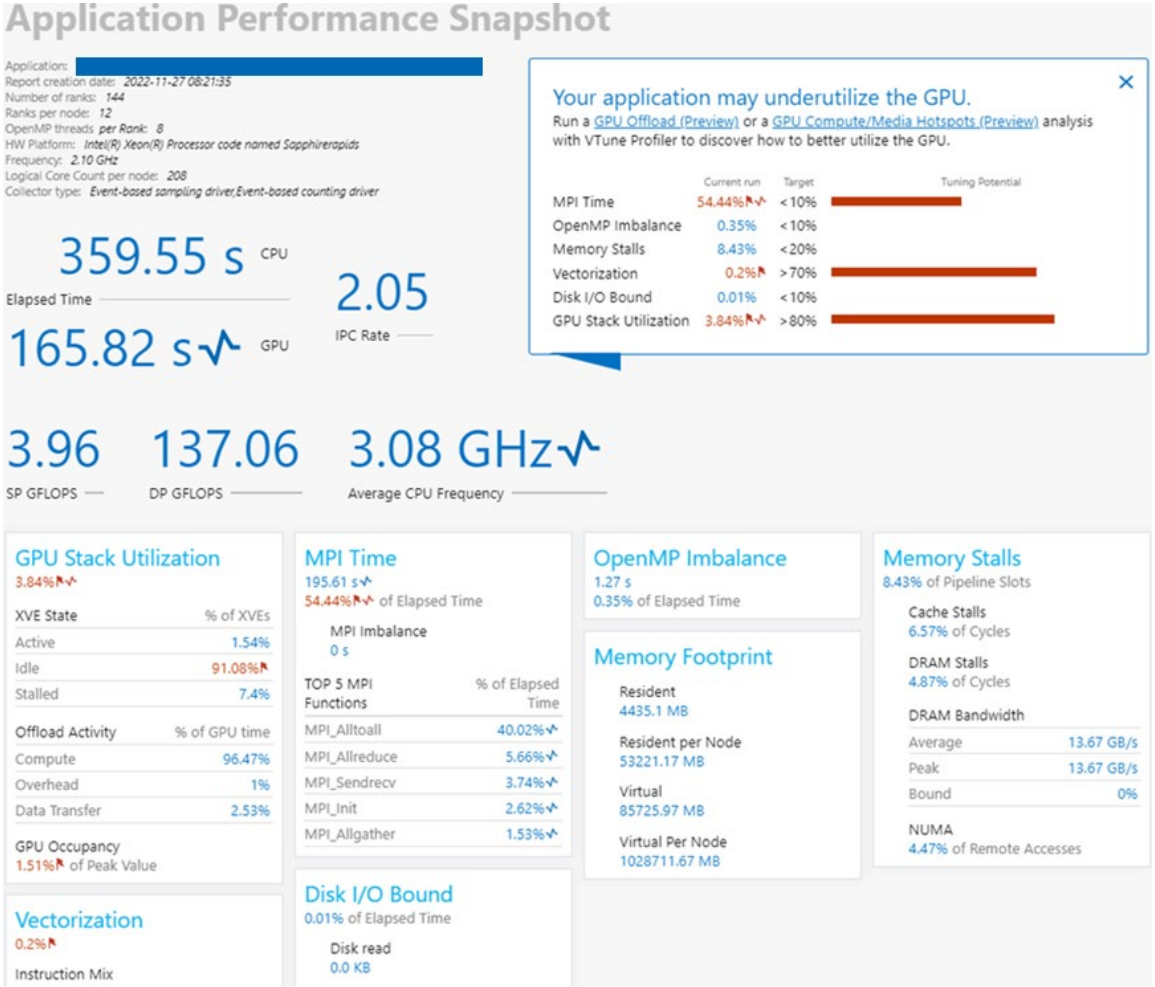
# Custom Analysis with VTune Profiler



# General strategy



# Intel® VTune™ Profiler Application Performance Snapshot (APS)

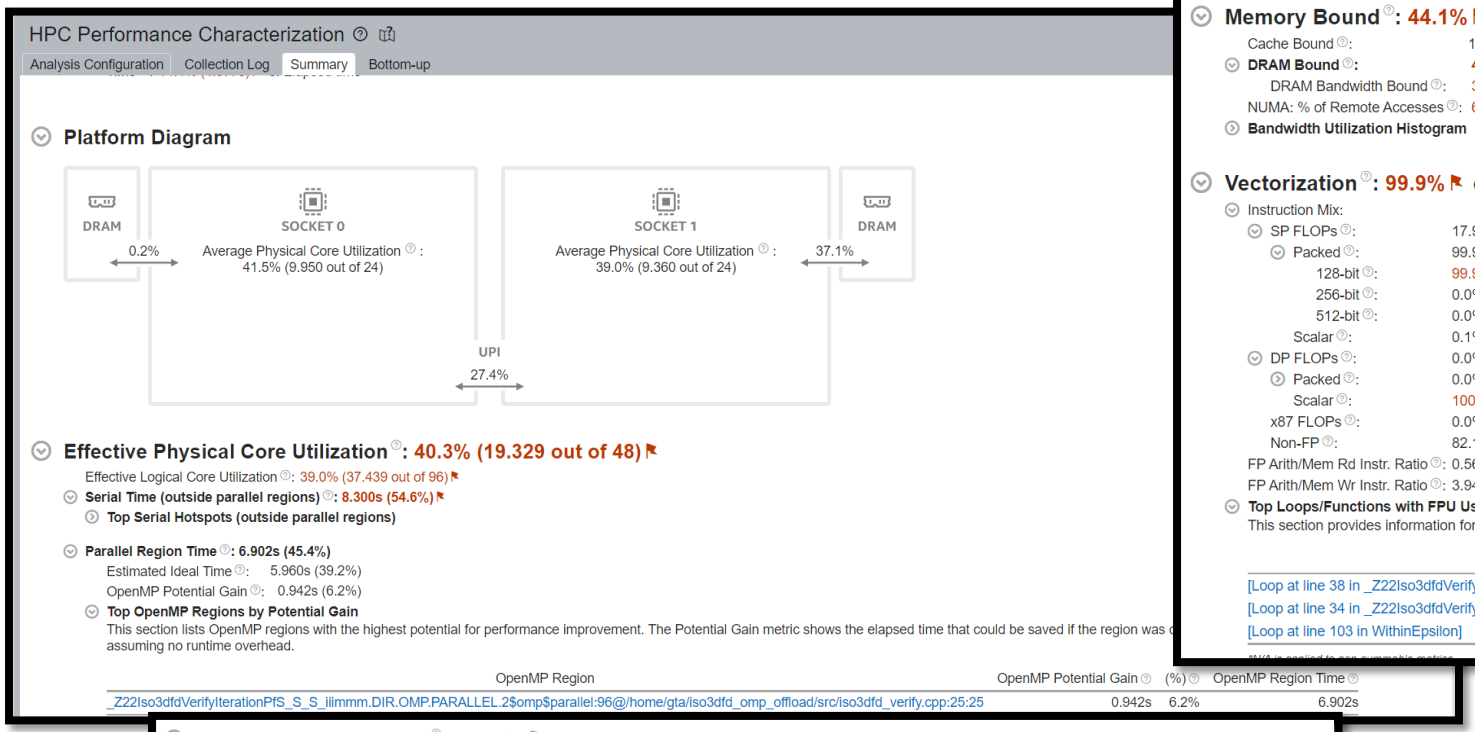


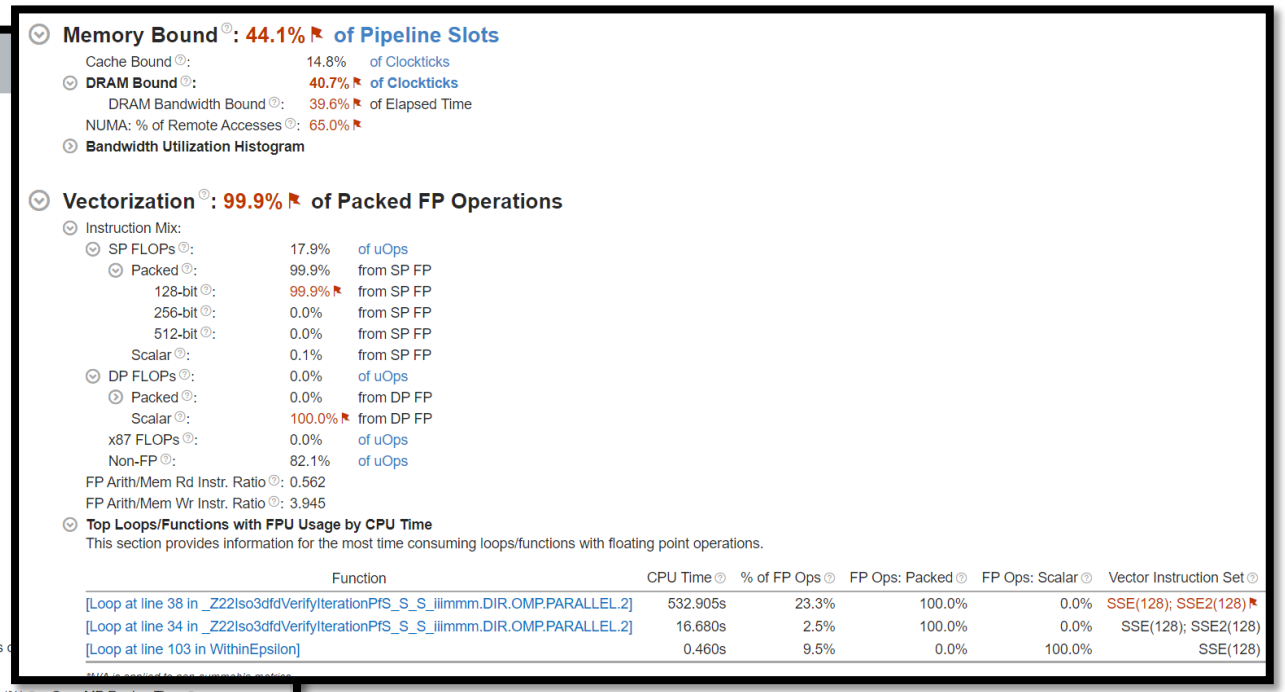

- High-level **overview** of application performance
  - Detailed reports on MPI statistics
- Primary optimization areas and **next steps** in analysis with deep tools – e.g. outlier analysis for MPI applications at scale
  - Explore on source of imbalance
  - Choose nodes/ranks for detailed profiling 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:

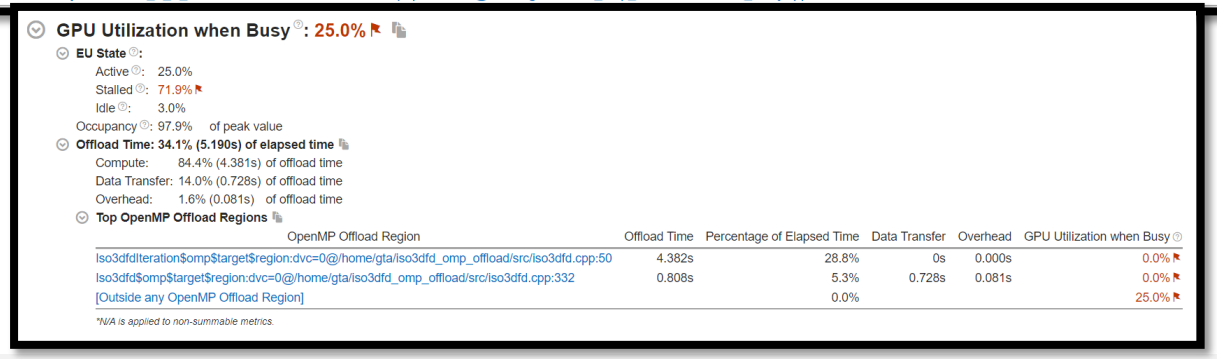
```
<mpi launcher> <mpi parameters> aps  <app>
```

Intel® VTune™ Profiler

# HPC Performance Characterization




A starting point for performance optimization

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

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

# 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
```

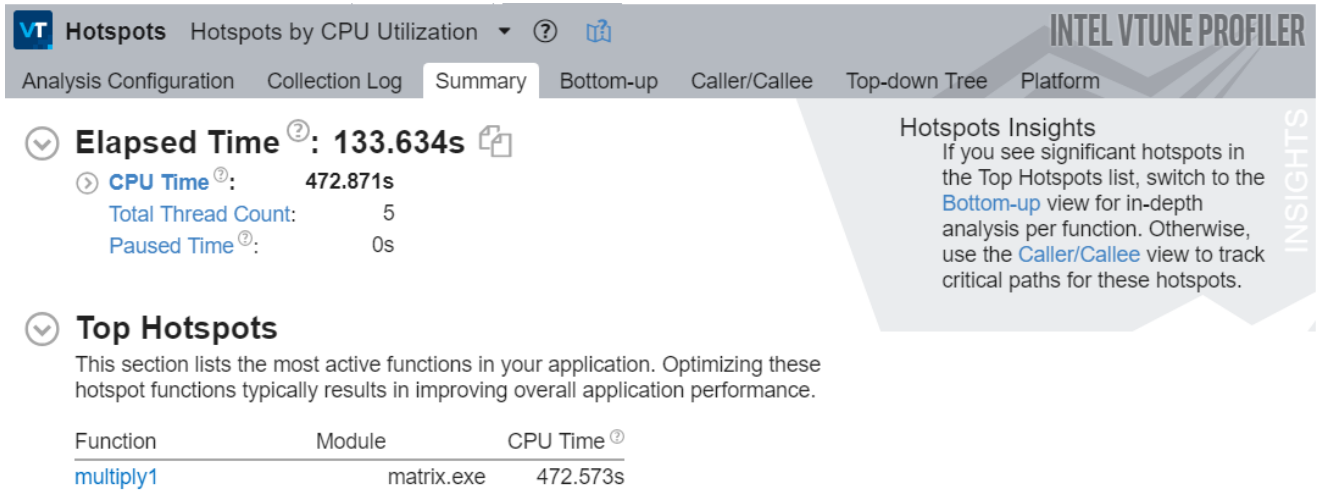
# Hotspots Analysis

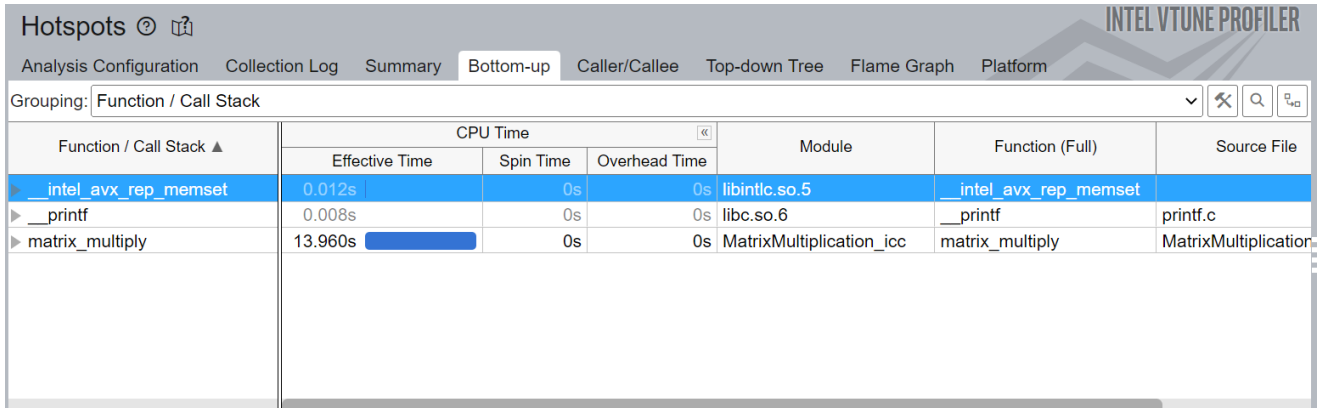
- Understand an application flow
- Identify sections of code that get a lot of execution time
- Sampling-based collection modes
  - User-Mode Sampling
  - Hardware Event Based Sampling
- Define a performance baseline.

- Identify the hottest function.

- Identify algorithm issues.

- Analyze source.



# What's Using All The Memory?

## Memory Consumption Analysis

### See What Is Allocating Memory

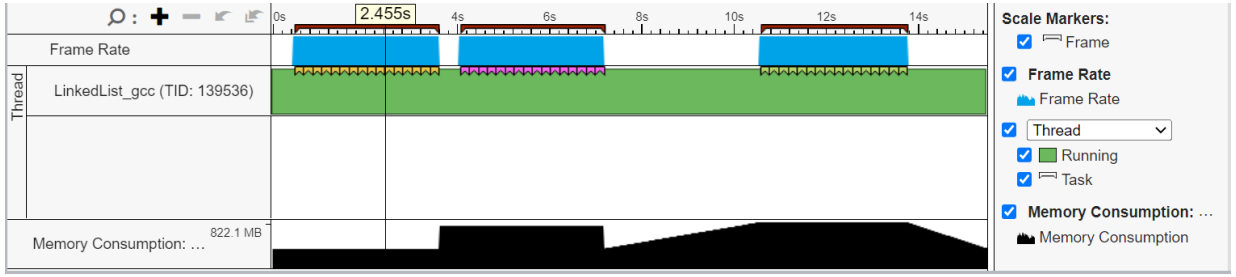
- Lists top memory consuming functions
- memory consumption distribution over time.
- View source to understand cause
- Filter by time using the memory consumption timeline
  - Focus on the peak values on the Timeline pane
- Introduce additional overhead due to instrumentation .

**Top Memory-Consuming Functions**

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

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

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

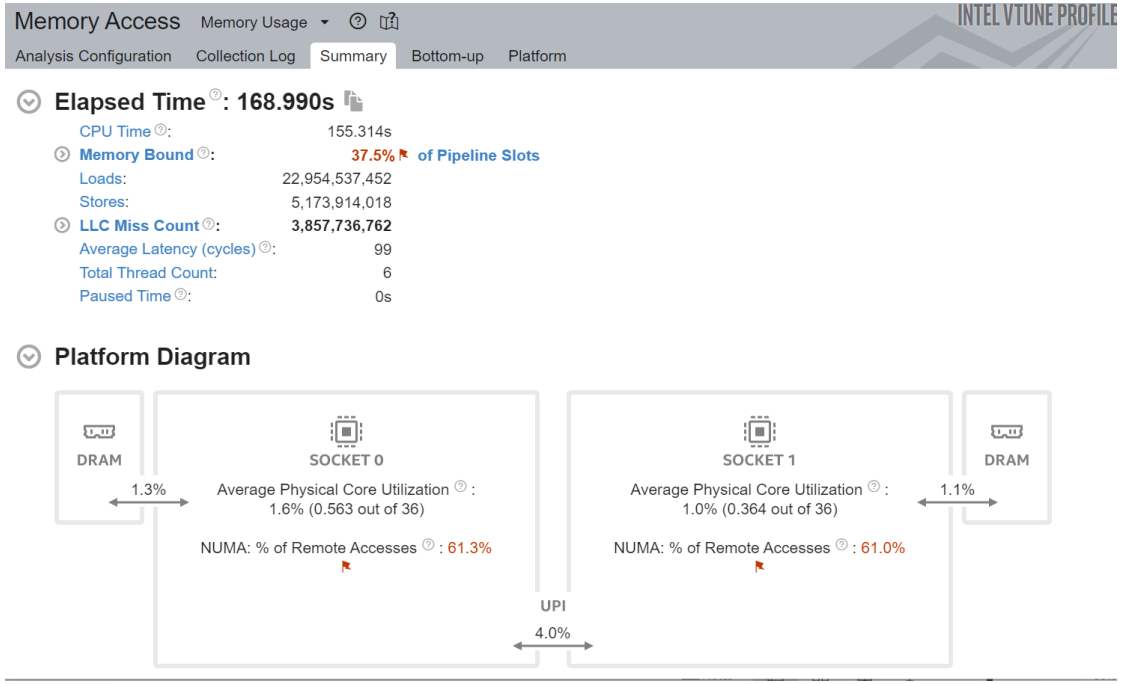



Native language support is not currently available for Windows*

# Optimize Memory Access

## Memory Access Analysis

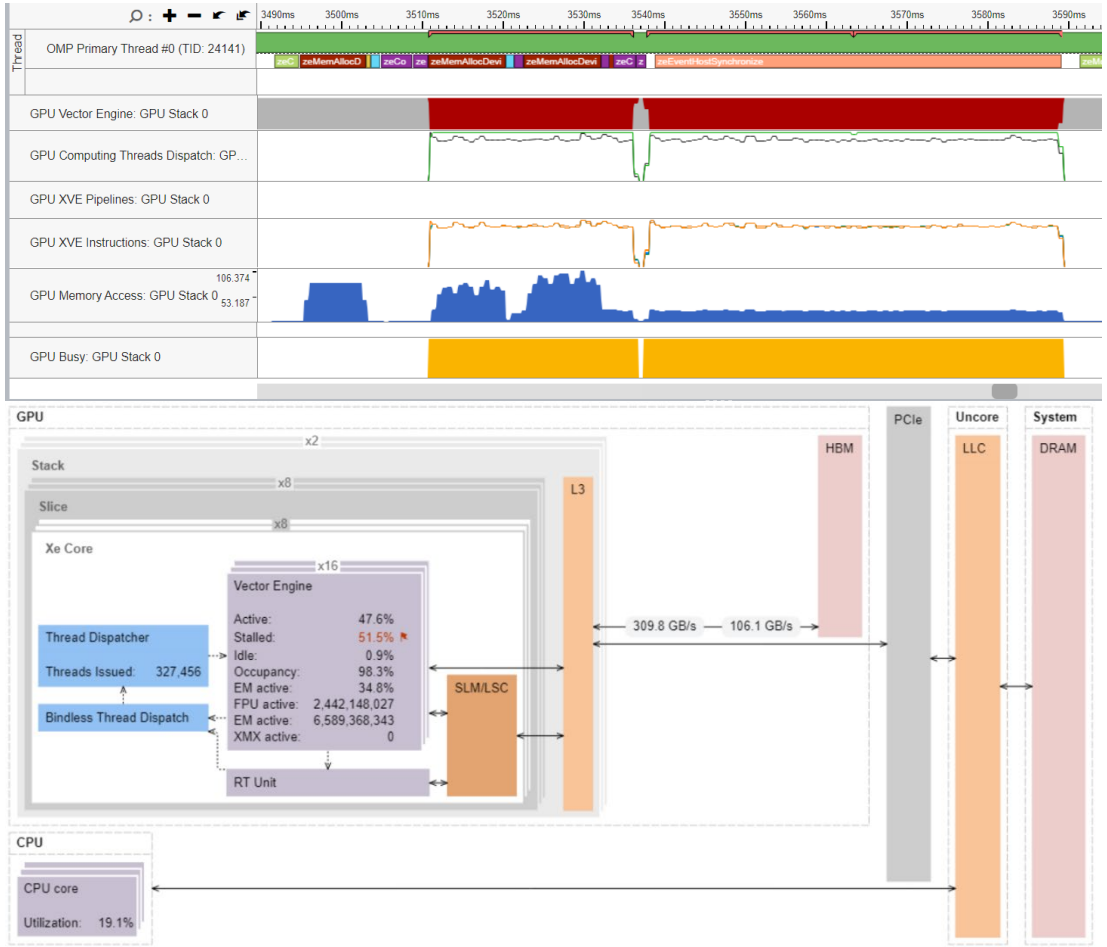
- Tune data structures for performance
  - Attribute cache misses to data structures (not just the code causing the miss)
  - Support for custom memory allocators
  - Shows average load latency in cycles
- Optimize NUMA latency & scalability
  - Auto detect max system bandwidth
  - Detects inter-socket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



Intel® VTune™ Profiler

# Profile GPU Performance

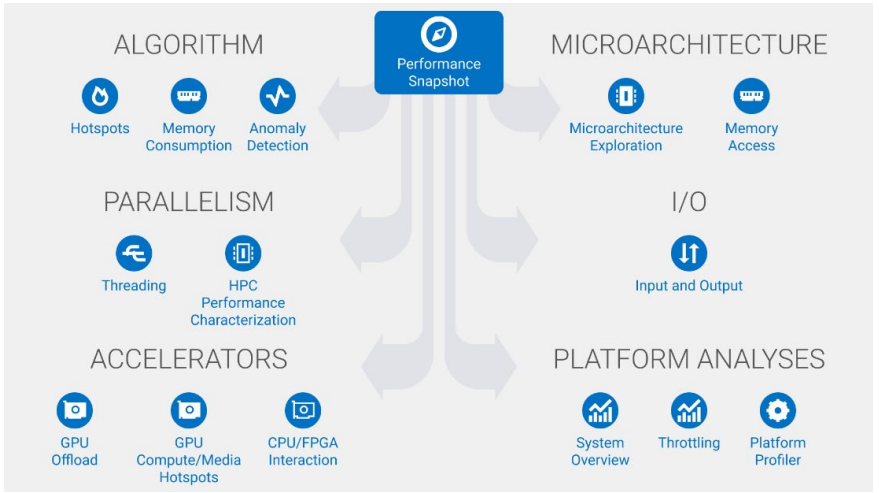
- Explicit support of **DPC++, DirectX, Intel® Media SDK, OpenCL™, and OpenMP-offload** software technology
- **Multi-GPU systems analysis**
- **GPU Offload cost profiling**
  - CPU vs GPU boundness
  - Offload overhead & host-to-device traffic, GPU compute vs data transfer
  - GPU utilization and software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

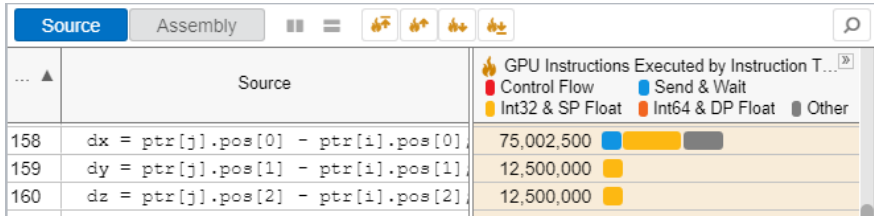


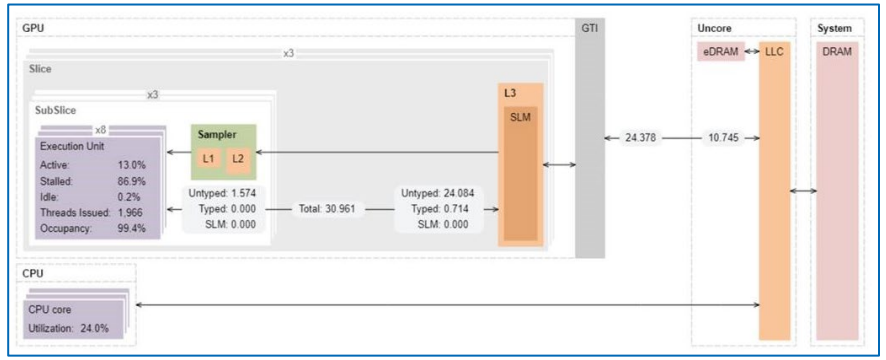
# GPU Performance Problems

Addressing performance issues with dynamic analysis tools

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



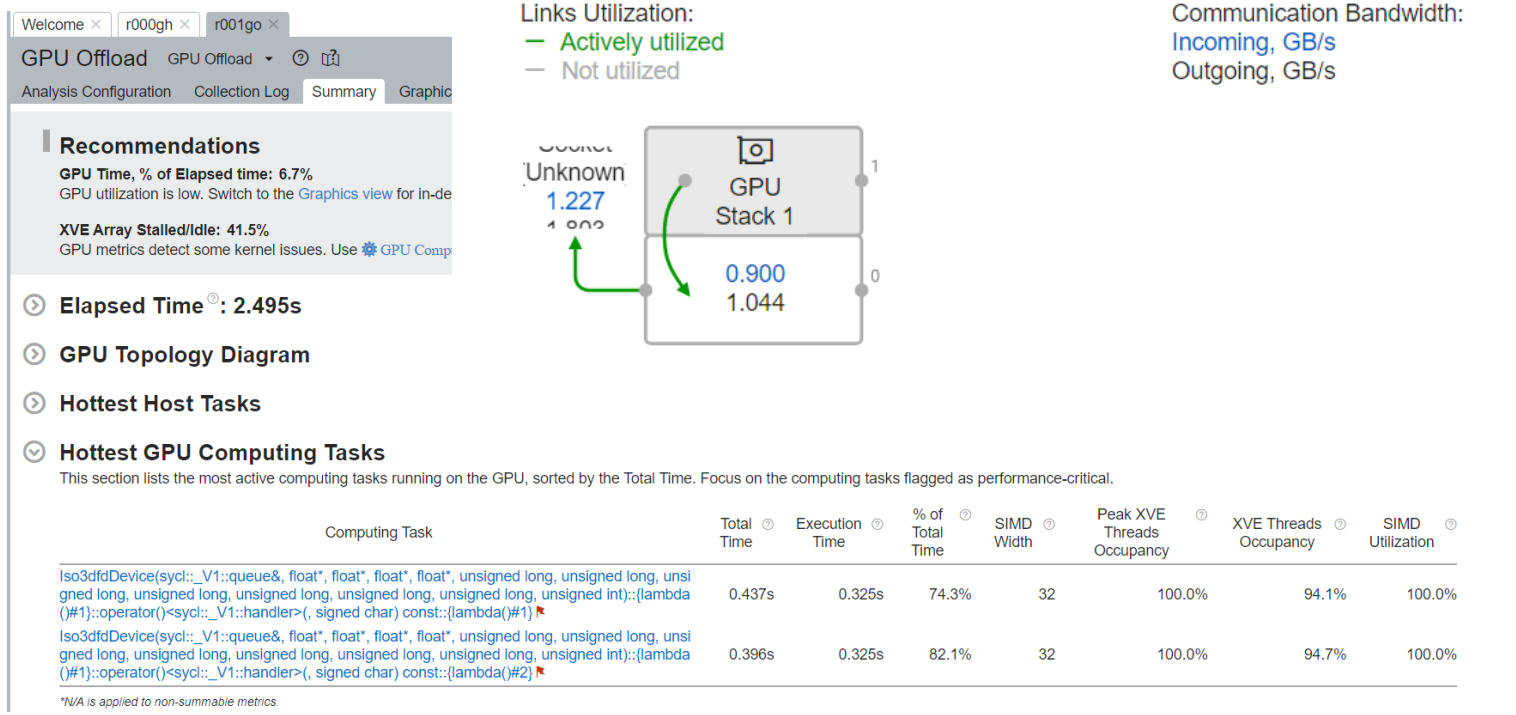



# Work Distribution

## Work distribution among computing resources

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

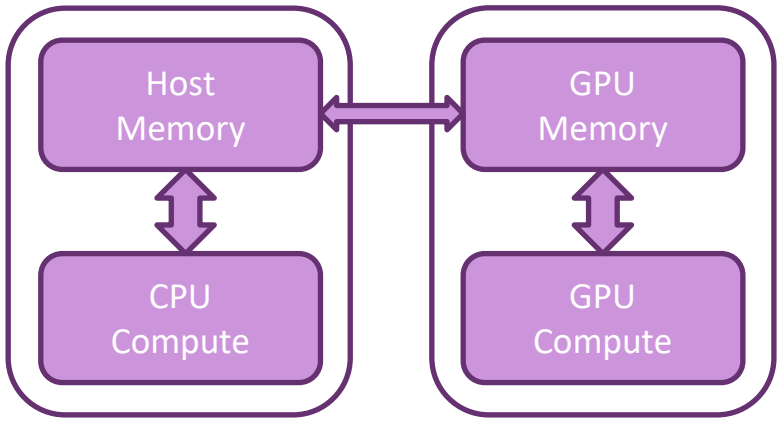



```
vtune -collect gpu-offload [-knob <knobName=knobValue>] [--] <app>
```

# Host and GPU Data Transferring

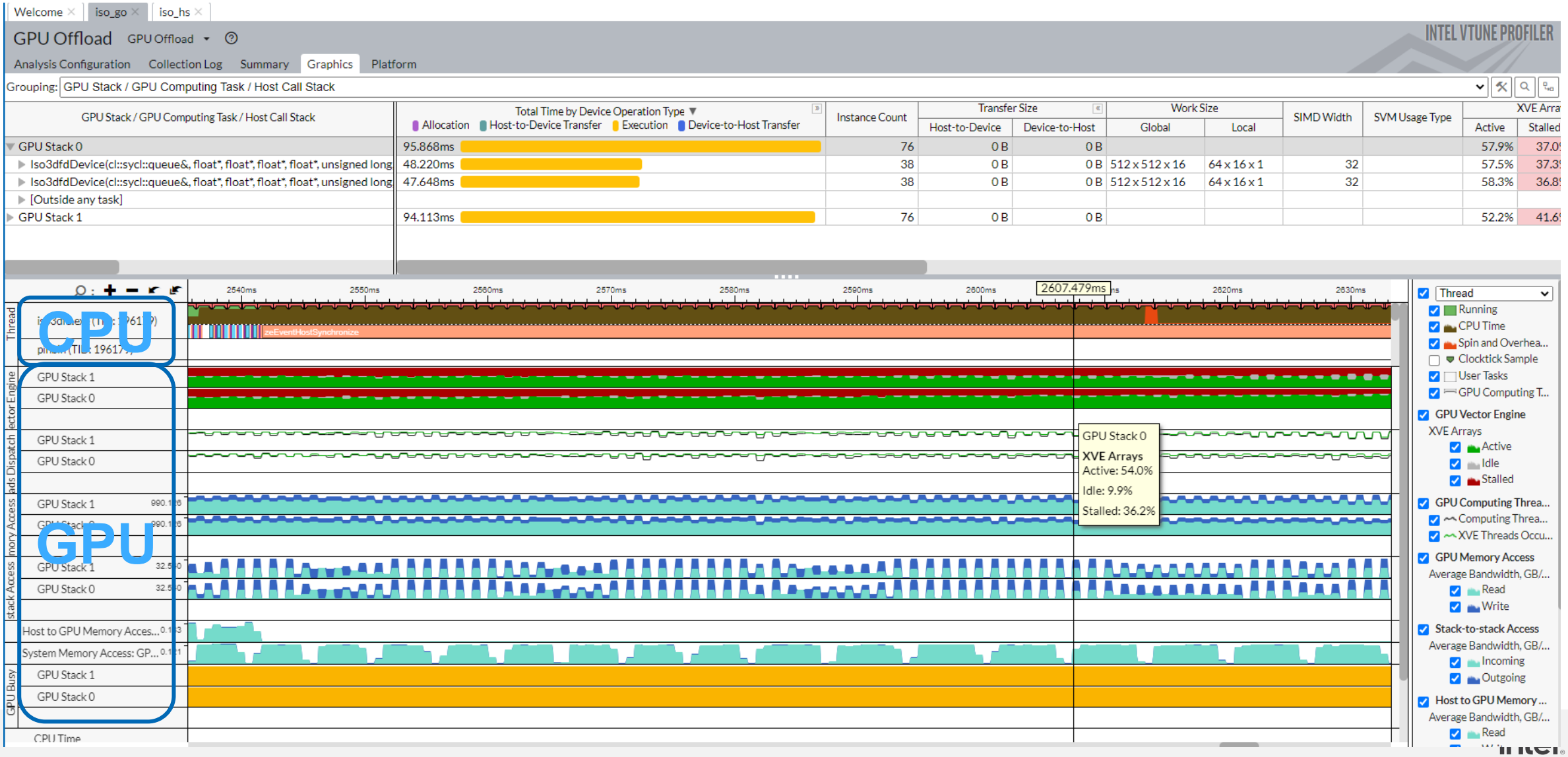
A commonly known problem of host-to-device transfer performance

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

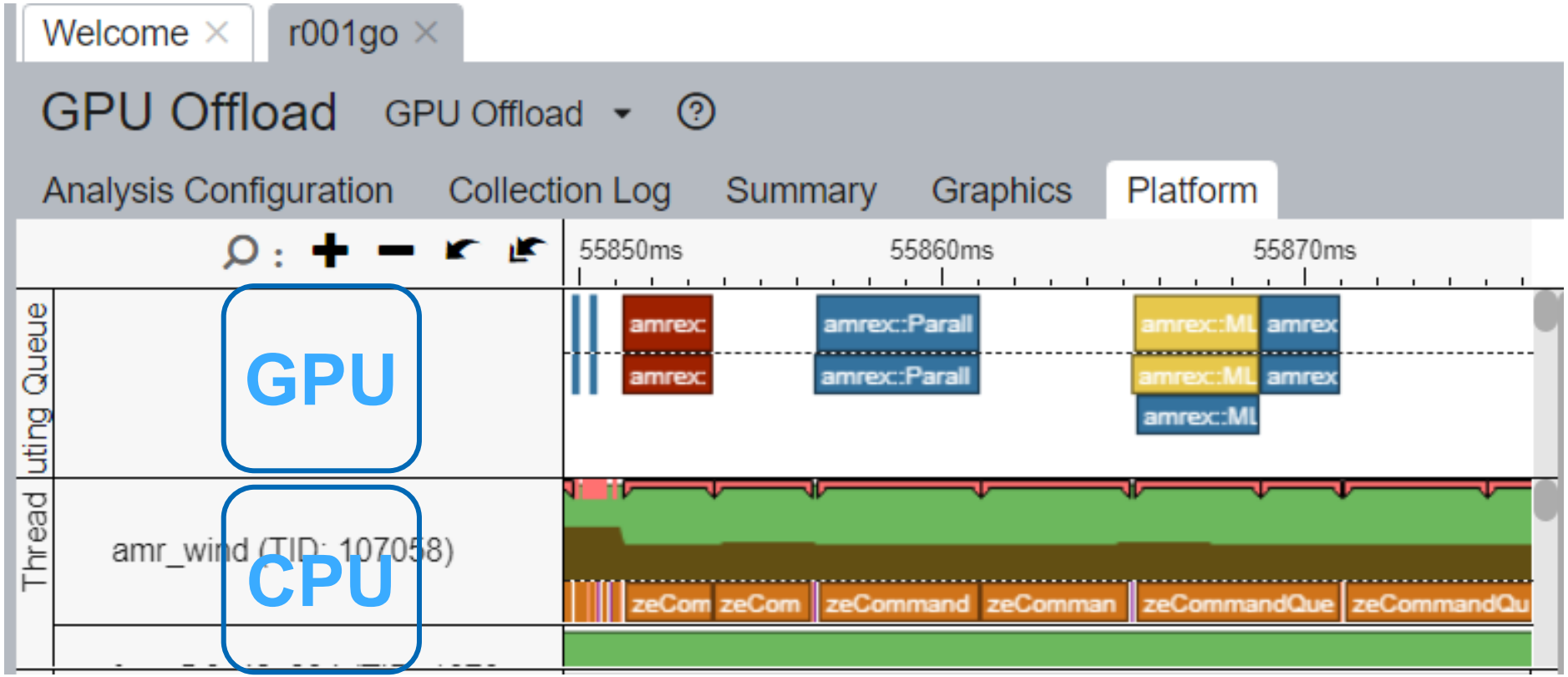



```
vtune -collect gpu-offload [-knob <knobName=knobValue>] [--] <app>
```

# Graphics View of GPU Offload



# Graphics View of GPU Offload

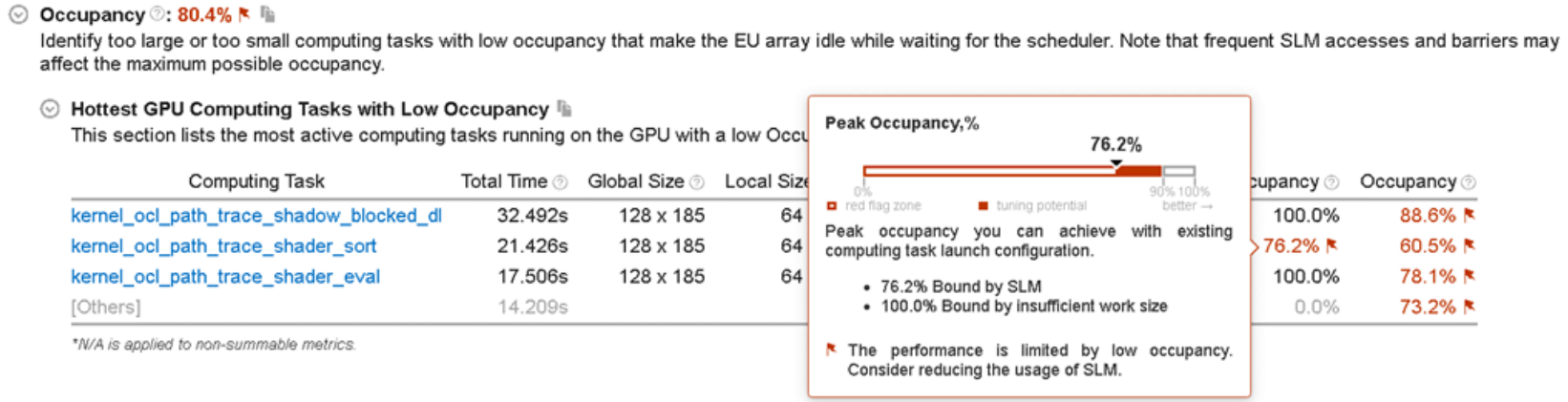


# Achieving High XVE Threads Occupancy

## 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

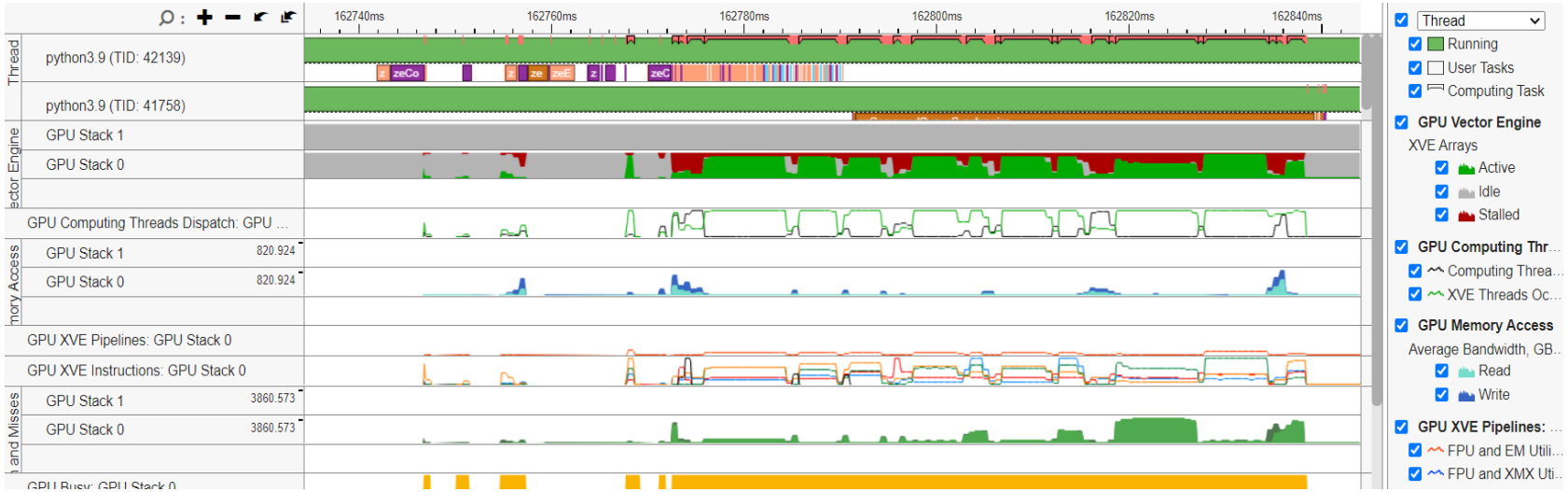



```
vtune -collect gpu-hotspots [-knob <knobName=knobValue>] [--] <app>
```

# 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 XMX pipeline Utilization



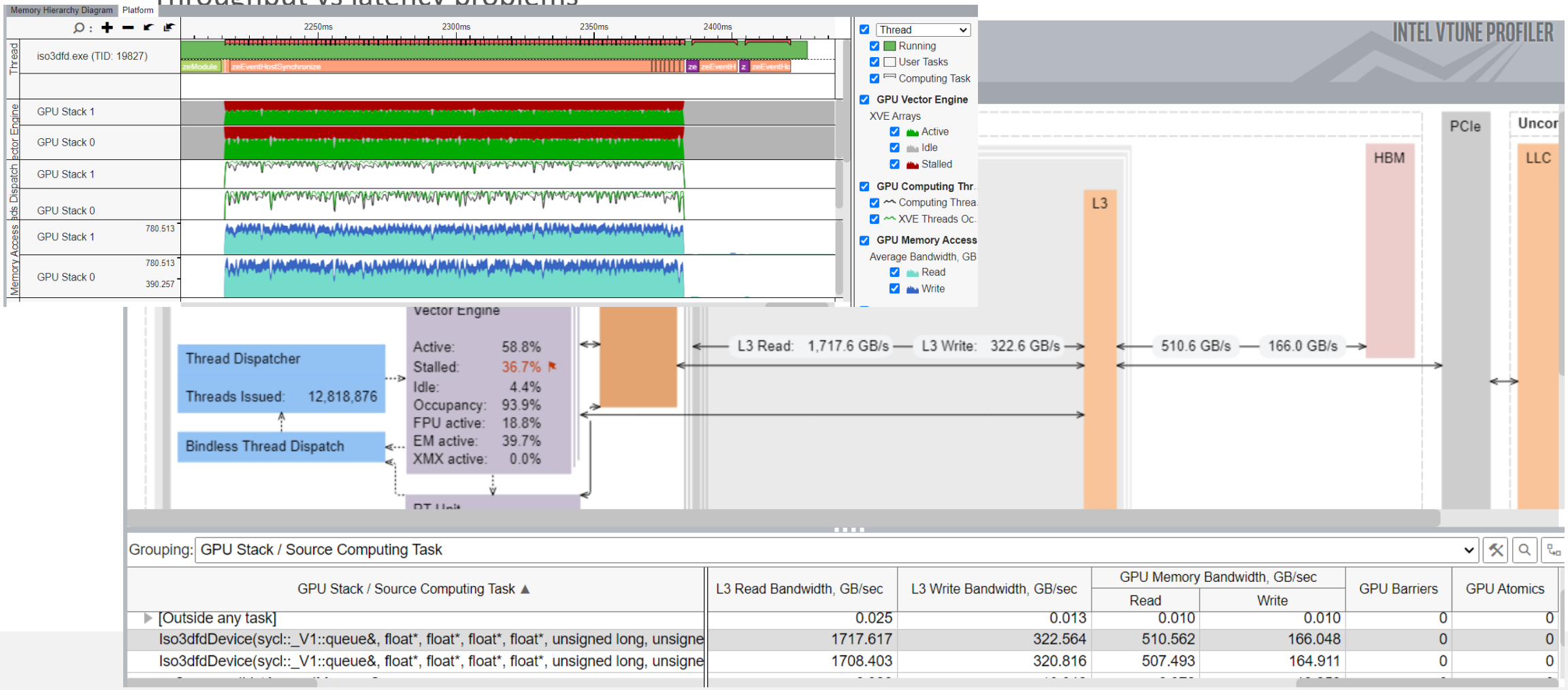
# oneDNN with Intel® XMX

GEMM with different levels of precisions


FP32 FP16 BF16 INT8

# Memory Access problems

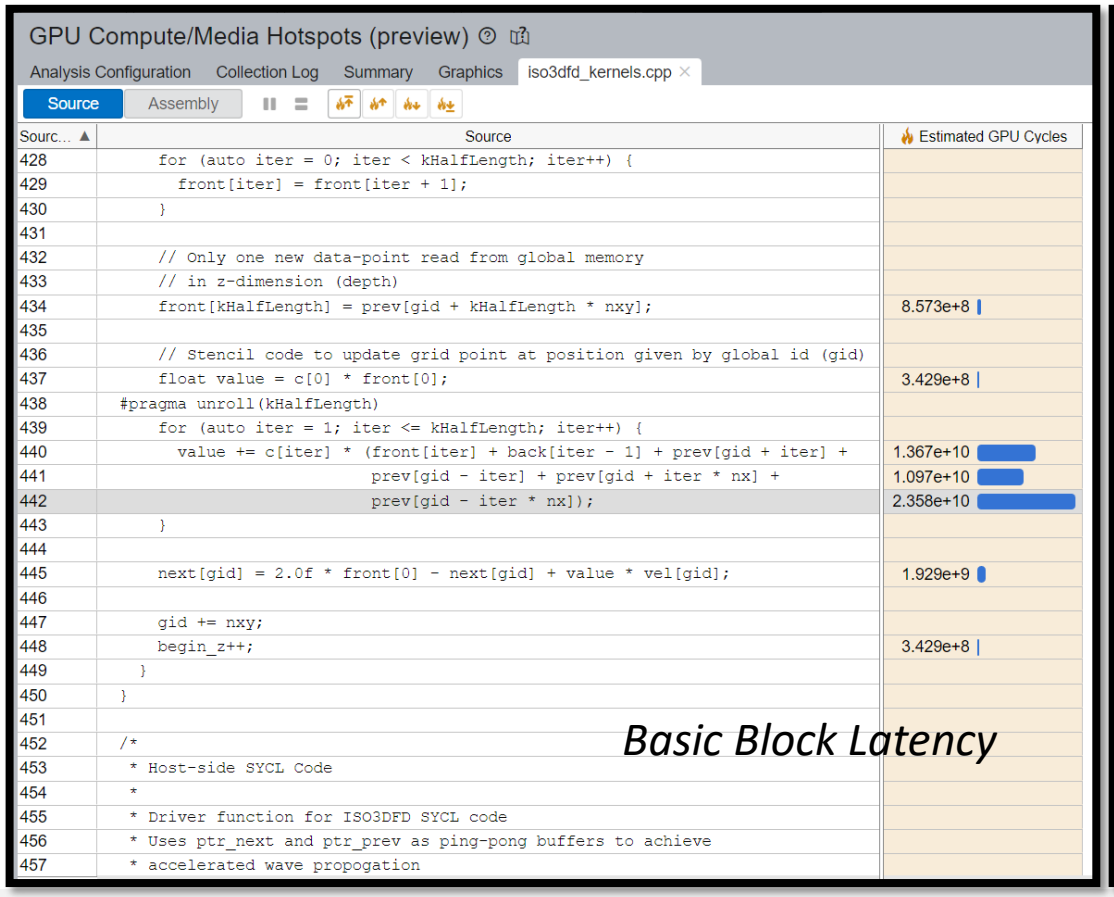
- Global memory access penalty
- Cache memory resource limit
- Throughput vs latency problems
- Which code is responsible for latency?
- Per Basic Block and latencies per individual instructions

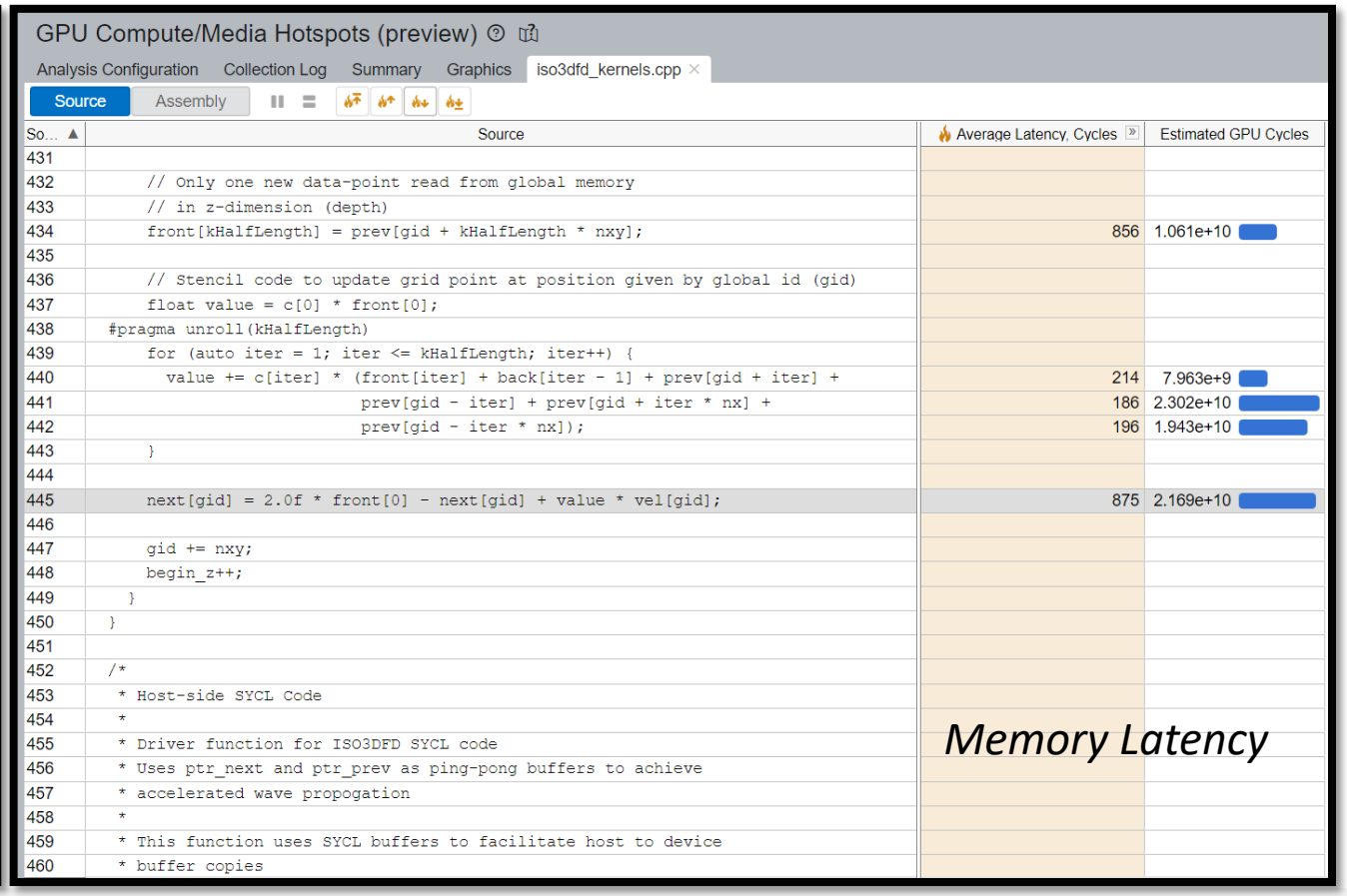



| GPU Stack / Source Computing Task ▲ | L3 Read Bandwidth, GB/sec | L3 Write Bandwidth, GB/sec | GPU Memory Bandwidth, GB/sec | | GPU Barriers | GPU Atomics |
|---|---|---|---|---|---|---|
| | | | Read | Write | | |
| ▶ [Outside any task] | 0.025 | 0.013 | 0.010 | 0.010 | 0 | 0 |
| Iso3dfdDevice(sycl::_V1::queue&, float*, float*, float*, float*, unsigned long, unsigne | 1717.617 | 322.564 | 510.562 | 166.048 | 0 | 0 |
| Iso3dfdDevice(sycl::_V1::queue&, float*, float*, float*, float*, unsigned long, unsigne | 1708.403 | 320.816 | 507.493 | 164.911 | 0 | 0 |

# 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>



# 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**

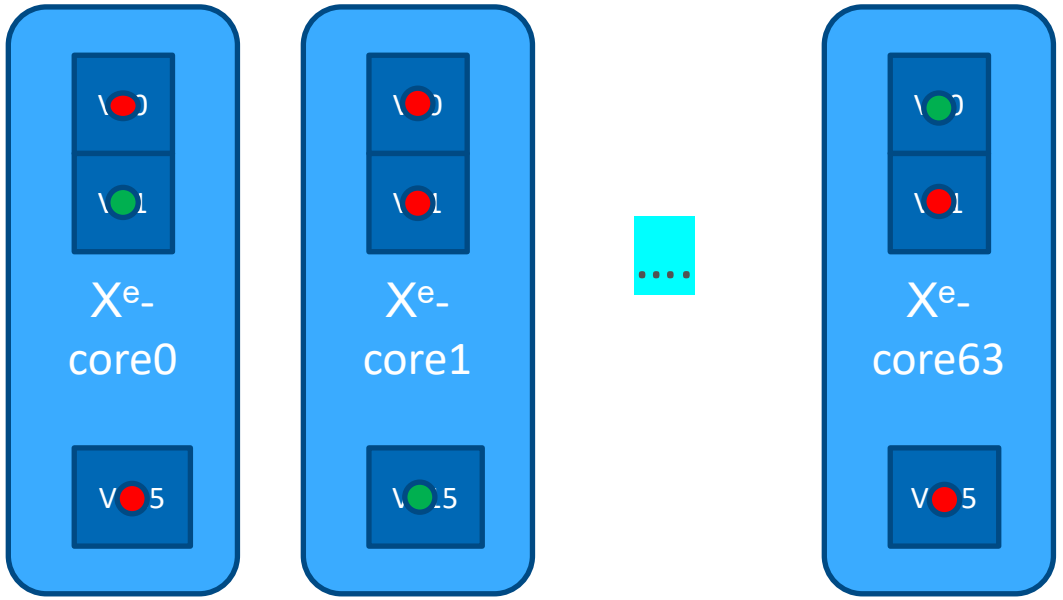
```
9   __kernel void spmv_jds_naive(__global float *ds        0.1%
10                               __global int *d_index,
11                              __global float *x_vec, cor
12                              __constant int *jds_ptr_ir
13                              __constant int *sh_zcnt_ir
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

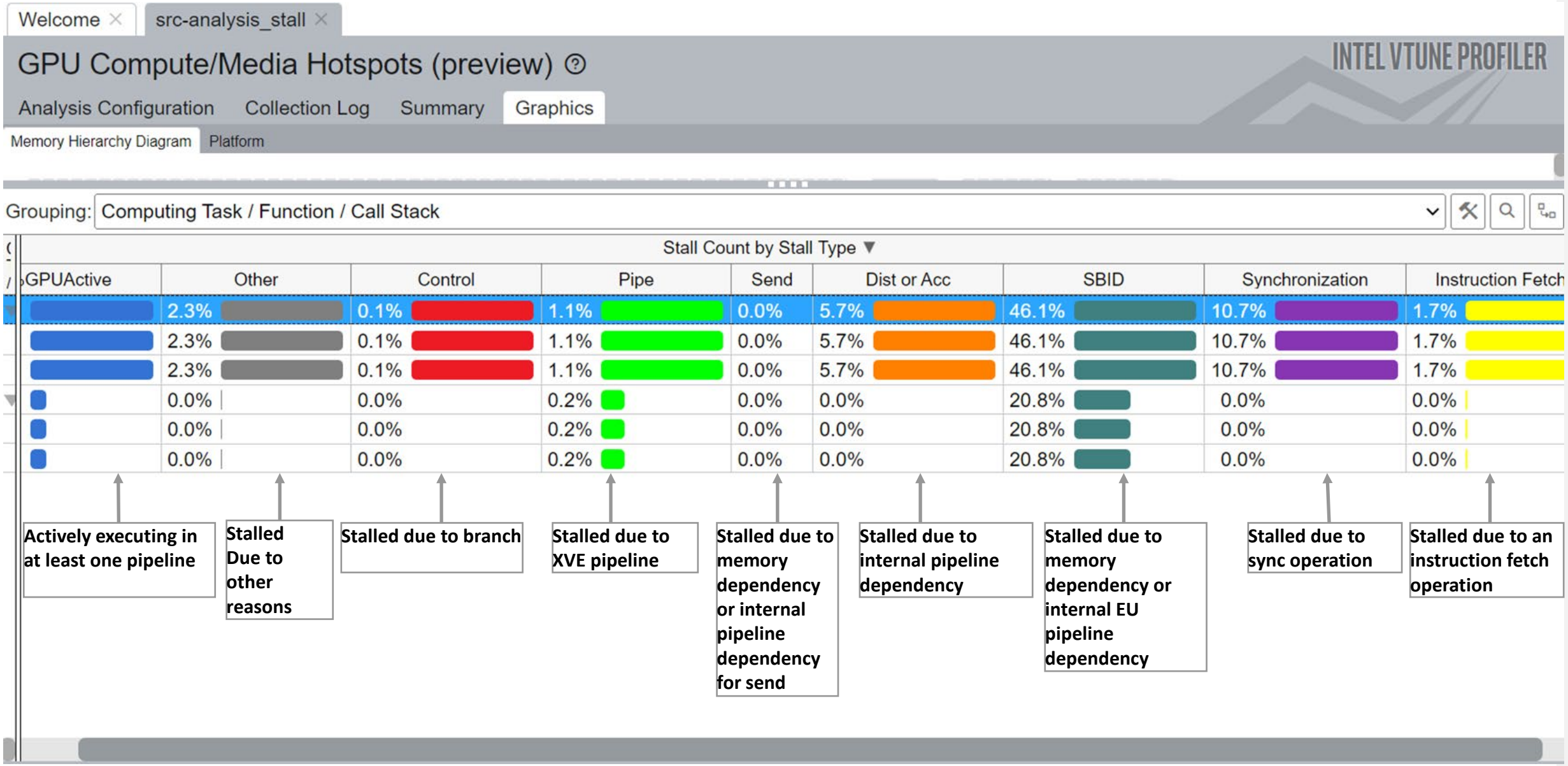
vtune -collect gpu-hotspots –knob profiling-mode=source-analysis -knob source-analysis=stall-sampling [--] <app>

# High-Level Stall Sampling (Xe-core)

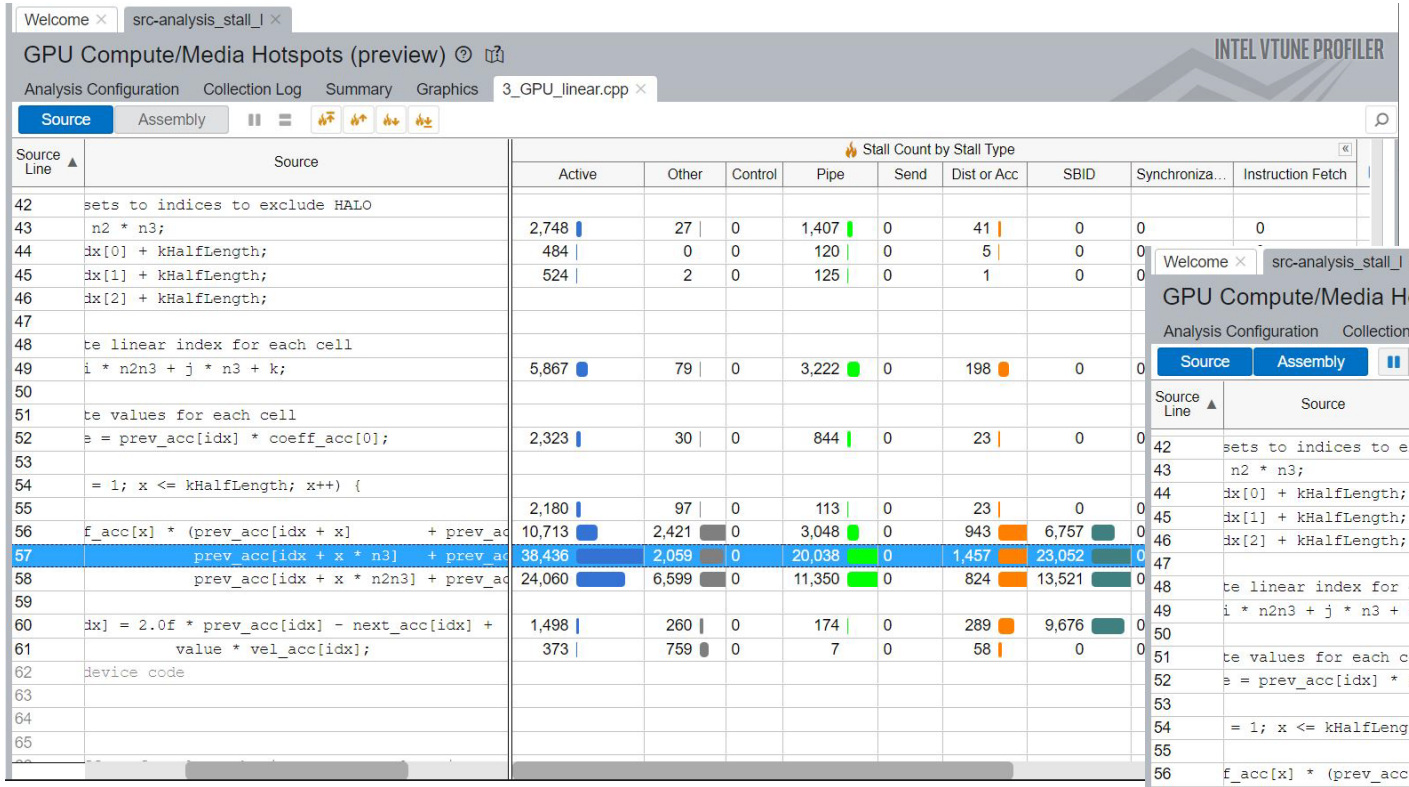
- Sample all Xe Vector Engines(XVE) statistically, one by one
  - Uses sampling interval large enough to make data manageable
  - Small enough to be representative
  - No assumption that all Xe Vector Engine are doing the same work
- Record both
  - Stall
  - Active
  - Idle (Ignore)

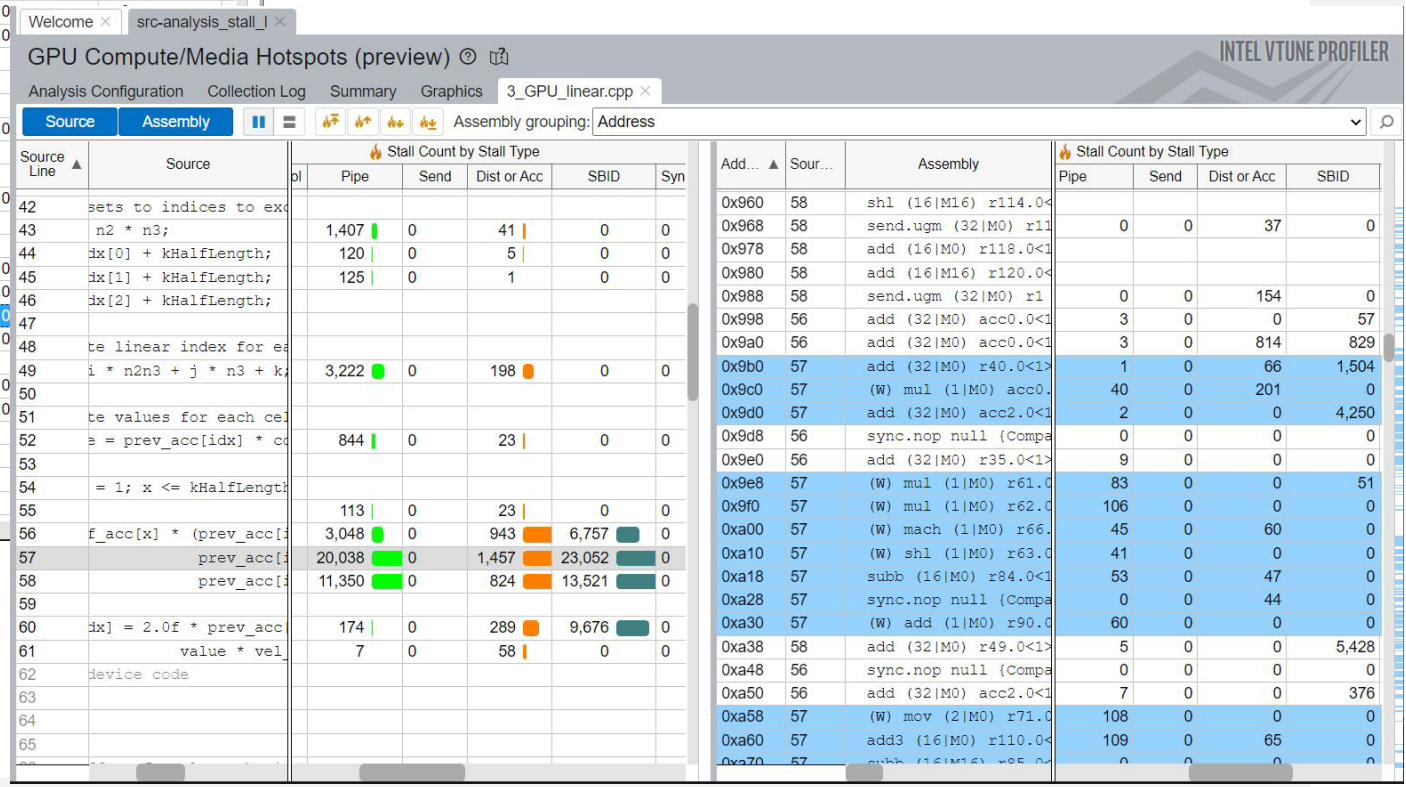



| State | Meaning |
|---|---|
| Idle | No threads are loaded on the XVE. Do nothing. |
| Active | At least one pipeline is dispatching an instruction on the sampled cycle |
| Stall | One or more threads are loaded on the EU, but no instruction is being dispatched to any pipeline |


Welcome
src-analysis_stall
GPU Compute/Media Hotspots (preview)
INTEL VTUNE PROFILER
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

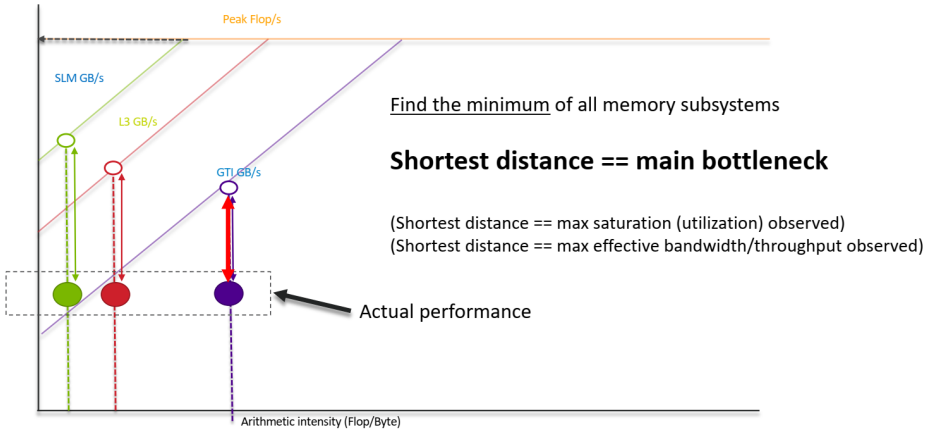
# Source Analysis view of Stall Sampling

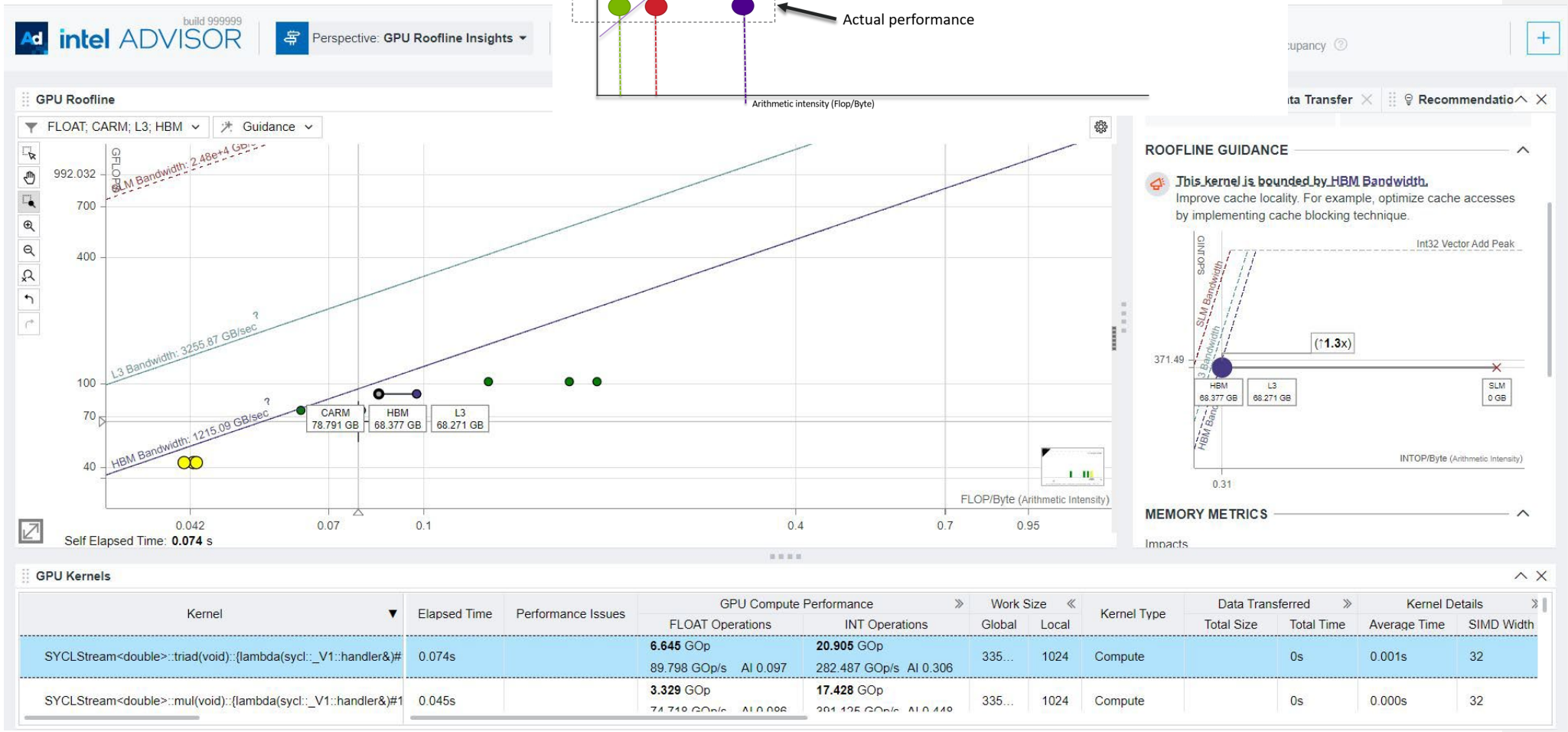


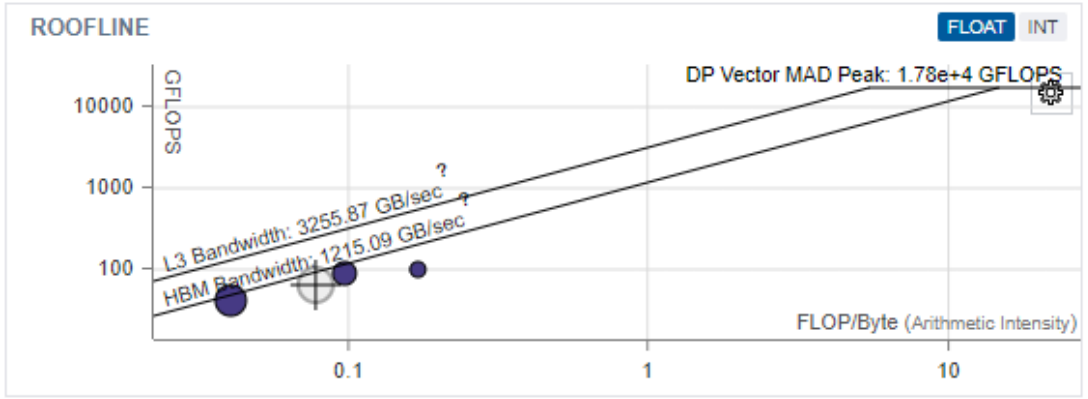
# GPU Roofline Analysis

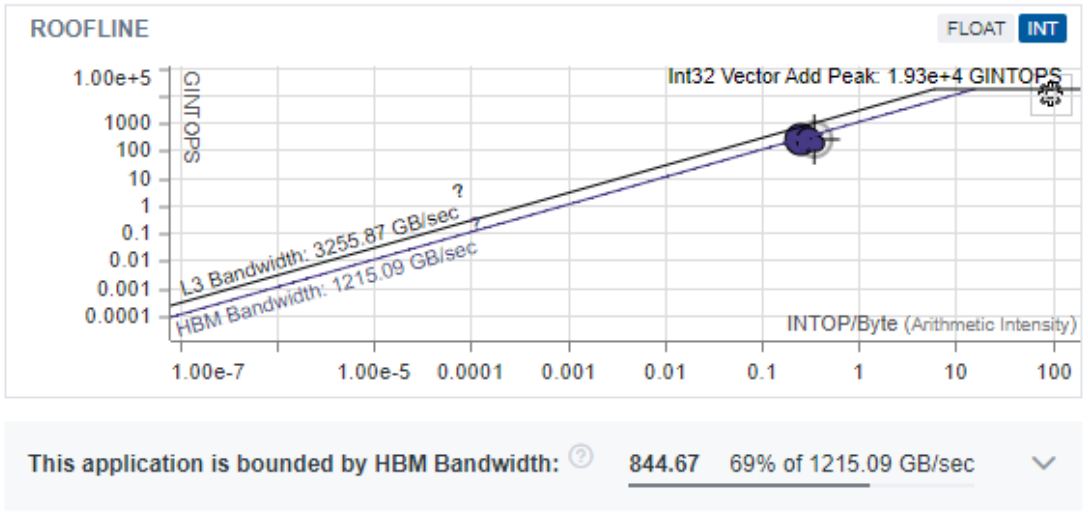
- kernels memory bound or compute bound?
- Compare the kernels performance to the capabilities of the GPU
- Memory Level Roofline - unambiguous bottleneck detection

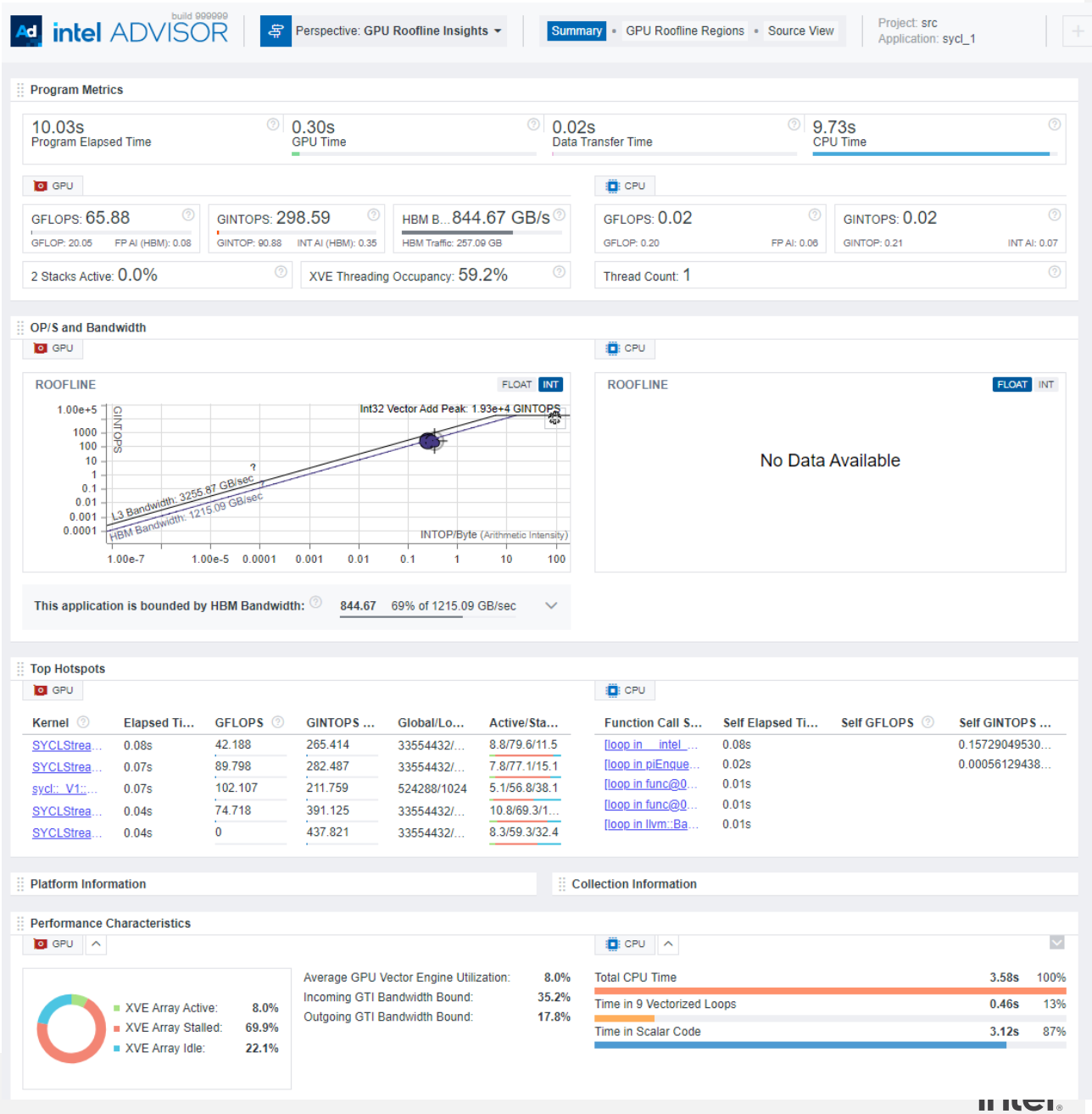


# GPU Roofline Analysis



# GPU Roofline in Intel® Advisor

**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>
```

- 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
```


intel