
Intel® oneAPI
IXPUG WEBINAR 2/20/20
ANALYZER TRAINING

AGENDA

- Introduction to Intel® oneAPI
- Intel® Advisor
  - Intel® Advisor – Offload Advisor
  - Intel® Advisor - GPU Roofline Analysis
- Intel® VTune™ Profiler
  - GPU Analysis

PROGRAMMING CHALLENGES FOR MULTIPLE ARCHITECTURES

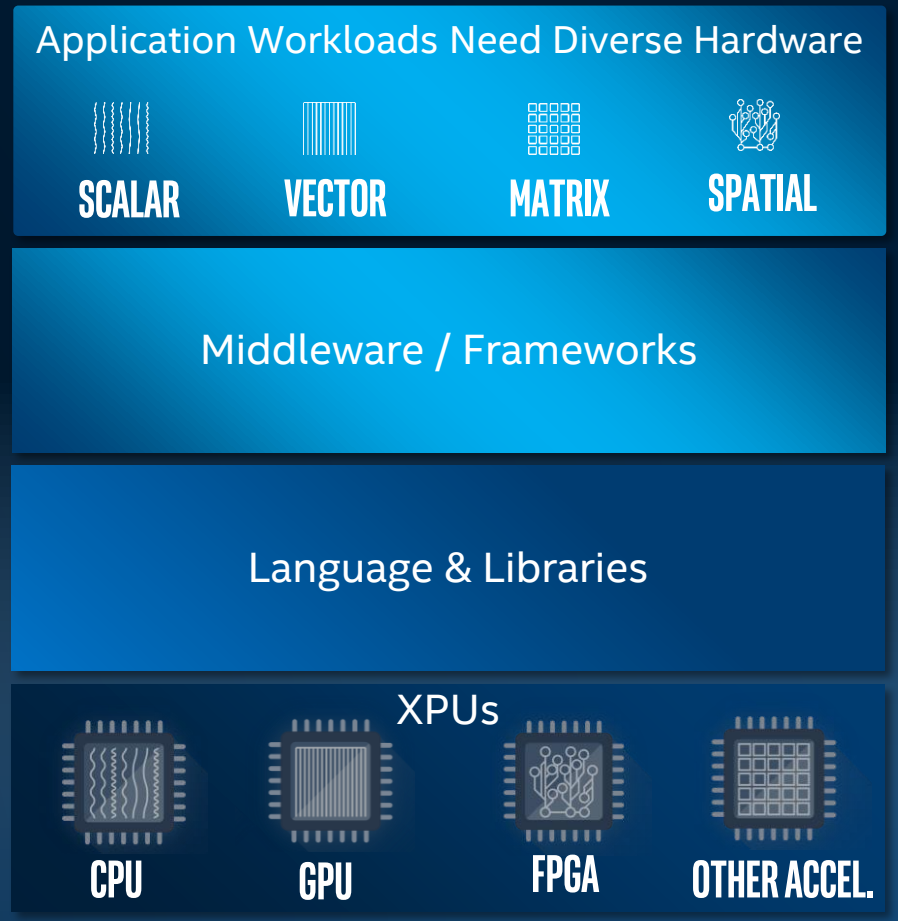
Growth in specialized workloads

No common programming language or APIs

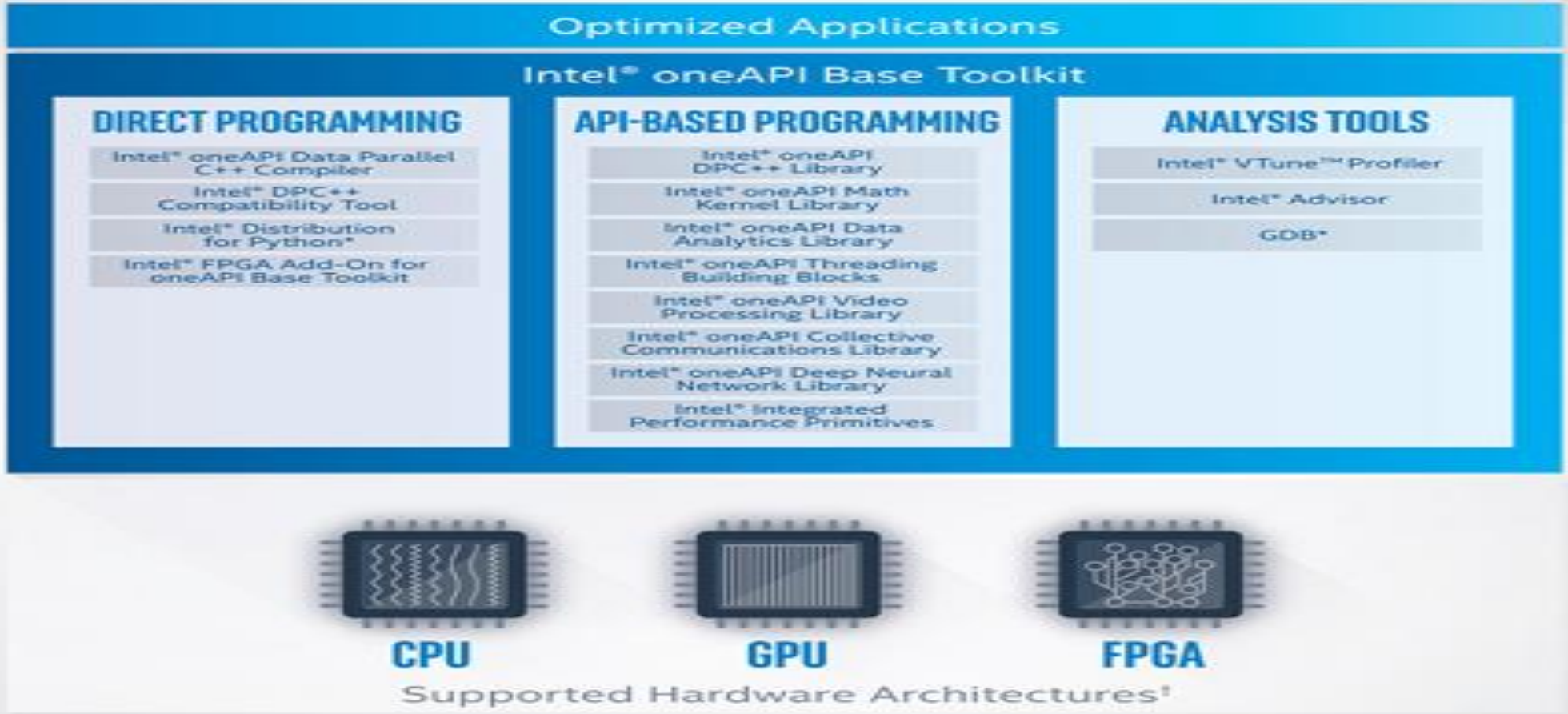
Inconsistent tool support across platforms

Each platform requires unique software investment

Diverse set of data-centric hardware required



INTEL® ONEAPI



Intel VTune™ Profiler vs Intel® Advisor

GPU ANALYSIS

Intel®Advisor

- Offload Advisor in design phase
  - Design code for efficient offloading to accelerators—even before you have hardware. Estimate code performance and compare it with data transfer costs. No recompilation required.
- GPU Roofline
  - Optimize your CPU or GPU code for compute and memory. Locate bottlenecks and identify performance headroom for each loop or kernel to prioritize which optimizations will deliver the highest performance payoff.

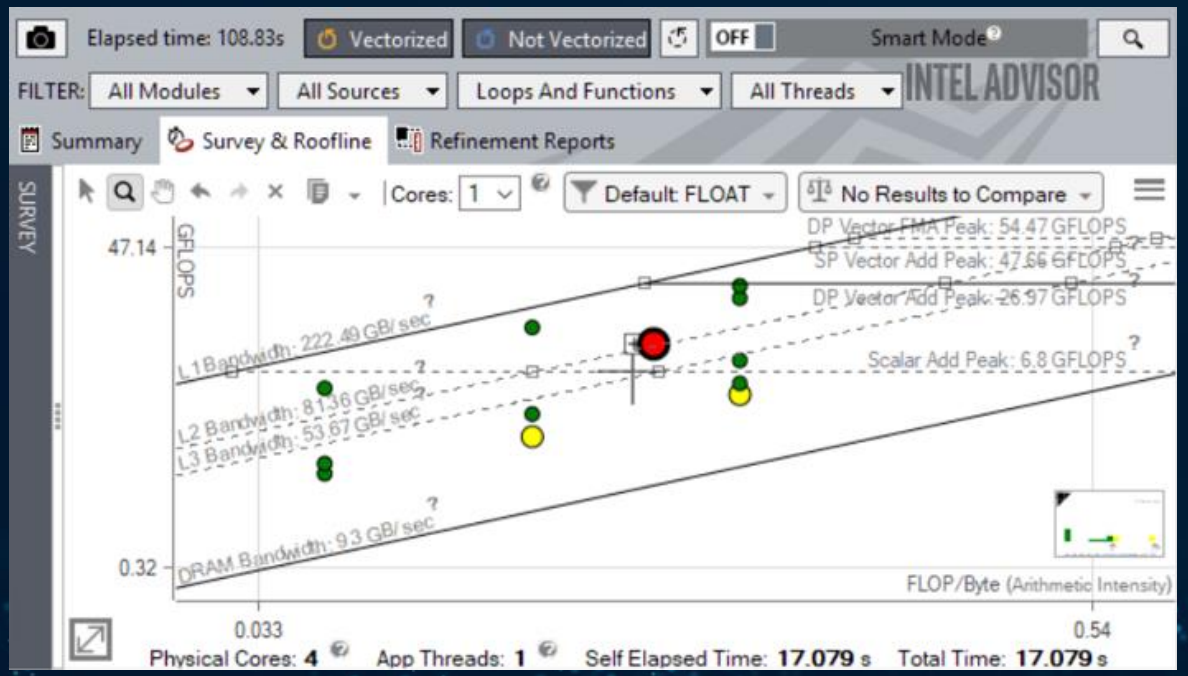
- Intel® VTune™ Profiler
  - GPU Offload to determine bottleneck
    - Explore code execution on various CPU and GPU cores on your platform, correlate CPU and GPU activity, and identify whether your application is GPU or CPU bound.
  - GPU Compute/Media Hotspots
    - Analyze the most time-consuming GPU kernels, characterize GPU usage based on GPU hardware metrics
    - GPU code performance at the source-line level and kernel assembly level

INTEL® ADVISOR

Vectorization Optimization, Thread Prototyping & Flow Graph Analysis

Design Modern Software with Intel® Advisor

Optimize Vectorization, Prototype Threading, Create & Analyze Flow Graphs



Learn More: software.intel.com/advisor

Capabilities

- Adds & optimizes vectorization
- Analyzes memory patterns
- Quickly prototypes threading
- Shows performance headroom for each loop
- Creates and analyzes flow graphs

New for 2020 Release (partial list)

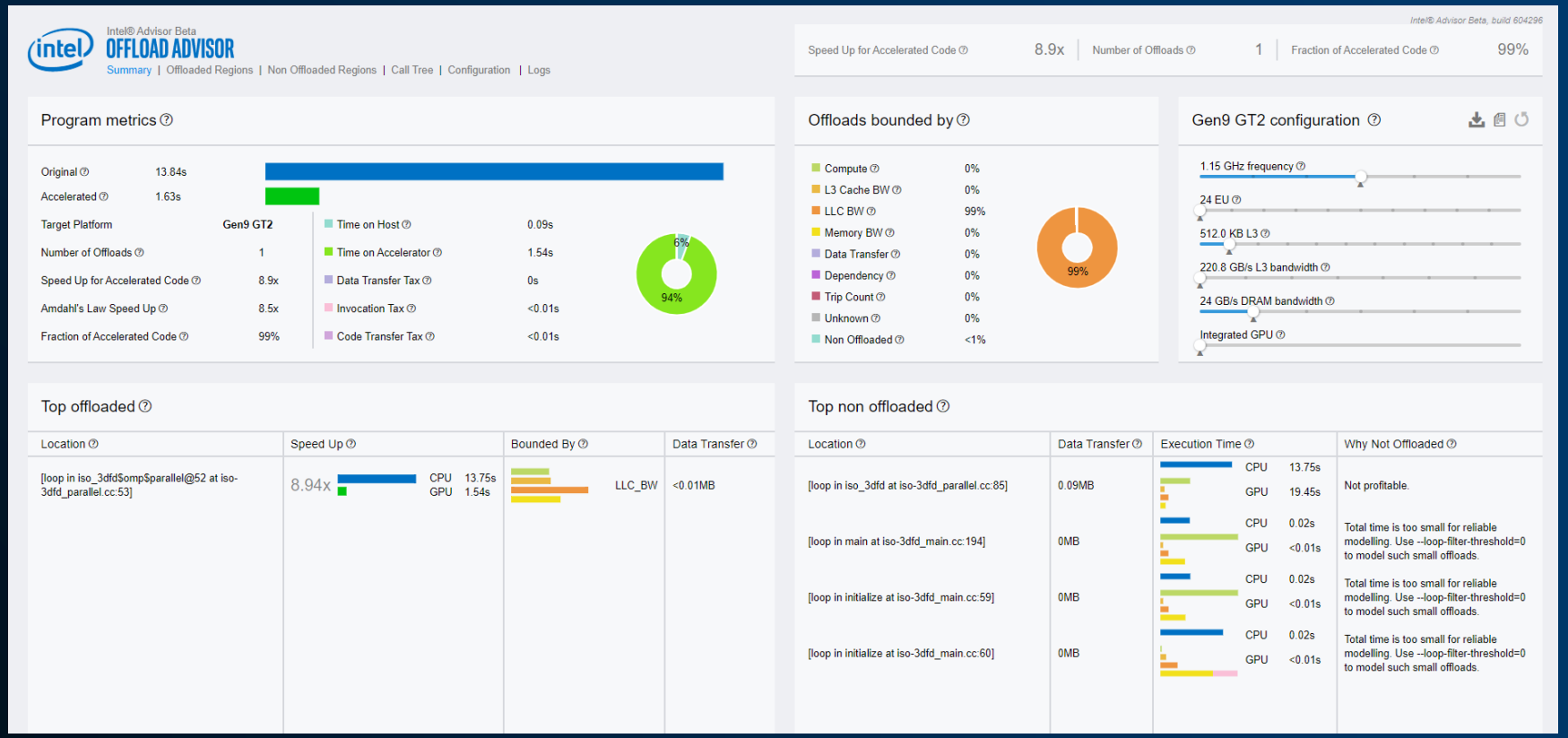
- AVX-512 VNNI support
- Roofline Guidance
  - Loop specific actionable optimization recommendations
- Roofline Compare
  - Visualize multiple Roofline results in one chart to track optimization progress
- Integrated Roofline (Technical Preview)
  - Pinpoint the cache level causing bottlenecks

INTEL® ADVISOR

OFFLOAD ADVISOR

INTEL® ADVISOR - OFFLOAD ADVISOR (BETA)

- Starting from a optimized binary (running on CPU):
  - Helps define which sections of the code should run on a given accelerator
  - Provides performance projection on accelerators (currently gen9 and gen11)



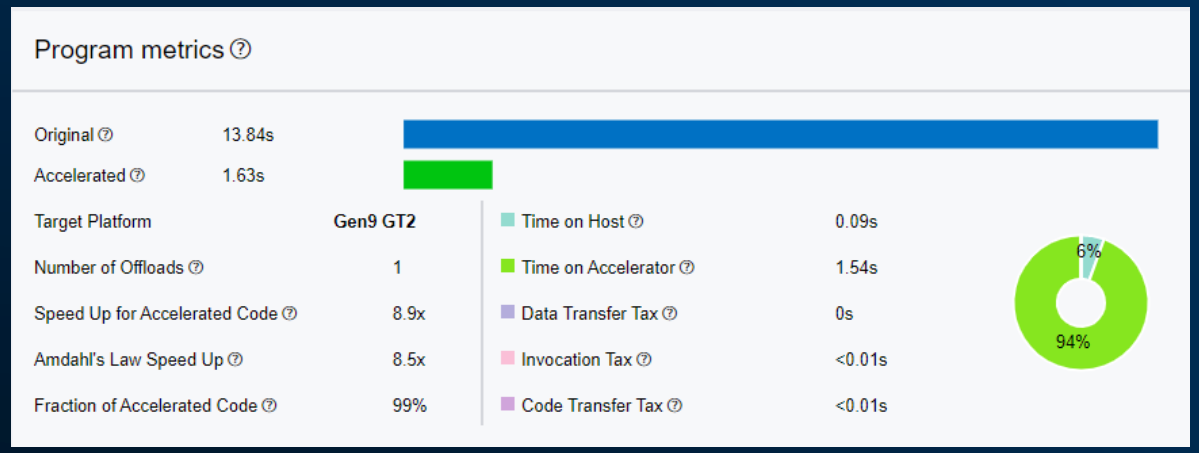
MODELING FLOWS SUPPORTED

	Baseline HW	(Programming model)		Target HW	
1.	**CPU** measured	(C,C++,Fortran, Py)	→	**CPU** + measured	**GPU** estimated
1.a	**CPU** measured	(DPC++, OCL, OMP, “target=host”)	→	**CPU** + measured	**GPU** estimated
2	**CPU+iGPU** measured	(C, C++, Fortran, DPC++, OCL, OMP)	→	**CPU** + measured	**GPU** estimated

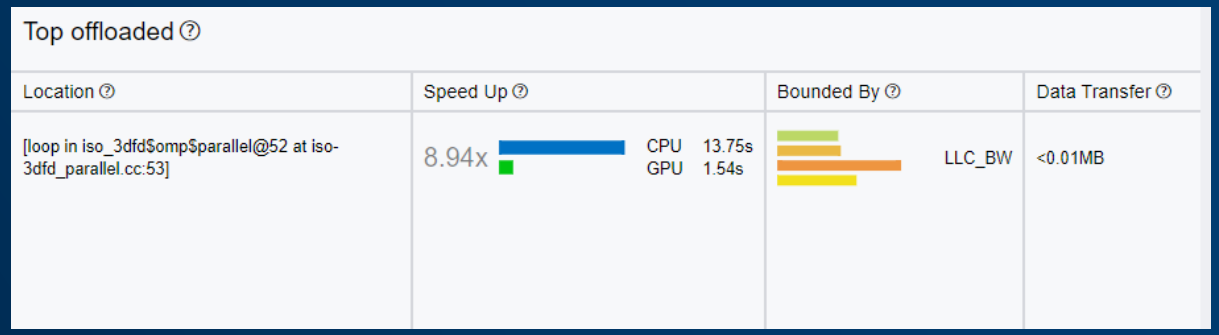
Coming soon!

WILL OFFLOAD INCREASE PERFORMANCE?

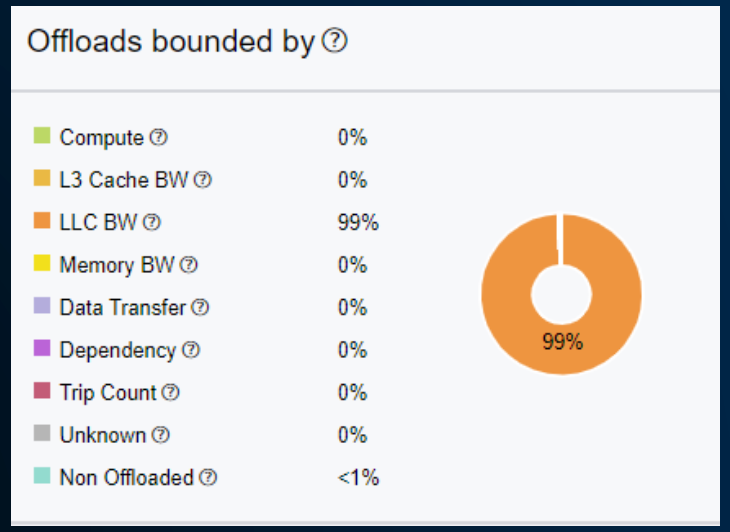
- How your code might perform on an accelerator ?



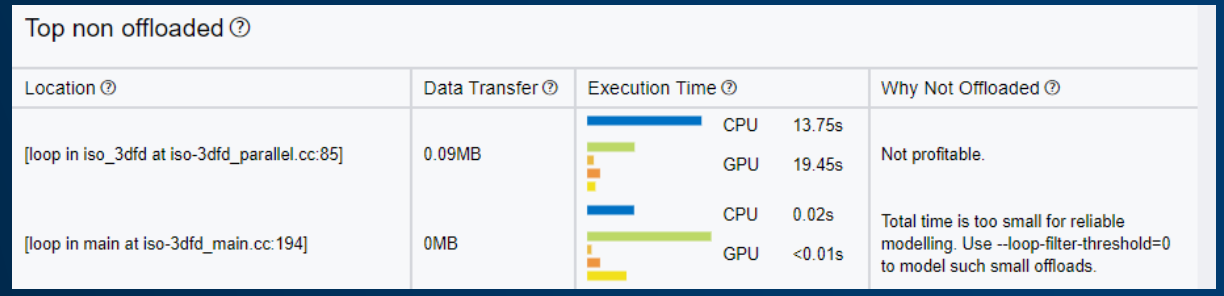
- What should you offload ?



- What might be limiting your performance on the accelerator ?



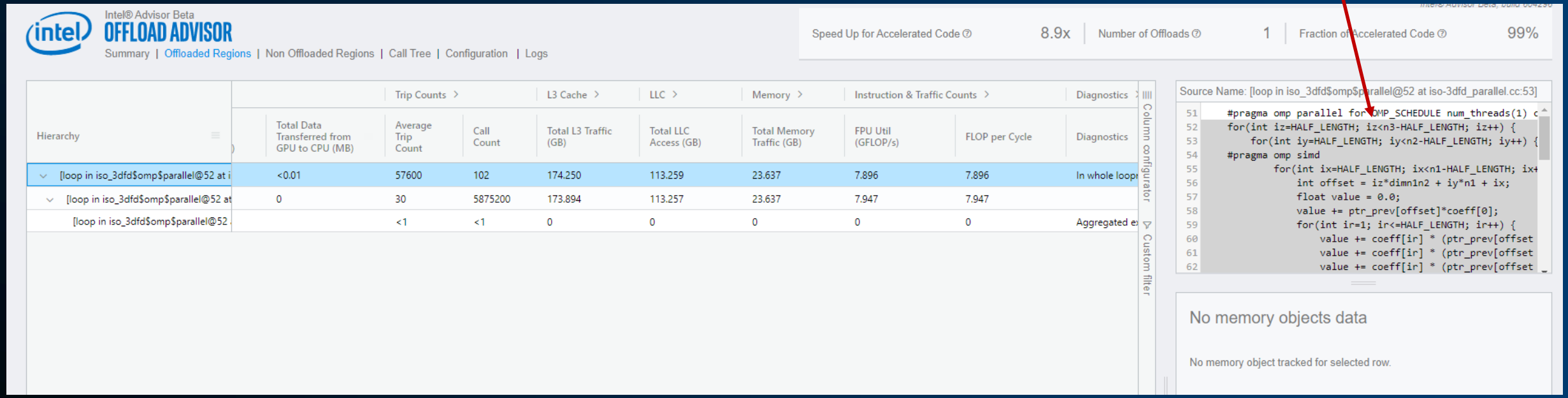
- What are the bad candidates for offload and Why ?



IN DEPTH ANALYSIS OF TOP OFFLOAD REGIONS

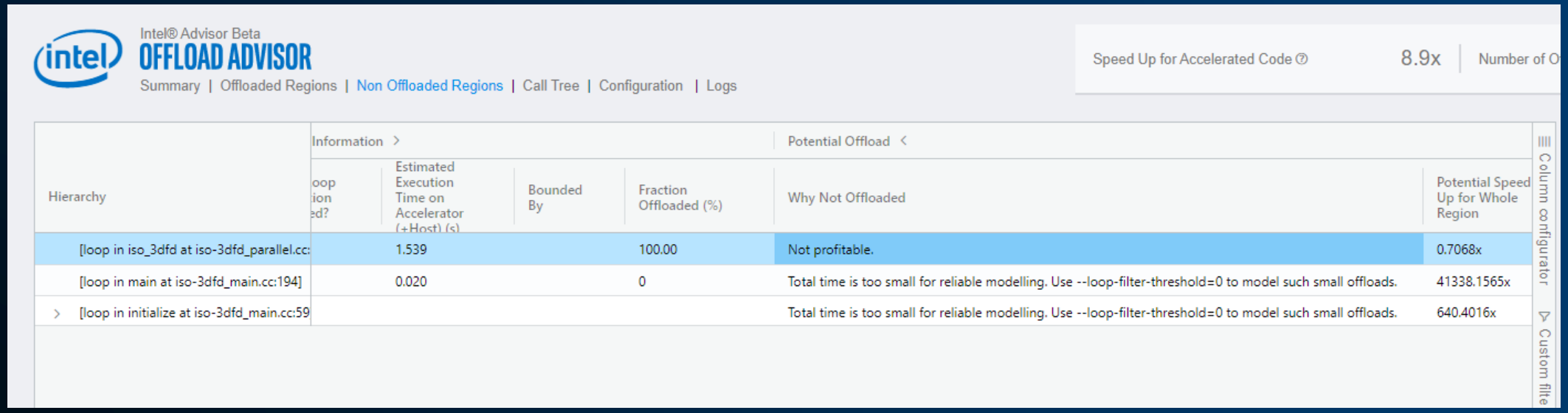
- Provides a detailed description of each loop interesting for offload
  - Timings (total time, time on the accelerator, speedup)
  - Offload metrics (offload taxe, data transfers)
  - Memory traffic (DRAM, L3, L2, L1), trip count
  - Highlight which part of the code should run on the accelerator

This is where you will use DPCPP or OMP target for offload



WHAT SHOULD NOT BE OFFLOADED

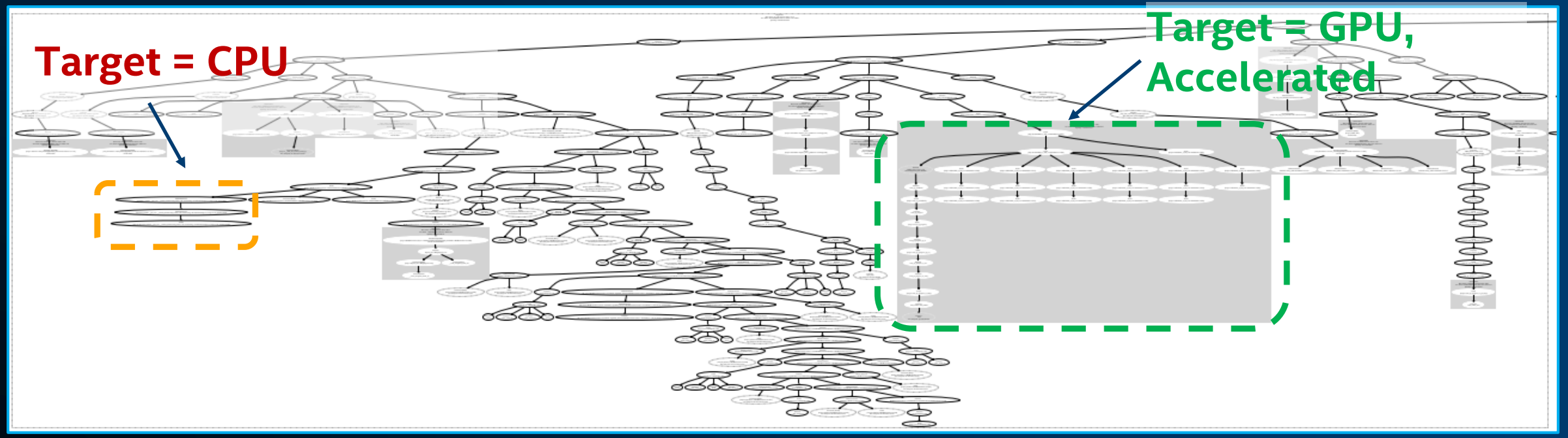
- Explains why Intel® Advisor doesn't recommend a given loop for offload
  - Dependency issues
  - Not profitable
  - Total time is too small



ANOTHER VIEW OF YOUR CODE

PROGRAM TREE

The program tree offers another view of the proportion of code that can be offloaded to the accelerator.



BEFORE YOU START

INTEL® ADVISOR OFFLOAD ADVISOR

- The only strict requirement for compilation and linking is full debug information:

  `-g:` Requests full debug information (compiler and linker)

- Offload Advisor supports any optimization level, but the following settings are considered the optimal requirements:

  `-O2:` Requests moderate optimization

  `-no-ipo:` Disables inter-procedural optimizations that may inhibit Offload Advisor to collect performance data (Intel® C++ & Fortran Compiler specific)

SETTING UP YOUR ENVIRONMENT

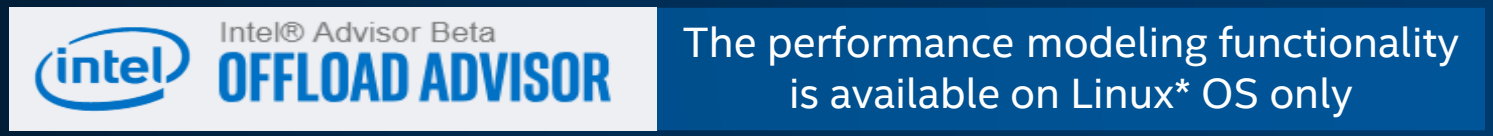
- To set up the Intel® Advisor Beta environment, run one of the shell script:

  ```
  source <ONEAPI_INSTALL_DIR>/setvars.sh
  ```

  or

  ```
  source <ADV_INSTALL_DIR>/env/vars.sh
  ```

- This script sets all required Intel Advisor environment variables, including APM, which points to `<ADV_INSTALL_DIR>/perfmodels`
- This is the location of the Offload Advisor scripts in the Intel® Advisor Beta installation directory



HOW TO RUN

- Easy to collect data and generate output with batch mode:

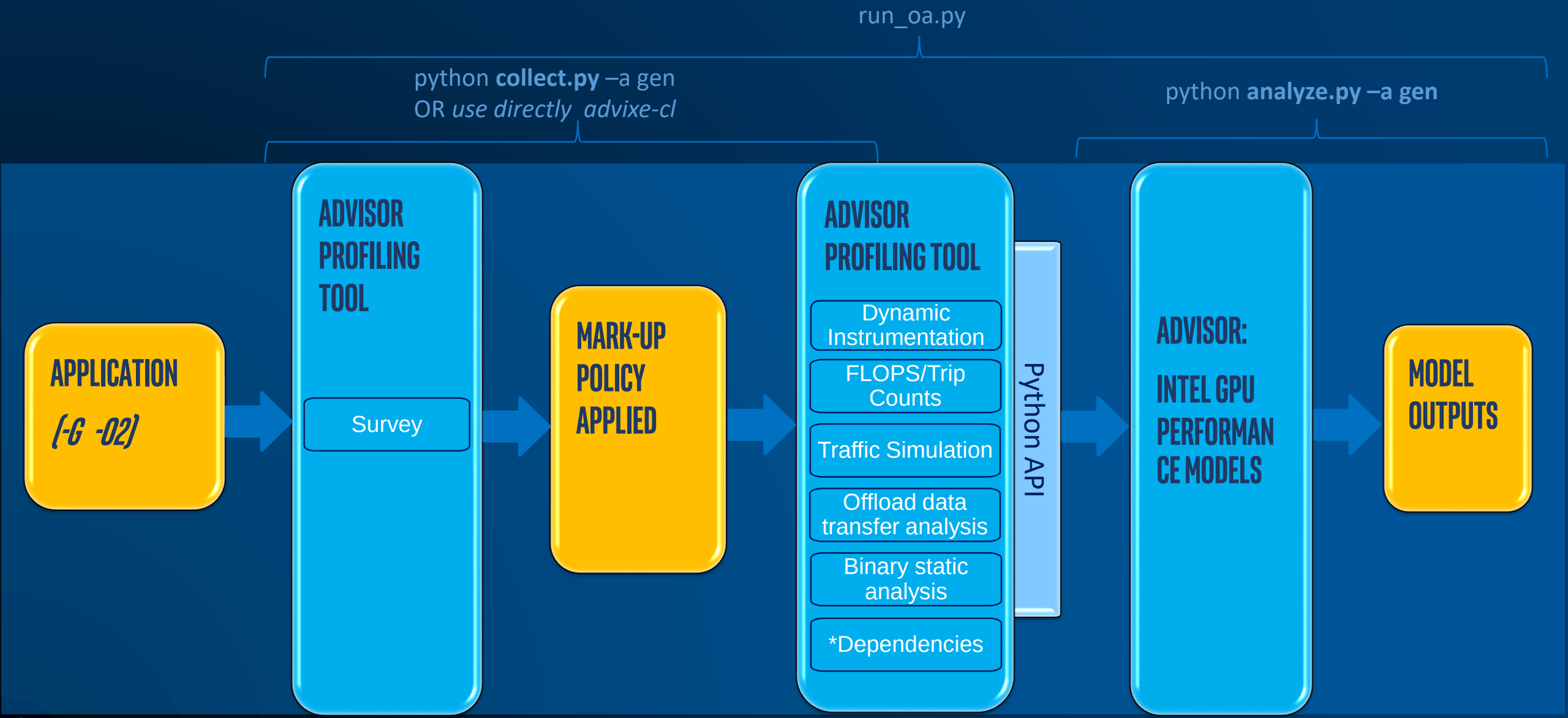
```
advixe-python <ADV_INSTALL_DIR>/perfmodels/run_oa.py <path_to_result_dir>
-config gen9 --out-dir <path_to_result_dir> [--options] -- <app>
```

- By default, **run_oa.py** marks up all regions and only selects the most profitable ones for analysis
- To generate the report.html, uses the following command:

```
advixe-python $APM/analyse.py <project_dir> --config gen9 [--options] --
<app_binary> [app_options]
```

```
u31313@s001-n004:/opt/intel/inteloneapi/advisor/latest/perfmodels$ ls
accelerators     analyze.py       collect.py       debug.so      environ.py  oa_wrapper.so  shared.so  toml
analyze_impl.so  collect_impl.so  compute_stats.py  dot_graph.so  helpers     run_oa.py      template   tree.so
```

HOW TO RUN



INTEL ADVISOR® - OFFLOAD ADVISOR OUTPUT

- **`report.html:`** Main report in HTML format
- `report.csv` and `whole_app_metric.csv:` Comma-separated CSV files
- `program_tree.dot:` A graphical representation of the call tree showing the offloadable and accelerated regions
- **`program_tree.pdf:`** A graphical representation of the call tree

  Generated if the DOT(GraphViz*) utility is installed

  1:1 conversion from the `program_tree.dot` file
- `JSON` and `LOG` files that contain data used to generate the HTML report and logs, primarily used for debugging and reporting bugs and issues

HOW TO SPEED UP COLLECTION

AVOID DEPENDENCY CHECKING

- Dependency adds a lot of time to the collection and you might want to remove it.
- Add the option –c basic for the collection:

```
advixe-python <ADV_INSTALL_DIR>/perfmodels/run_oa.py <path_to_result_dir>
–config gen9 –c basic --out-dir <path_to_result_dir> [--options] -- <app>
```

- Add the option --assume-parallel for the analysis:

```
advixe-python $APM/analyse.py <project_dir> --assume-parallel --config
gen9 [--options] -- <app_binary> [app_options]
```

GET BETTER CONTROL OF YOUR RUN

INDIVIDUAL LOW LEVEL COMMANDS

- You might want to run the command lines independently to tweak the parameters
- A good start is to use run_oa.py script with --dry-run to see the list of command lines and retrieve the cache configuration of the target accelerator.
- The next command will output the different command lines for doing separate analyses without running advisor collection.
- **advixe-python <ADV_INSTALL_DIR>/perfmodels/run_oa.py <path_to_result_dir> --dry-run –config gen9 –c basic --out-dir <path_to_result_dir> [--options] -- <app>**

GET BETTER CONTROL OF YOUR RUN

INDIVIDUAL LOW LEVEL COMMANDS

- We start with the survey
- `advixe-cl --collect=survey --auto-finalize --stackwalk-mode=online --static-instruction-mix --project-dir=./oa_report – my_app`
- The survey times your application and run some static analysis on the binary without impact on the application's performance.
  - Sampling
  - Binary static analysis
  - Static code analysis (compiler and debug infos)

GET BETTER CONTROL OF YOUR RUN

INDIVIDUAL LOW LEVEL COMMANDS

- We continue with the trip count and cache simulation
- `advixe-cl --collect=tripcounts -return-app-exitcode -flop -stacks -auto-finalize -ignore-checksums -enable-data-transfer-analysis -track-heap-objects -profile-jit -cache-sources -enable-cache-simulation -cache-config=1:8w:32k/1:64w:512k/1:16w:8m --project-dir=./oa_report – my_app`
- The tripcounts with –flop and –cache-simulation counts:
  - The number of iterations in your loops
  - The number of operations
  - Evaluate the data transfers between memory subsystems configured with –cache-config
- This analysis has usually =~10x slowdown

GET BETTER CONTROL OF YOUR RUN

INDIVIDUAL LOW LEVEL COMMANDS

- Optional step: Dependency analysis
- `advixe-cl --collect=dependencies --loops="total-time>5" --filter-reductions --loop-call-count-limit=16 --project-dir=./oa_report – my_app`
- Detects data dependencies in your loop by checking your memory accesses
- This analysis has an important impact on the performance
- It is up to the user to define how loops will be selected for this anlysis, here we use –loops="total-time>5" which select all loops impacting more than 5% of the overall time

GET BETTER CONTROL OF YOUR RUN

INDIVIDUAL LOW LEVEL COMMANDS

- Last step: Generating the report
- 2 Cases:
  - You ran the dependency analysis:

```
advixe-python $APM/analyse.py ./oa_report --config gen9 --out-dir
oa_report – my_app
```

  - You didn’t run the dependency analysis

```
advixe-python $APM/analyse.py ./oa_report --config gen9 --assume-
parallel --out-dir oa_report – my_app
```

MODELING PERFORMANCE

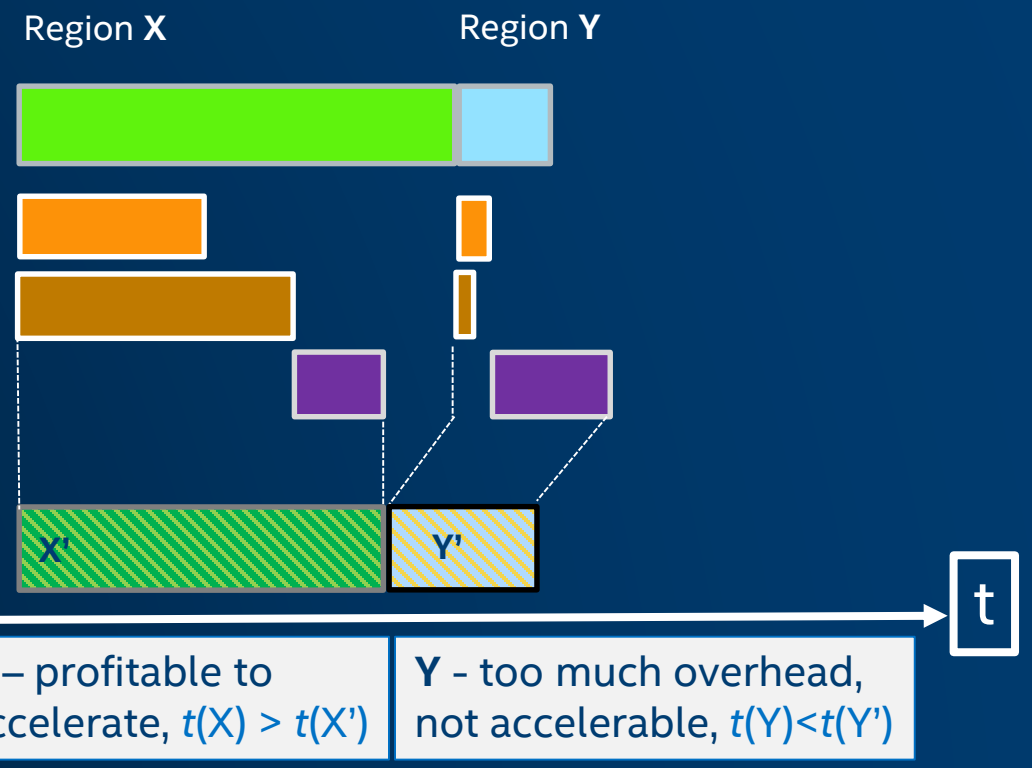
First order analytical modeling pillars:

- Compute throughput model
- Memory sub-system model
- Offload data transfer modeling

Execution time on baseline platform (CPU)

- Execution time on accelerator. Estimate assuming bound exclusively by Compute
- Execution time on accelerator. Estimate assuming bound exclusively by caches/memory
- Offload Tax estimate (data transfer + invoke)

Final estimated time on target platform (eg GPU)



X – profitable to accelerate, $t(X) > t(X')$

Y - too much overhead, not accelerable, $t(Y) < t(Y')$

$$t_{region} = \max(t_{compute}, t_{memory\ subsystem}) + t_{data\ transfer\ tax} + t_{kernel\ launch}$$

MODELING PERFORMANCE

We minimize the total time spent in this loop hierarchy by varying offload strategies U (offload/non-offload, #threads for each component $loop_i$ of loopnest)

$$\boldsymbol{Objective\ function: T_{all}} = \min_{U=\{uf_1, uf_2, ..\}} \left(\sum_i \boldsymbol{T_i} + t_{data\ transfer} + t_{invoke} + T_{cpu} \right)$$

Reject loopnests for which $\boldsymbol{T}$ (x86) / $\boldsymbol{T_{all}}$ (x86+"X") < 1.0

$$\boldsymbol{T_i} = max \begin{cases} T_i^{Comp\_only}(\ ) \\ T_i^{M_k\_only}\left(M_i^k\right) = \dfrac{\boldsymbol{M_i^k}}{\boldsymbol{BW_k}} \end{cases}$$

This is effectively "balance" (throughput) model

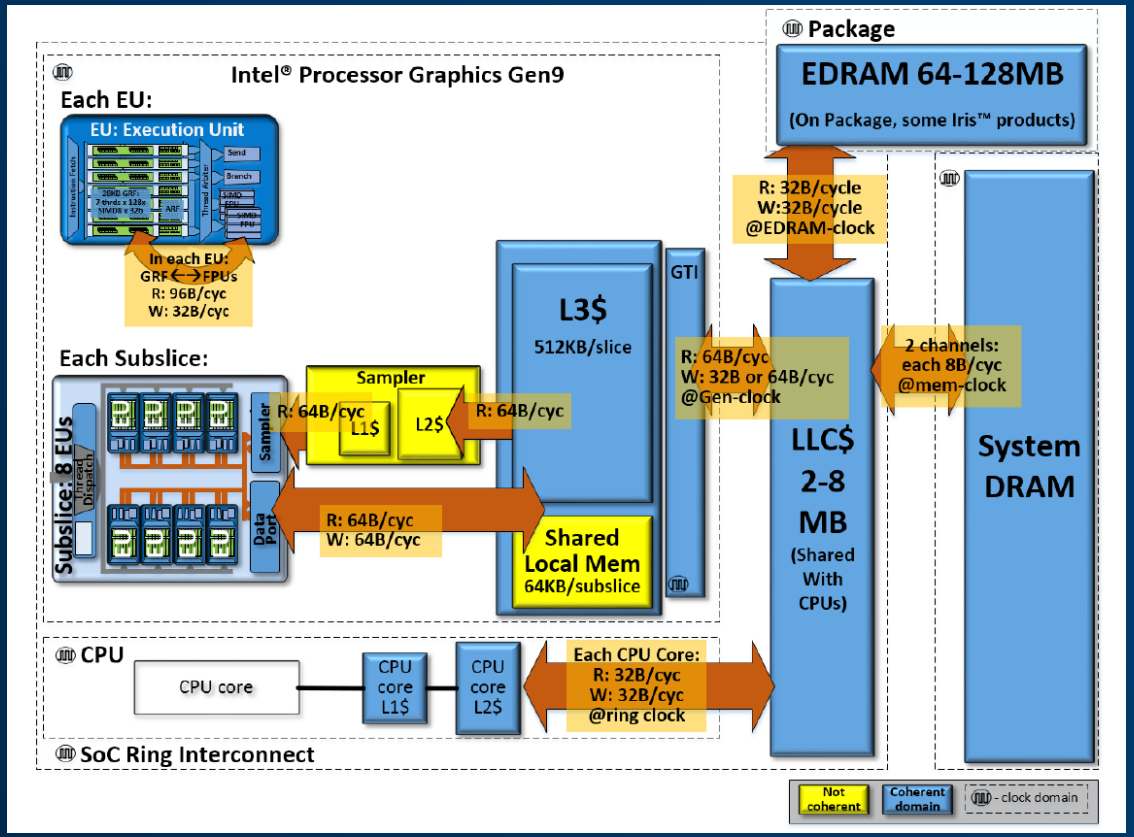
Under algorithmic constraints (Dependencies and TripCount/Granularity)

INTEL® ADVISOR

GPU ROOFLINE ANALYSIS

INTEL® GEN9 MEMORY HIERARCHY

- Intel® Graphics Compute Architecture uses the same DRAM with the CPU
- Level-3 (L3) data cache: slice-shared asset
- Shared Local Memory (SLM): a dedicated structure within the L3 that supports the work-group local memory address space
- Graphics Technology Interface (GTI): a dedicated interface unit connects the entire architecture interfaces to the rest of the SoC components
- The rest of SoC memory hierarchy includes the large Last-Level Cache (LLC, which is shared between CPU and GPU), possibly embedded DRAM and finally the system DRAM



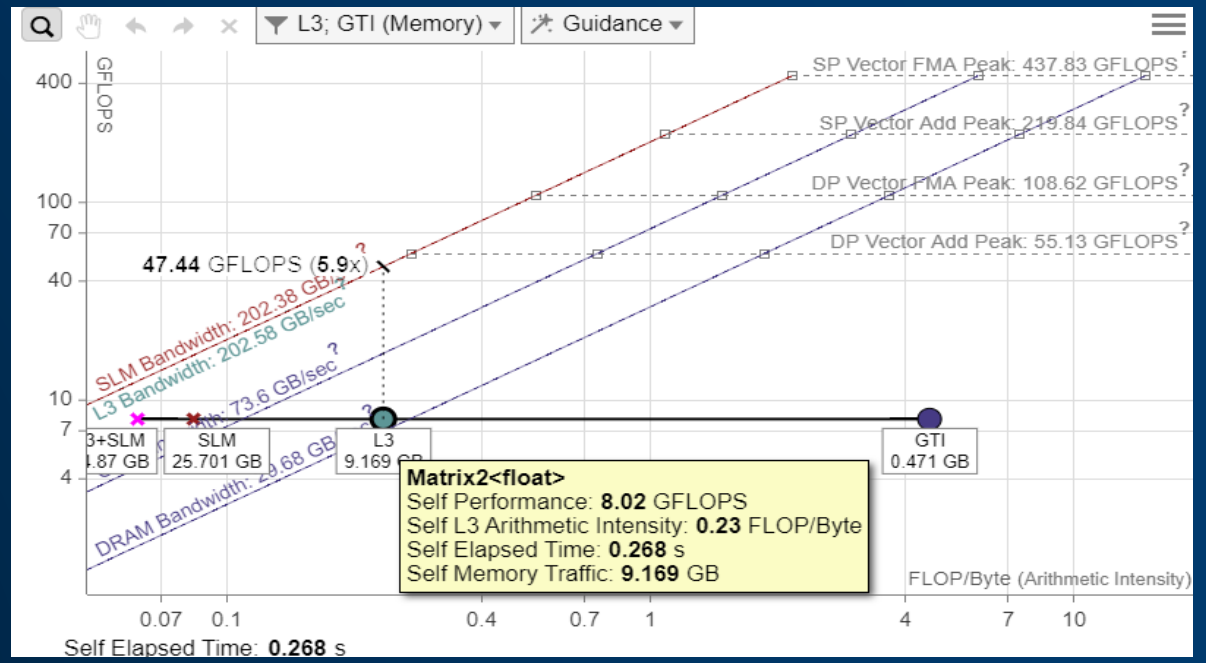
A view of the SoC chip level memory hierarchy and its theoretical peak bandwidths for the compute architecture of Intel processor graphics gen9

FIND EFFECTIVE OPTIMIZATION STRATEGIES

INTEL® ADVISOR - GPU ROOFLINE

GPU Roofline Performance Insights

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



RUNNING INTEL® ADVISOR GPU ROOFLINE ANALYSIS

The Roofline model on GPU is a technical preview feature and is not available by default. To enable it:

```
export ADVIXE_EXPERIMENTAL=gpu-profiling
```

To run the GPU Roofline analysis in the Intel® Advisor CLI:

Run the Survey analysis with the `--enable-gpu-profiling` option:

```
advixe-cl –collect=survey --enable-gpu-profiling --project-dir=<my_project_directory> --search-dir
     src:r=<my_source_directory> -- ./myapp [app_parameters]
```

Run the Trip Counts and FLOP analysis with `--enable-gpu-profiling` option:

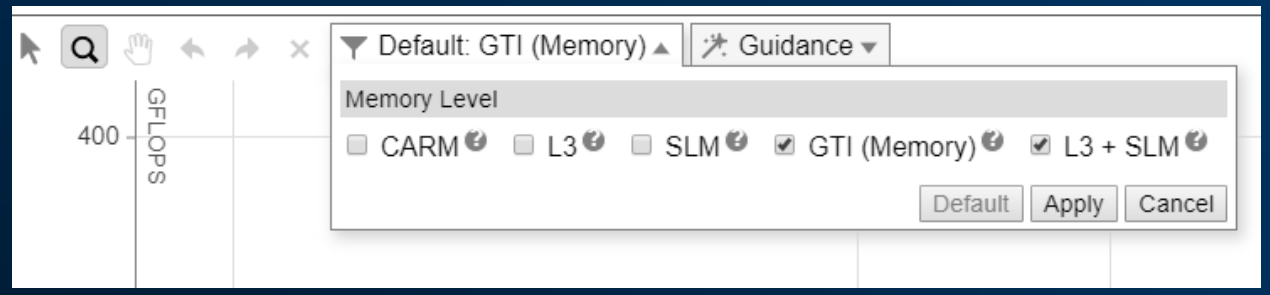
```
advixe-cl –collect=tripcounts --stacks --flop --enable-gpu-profiling --project-
dir=<my_project_directory> --search-dir  src:r=<my_source_directory> -- ./myapp [app_parameters]
```

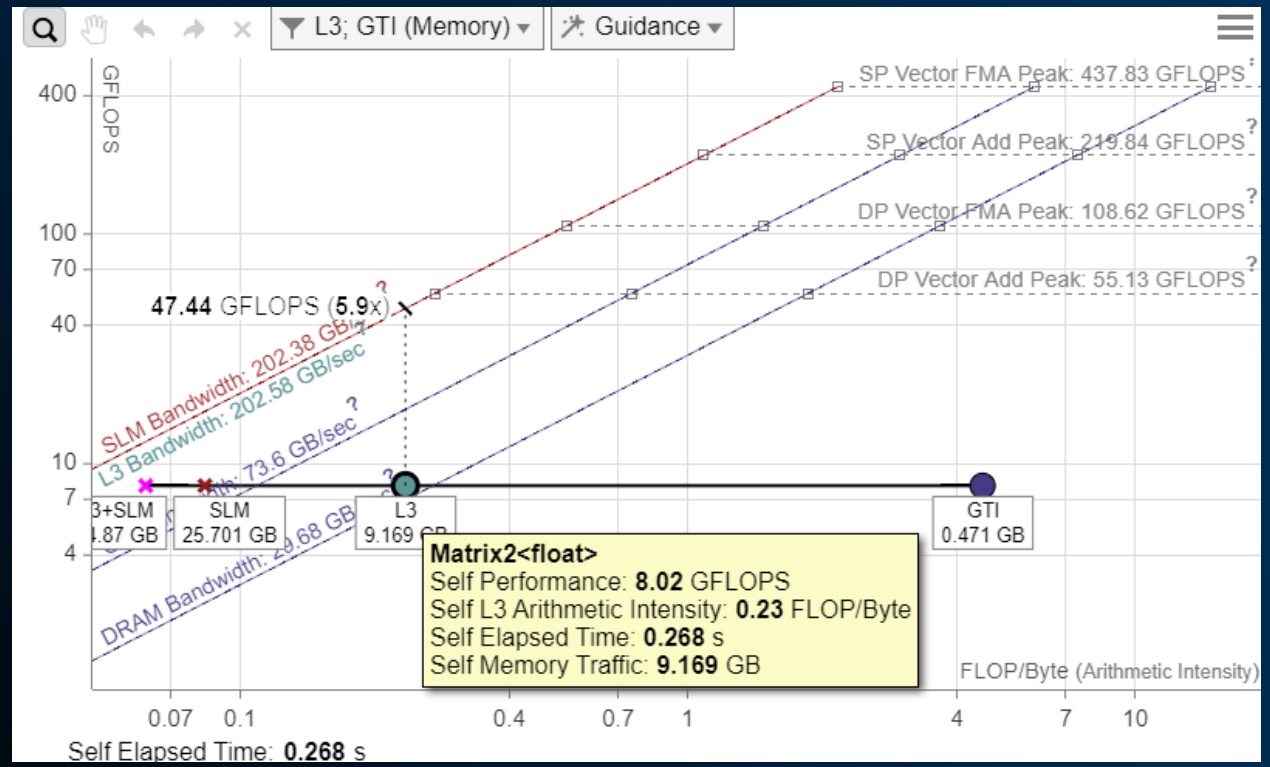
Generate a GPU Roofline report:

```
advixe-cl --report=roofline --gpu --project-dir=<my_project_directory> --report-output=roofline.html
```

Open the generated roofline.html in a web browser to visualize GPU performance.

ROOFLINE ANALYSIS ON INTEL® GPU





As an example you can see from the roofline chart, our L3 dot is very close to the L3 maximum bandwidth, to get more FLOPS we need to optimize our caches further. A cache blocking optimization strategy can make better use of memory and should increase our performance. The GTI (traffic between our GPU, GPU uncore (LLC) and main memory)is far from the GTI roofline, transfer costs between out CPU to GPU does not seem to be an issue.

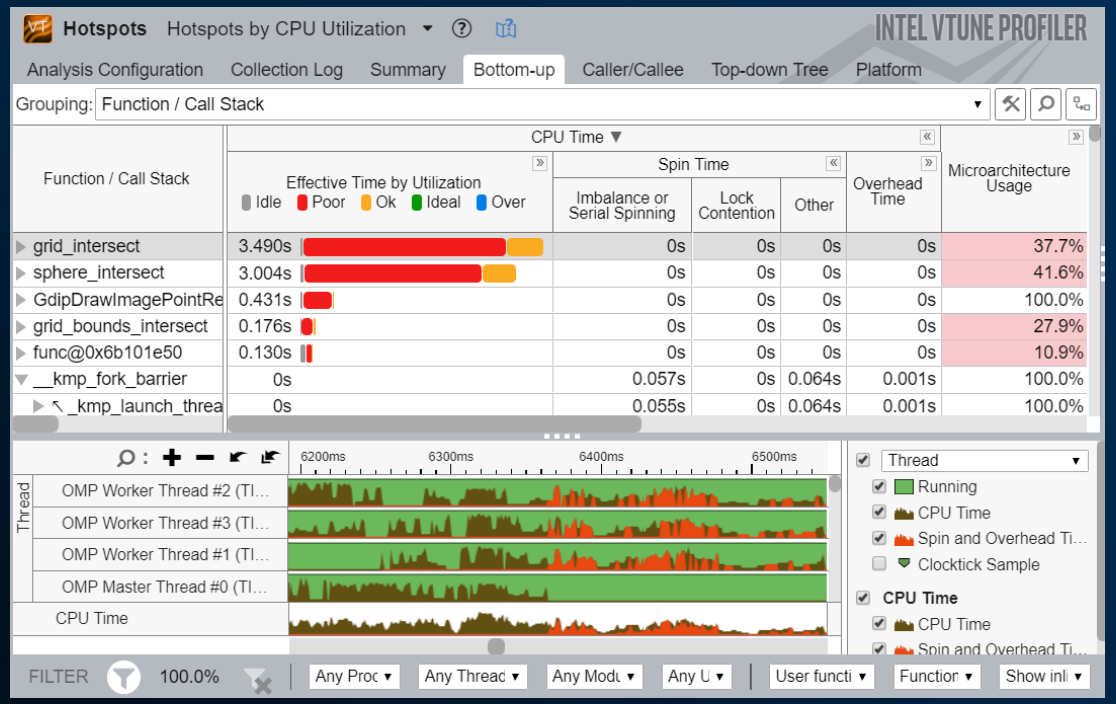
INTEL® VTUNE® PROFILER

GPU ANALYSIS

Analyze & Tune Application Performance

Intel® VTune™ Profiler — Performance Profiler

New more descriptive name!



Learn More: software.intel.com/vtune

Save Time Optimizing Code

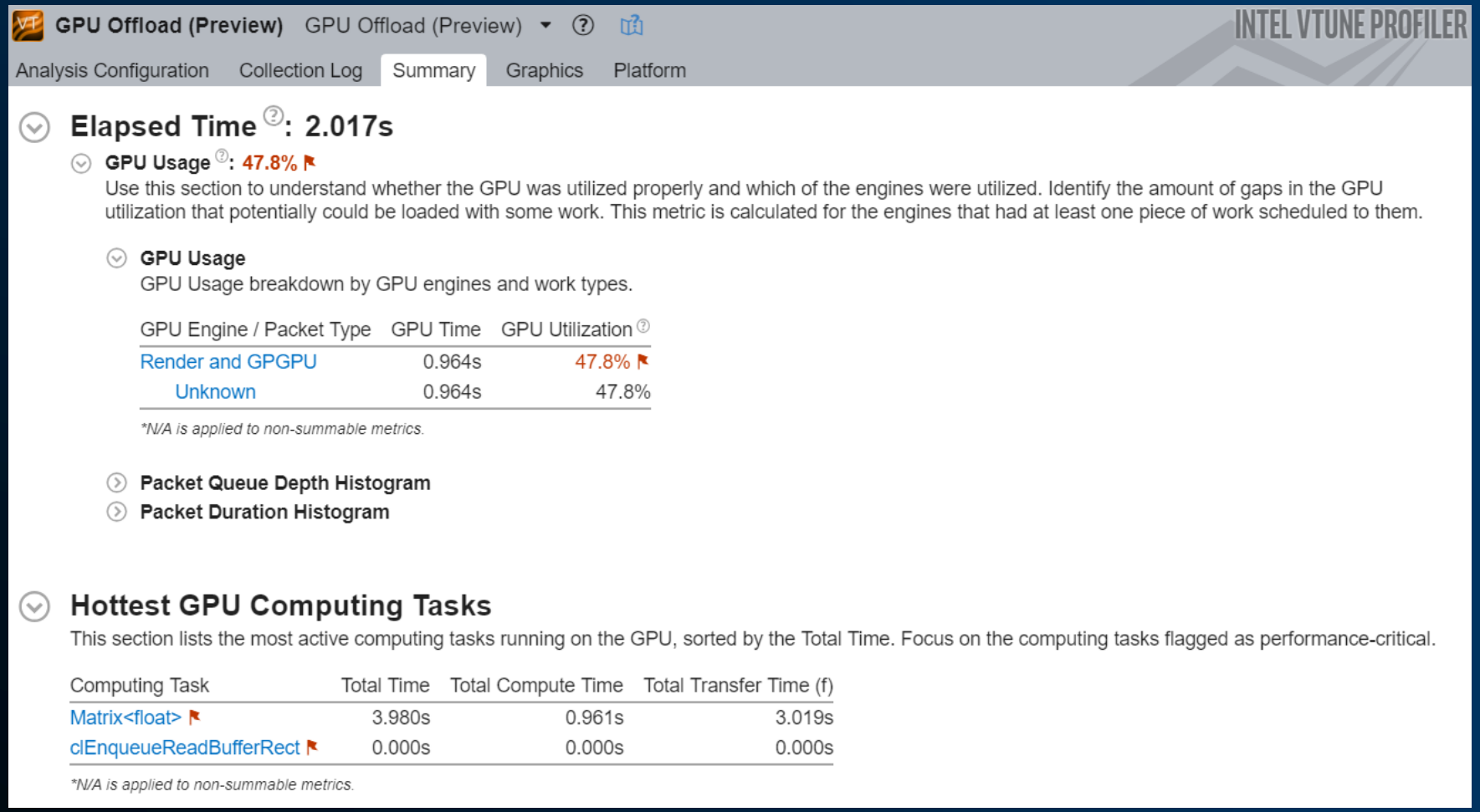
- Accurately profile C, C++, Fortran*, Python*, Go*, Java*, or any mix
- Optimize CPU, threading, memory, cache, storage & more
- Save time: rich analysis leads to insight
- Take advantage of Priority Support
  - Connects customers to Intel engineers for confidential inquiries (paid versions)

What's New in 2020 Release (partial list)

- Production release of Platform Profiler - Longer Data Collection
- Design & optimize for Intel® Optane™ DC Persistent Memory
- Application Performance Snapshot – adds communication pattern diagnosis, profiles more ranks
- Linux* – extensive additional use of Perf enables analysis without adding drivers

OPTIMIZE YOUR GPU USAGE USING INTEL® VTUNE PROFILER

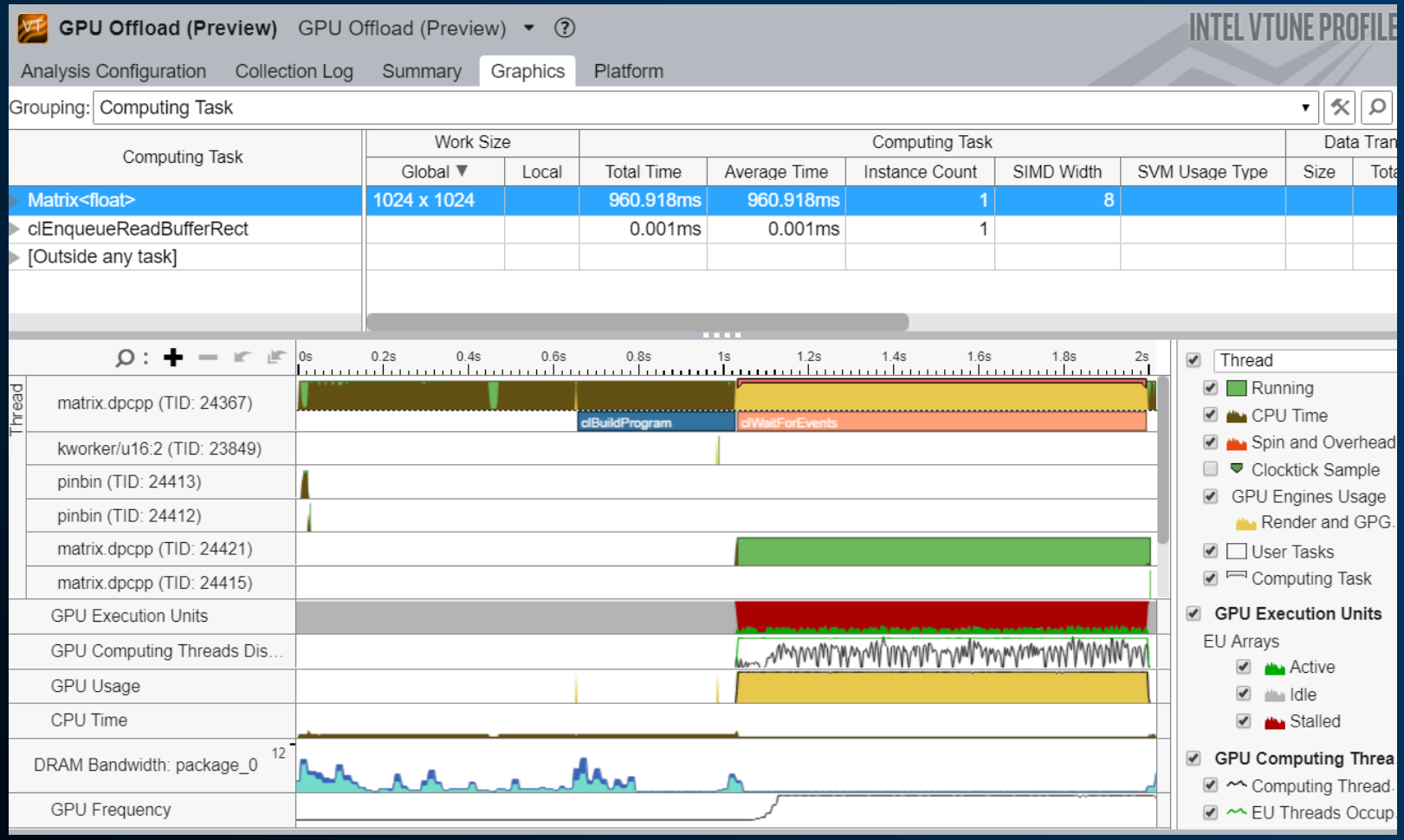
GPU OFFLOAD



This analysis enables you to:
Identify how effectively your application uses DPC++ or OpenCL kernels.
Analyze execution of Intel Media SDK tasks over time (for Linux targets only)
Explore GPU usage and analyze a software queue for GPU engines at each moment of time

OPTIMIZE YOUR GPU USAGE USING INTEL® VTUNE™ PROFILER

GPU OFFLOAD



Use the GPU offload features of Intel® VTune™ Profiler to see how effectively we are using our GPU.

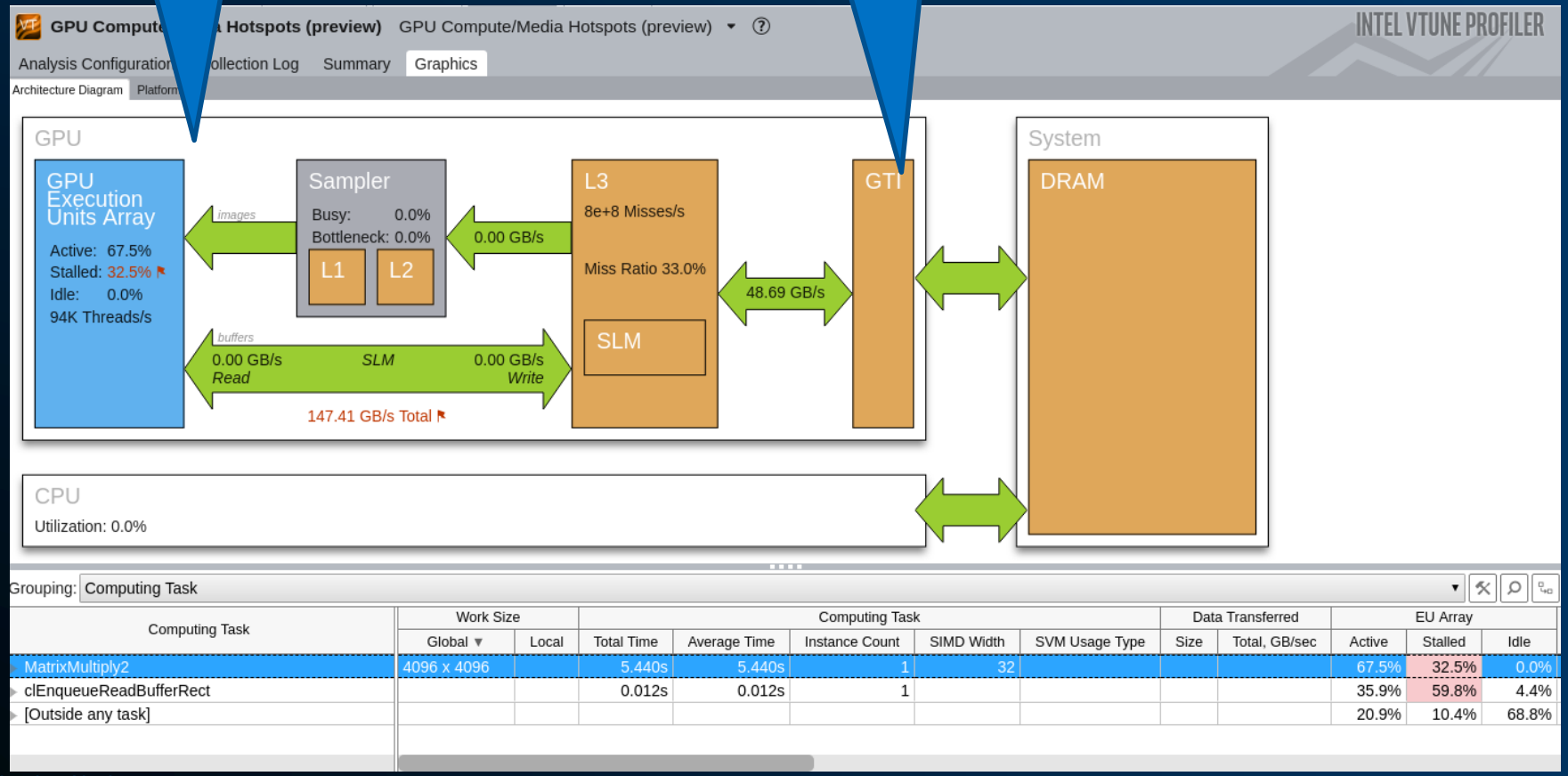
VTune Profiler shows a synchronized timeline between the CPU and GPU. GPU offload does indicate that our GPU execution units are stalling as indicated by the dark read bar in our timeline.

OPTIMIZE YOUR GPU USAGE USING INTEL® VTUNE PROFILER

GPU HOTSPOTS

Active vs. Idle EU activity

Bandwidth at multiple levels



Grouping: Computing Task

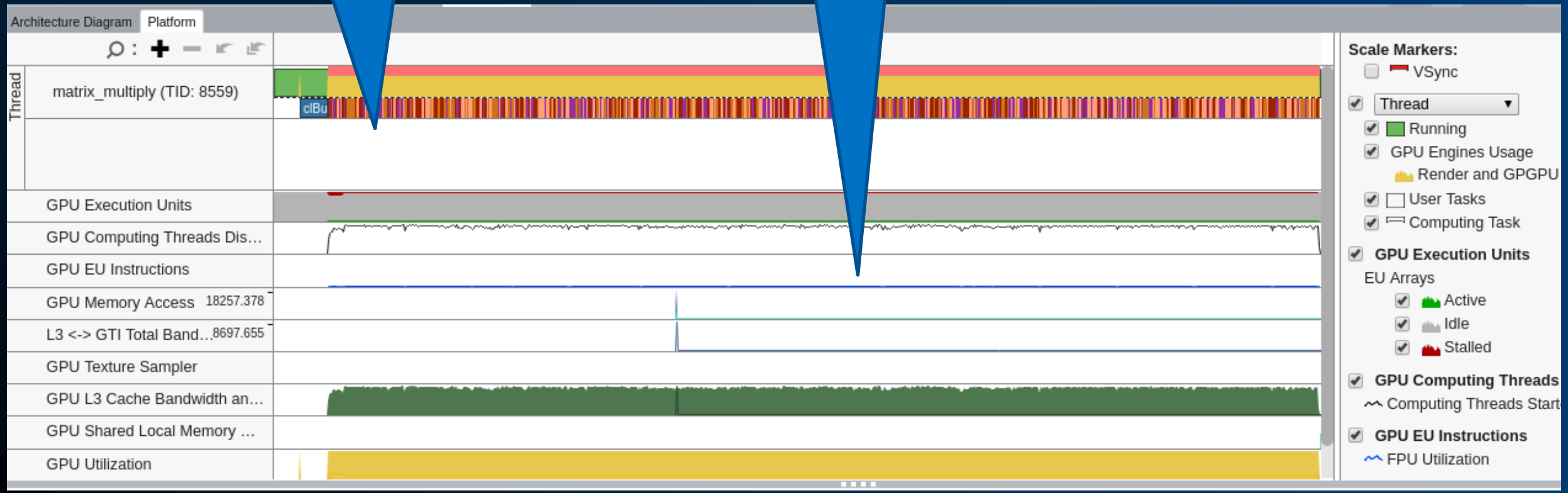
Computing Task	Work Size		Computing Task					Data Transferred		EU Array		
	Global ▼	Local	Total Time	Average Time	Instance Count	SIMD Width	SVM Usage Type	Size	Total, GB/sec	Active	Stalled	Idle
MatrixMultiply2	4096 x 4096		5.440s	5.440s	1	32				67.5%	32.5%	0.0%
clEnqueueReadBufferRect			0.012s	0.012s	1					35.9%	59.8%	4.4%
[Outside any task]										20.9%	10.4%	68.8%

Run VTune Profiler GPU Hotspots to try to identify the source of our low GPU utilization and stalls. Click on the graphics tab in GPU Hotspots and you can see a high-level diagram of your architecture.

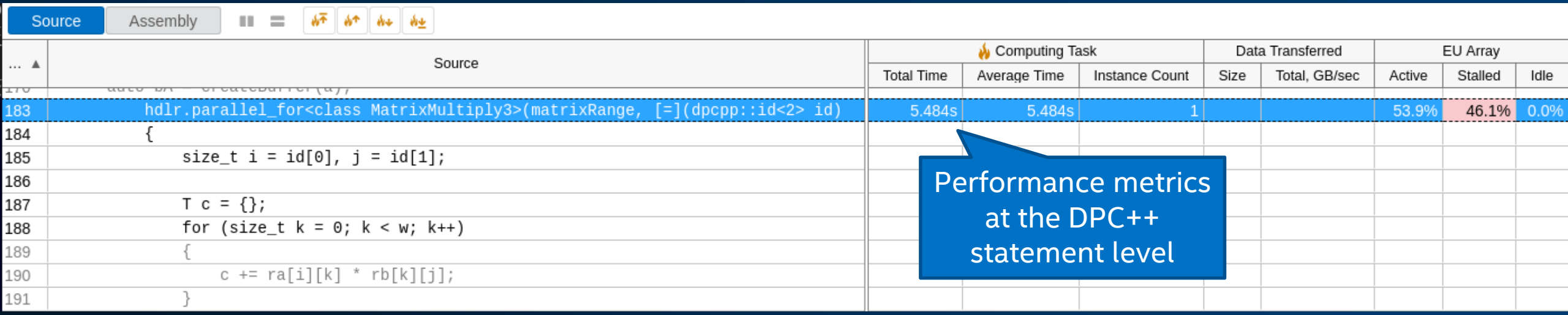
Intel ® VTune™ Profiler for Intel GPUs – Timelines for Correlation

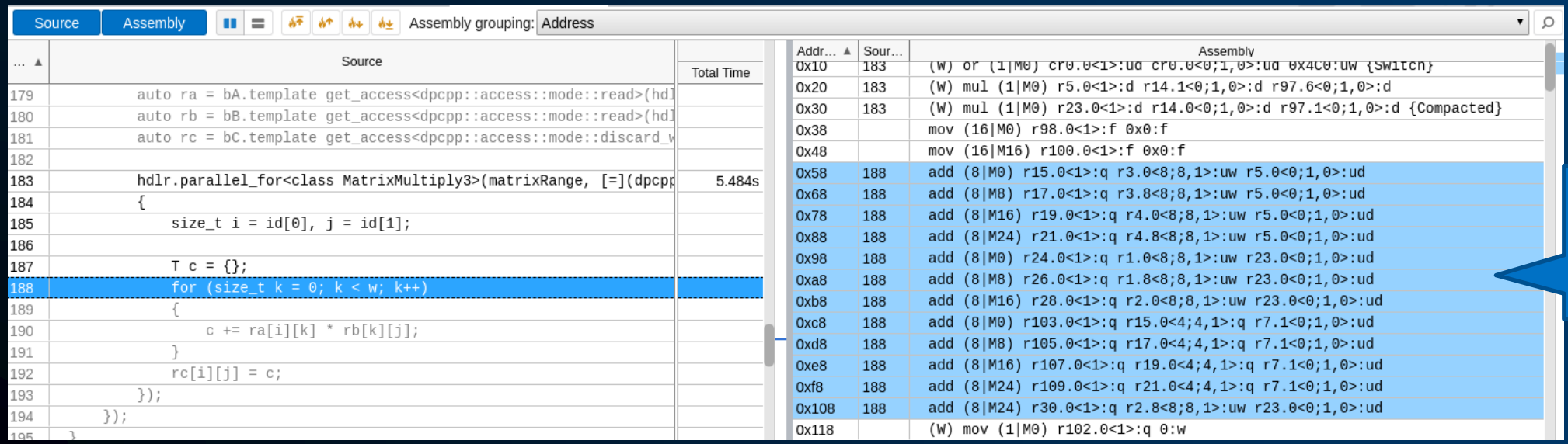
Identify too much or too little kernel activity

Correlate GPU activity with kernels and threads



Intel® VTune™ Profiler for DPC++ Code





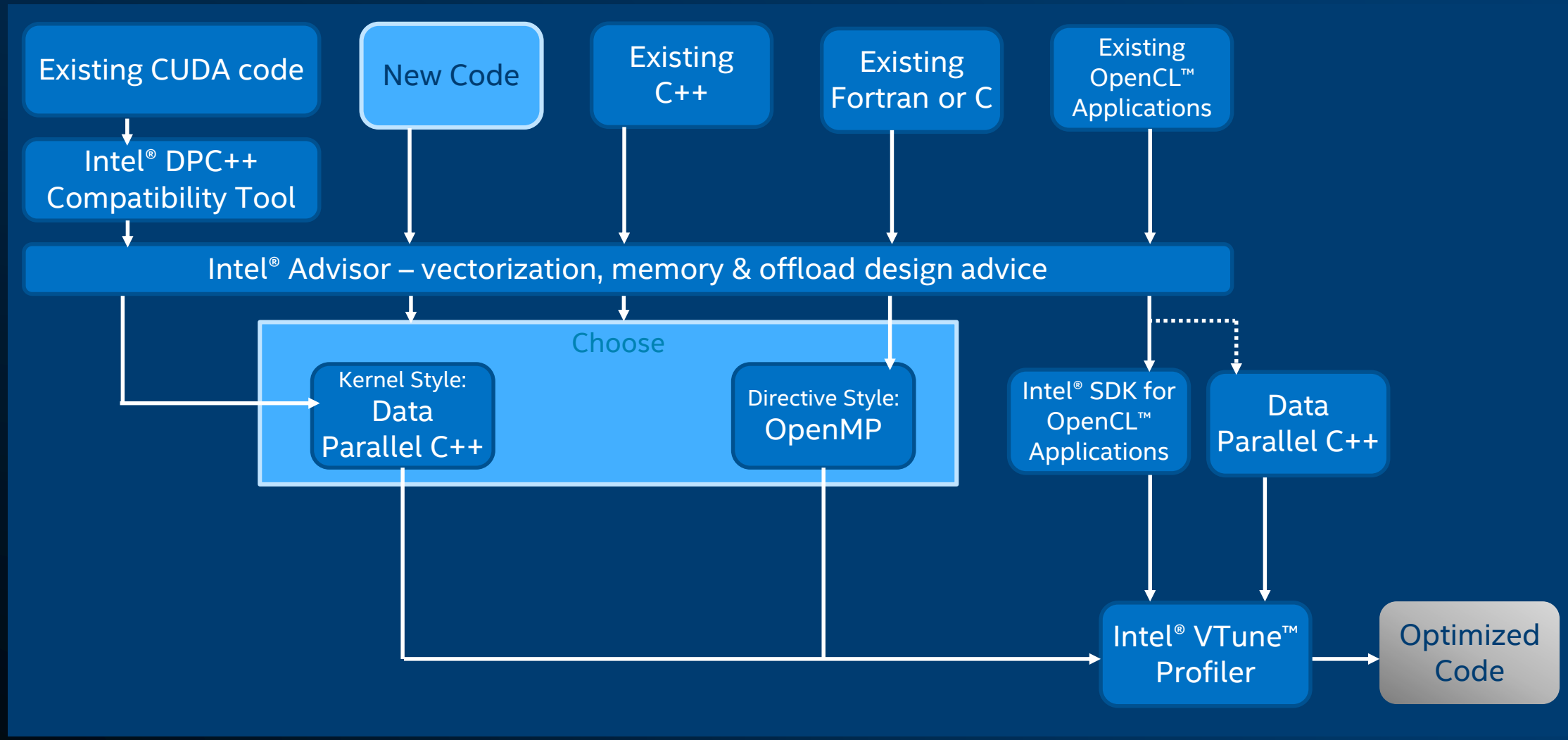
CASE STUDY/DEMO

PUTTING IT ALL TOGETHER

GPU ANALYSIS

HPC ONEAPI SINGLE NODE WORKFLOW

I WANT TO ACCELERATE USING DIRECT PROGRAMMING ON A GPU…



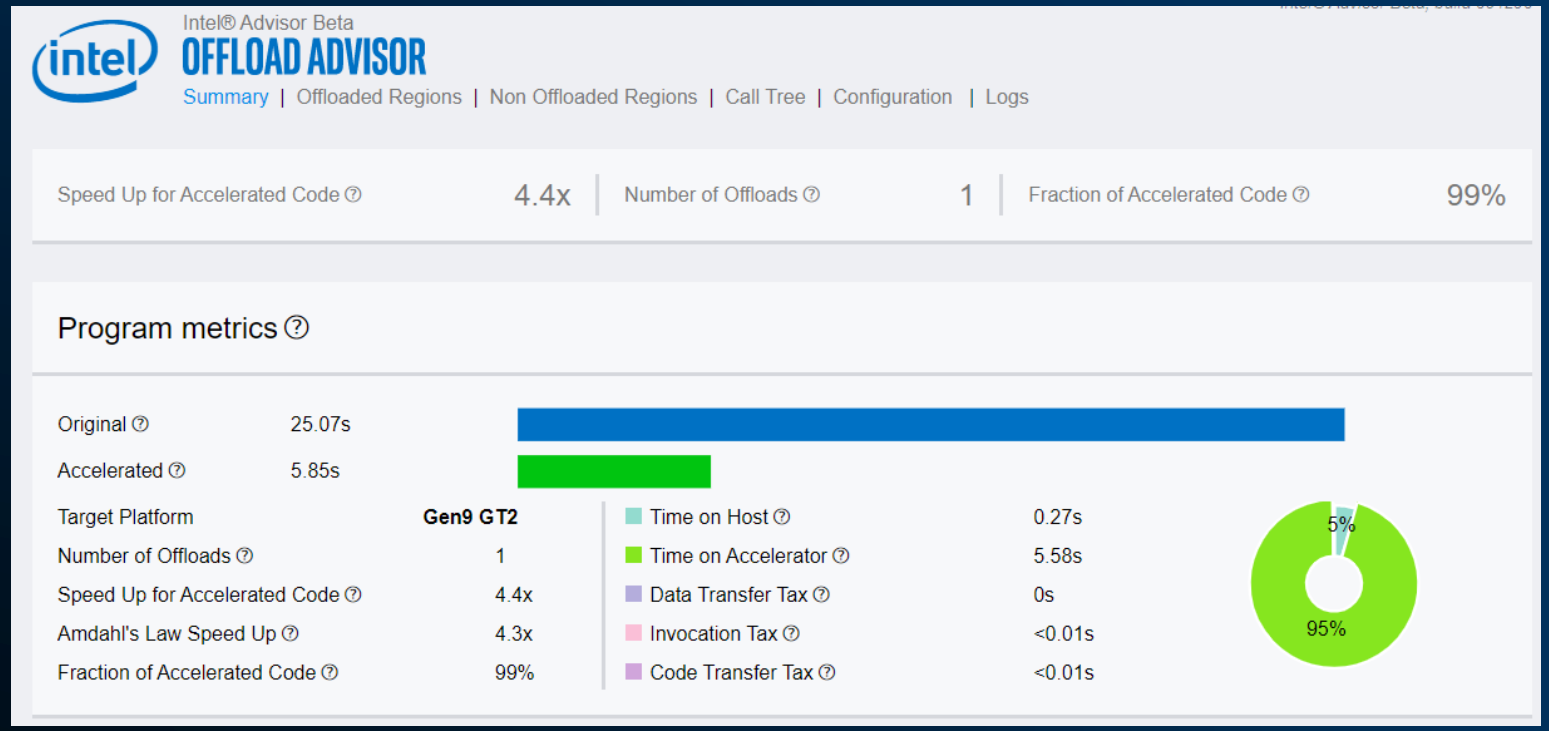
Purpose: Demo performance analysis tools for Intel® oneAPI

Steps:

1. Run Intel® Advisor - offload advisor on your CPU. Generate a report that indicates what should be offloaded to your accelerator.
2. After porting code to DPC++. Run Intel® VTune™ Profiler offload analysis and GPU hotspots. Show VTune Profiler results indicating places for improvement.
3. Re-Run a second VTune Profiler gpu-hotspot analysis on our optimized gpu kernel.
4. Run Intel Advisor GPU Roofline on our GPU workload to see where we can improve based on system maximums(roofs).

STEP 1 - RUN INTEL® ADVISOR - OFFLOAD ADVISOR

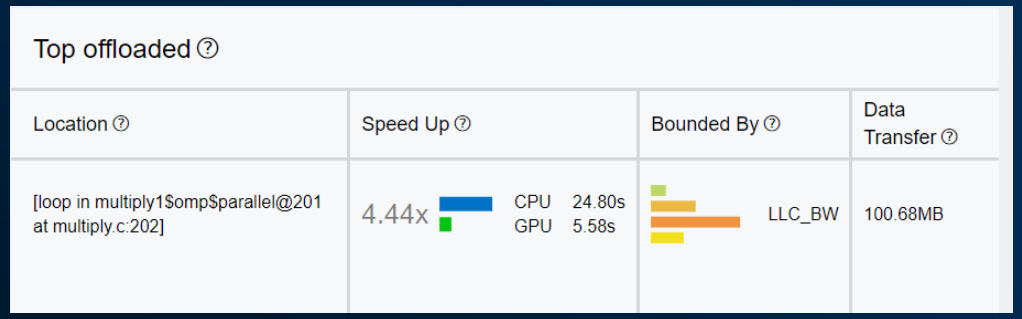
TO FIND CODE THAT CAN BE PROFITABLY OFFLOADED



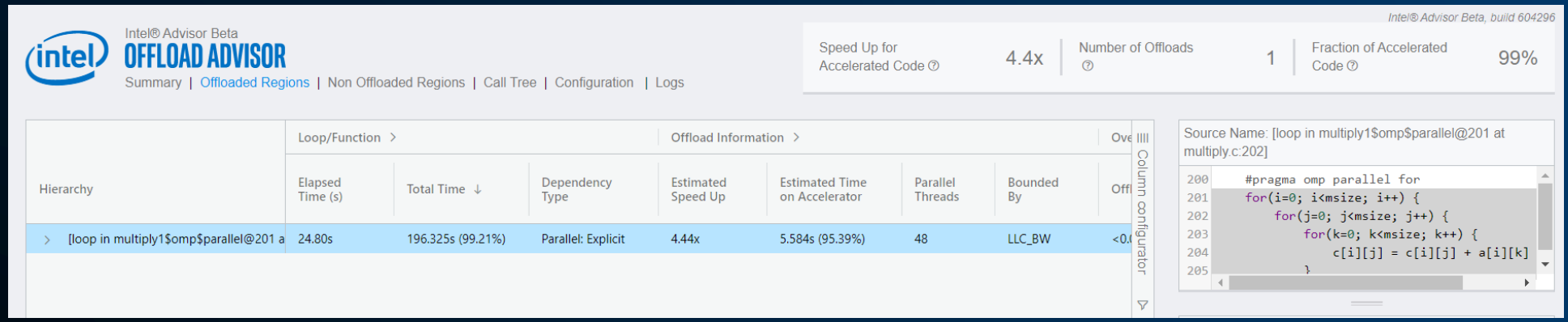
Some key observations
The workload was accelerated 4.4x
You can see in program metrics that the original workload ran in 25.07s and the accelerated workload ran in 5.85s

STEP 1 - RUN INTEL® ADVISOR - OFFLOAD ADVISOR

TO FIND CODE THAT CAN BE PROFITABLY OFFLOADED TO YOUR ACCELERATOR

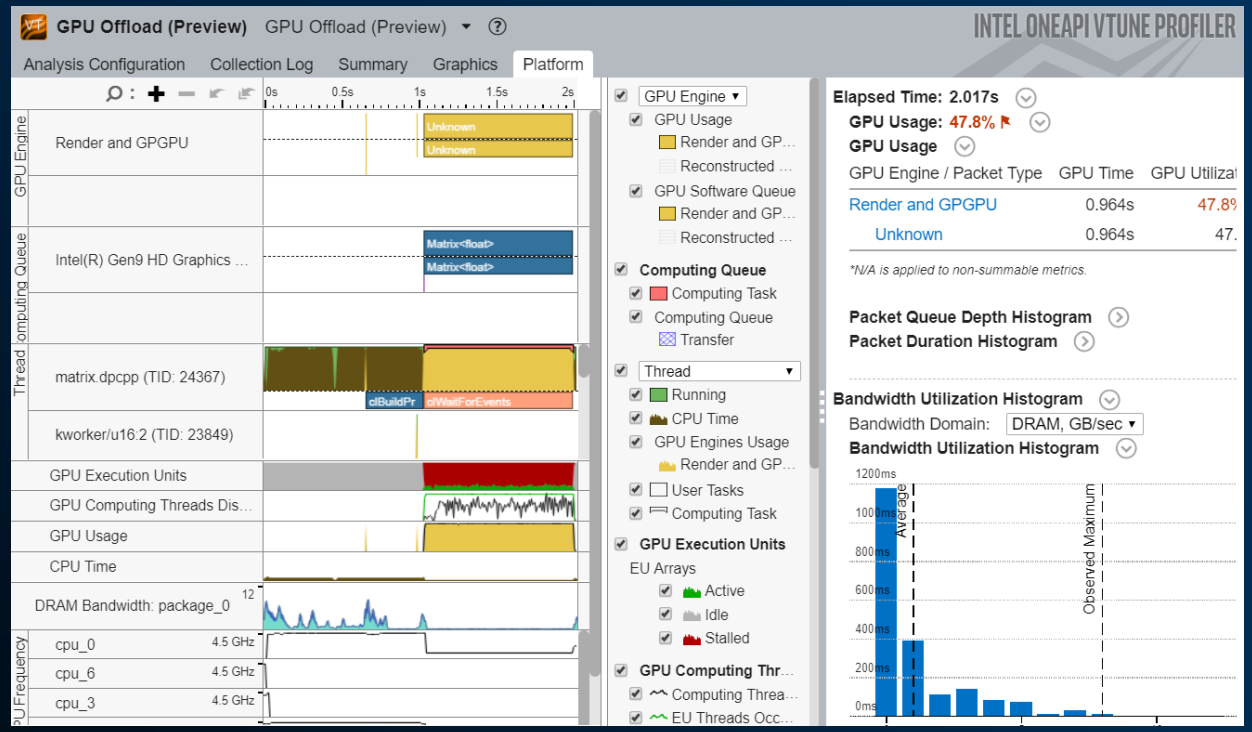


Click on the offloaded regions link.
Additional details are displayed including:
Specific line of code to be offloaded with the source.
Offload information
Data transfer information including the amount of data transferred from CPU->GPU. We can also see we are bounded by LLC_BW



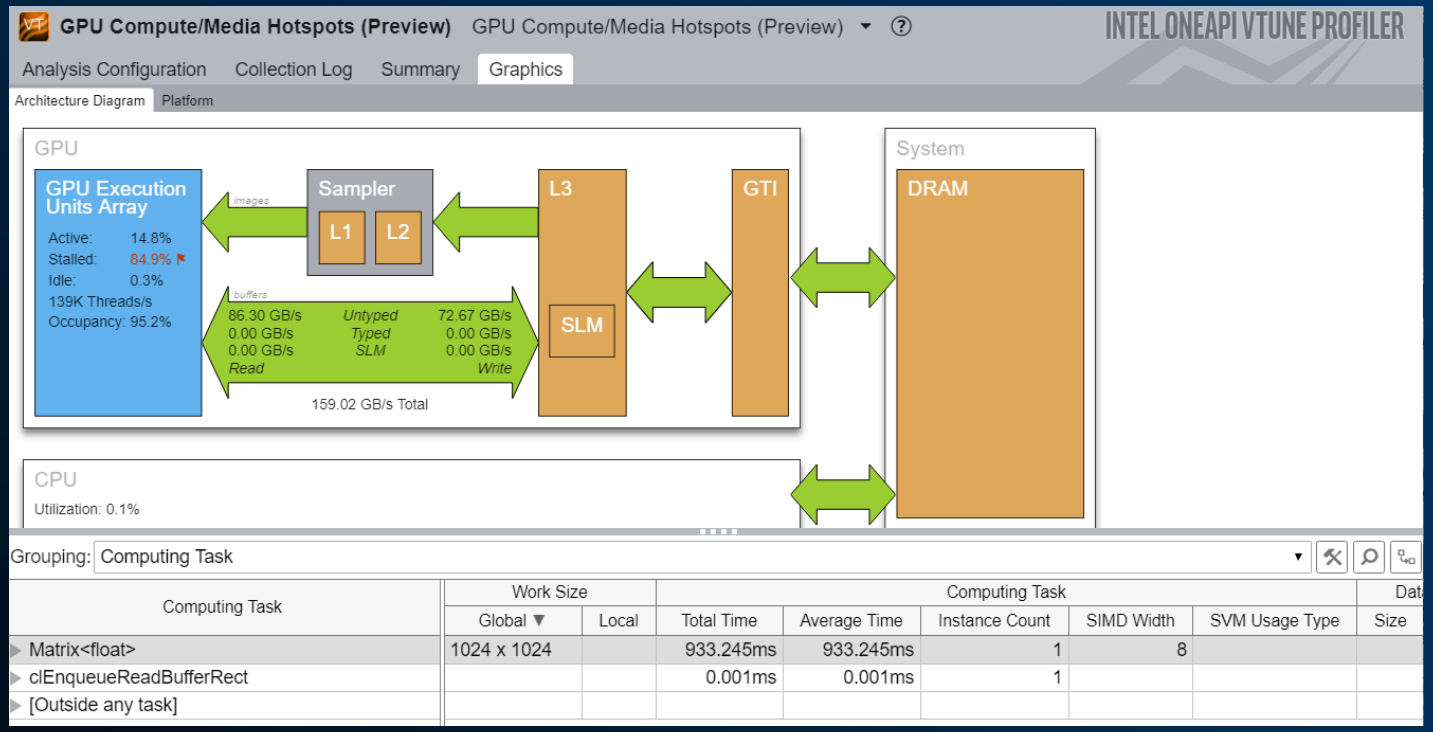
STEP 2: RUN INTEL® VTUNE™ PROFILER

TO OPTIMIZE YOUR USE OF GPU (OFFLOAD)



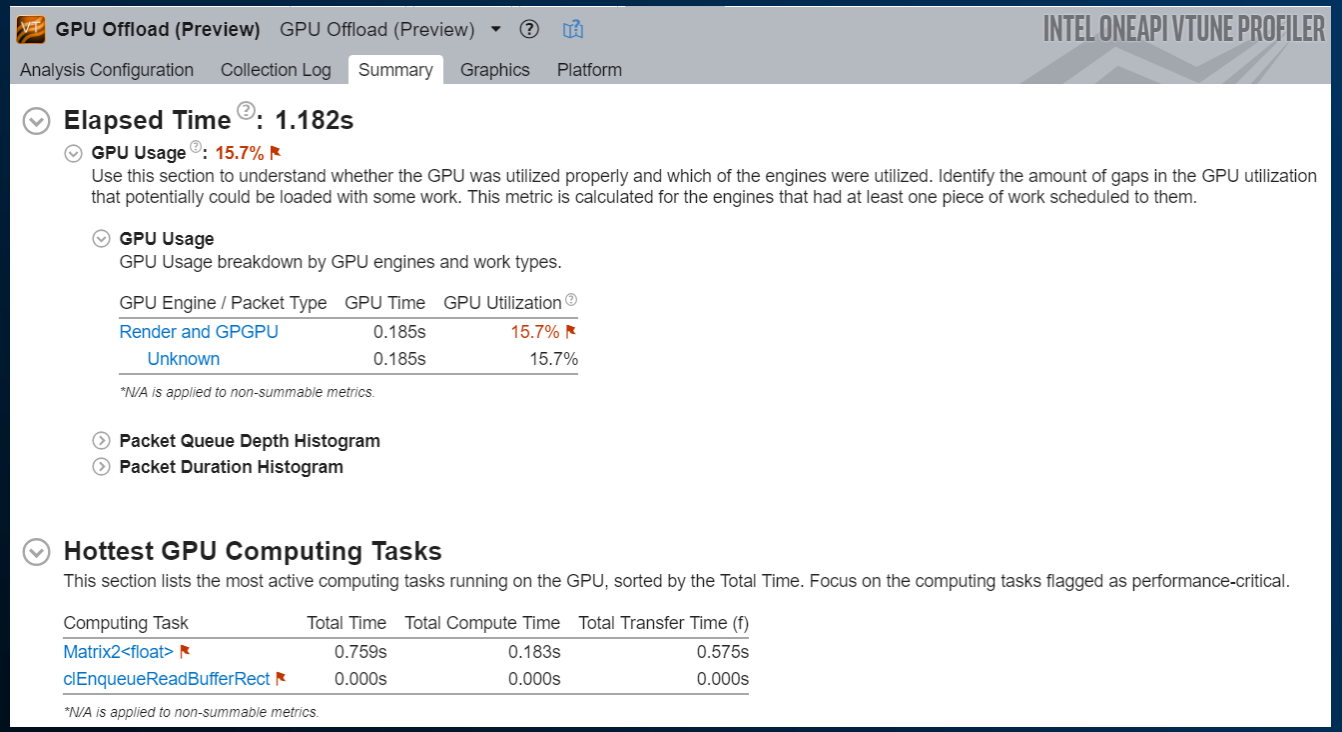
STEP 2: RUN INTEL® VTUNE™ PROFILER

TO OPTIMIZE YOUR USE OF GPU (GPU HOTSPOTS)



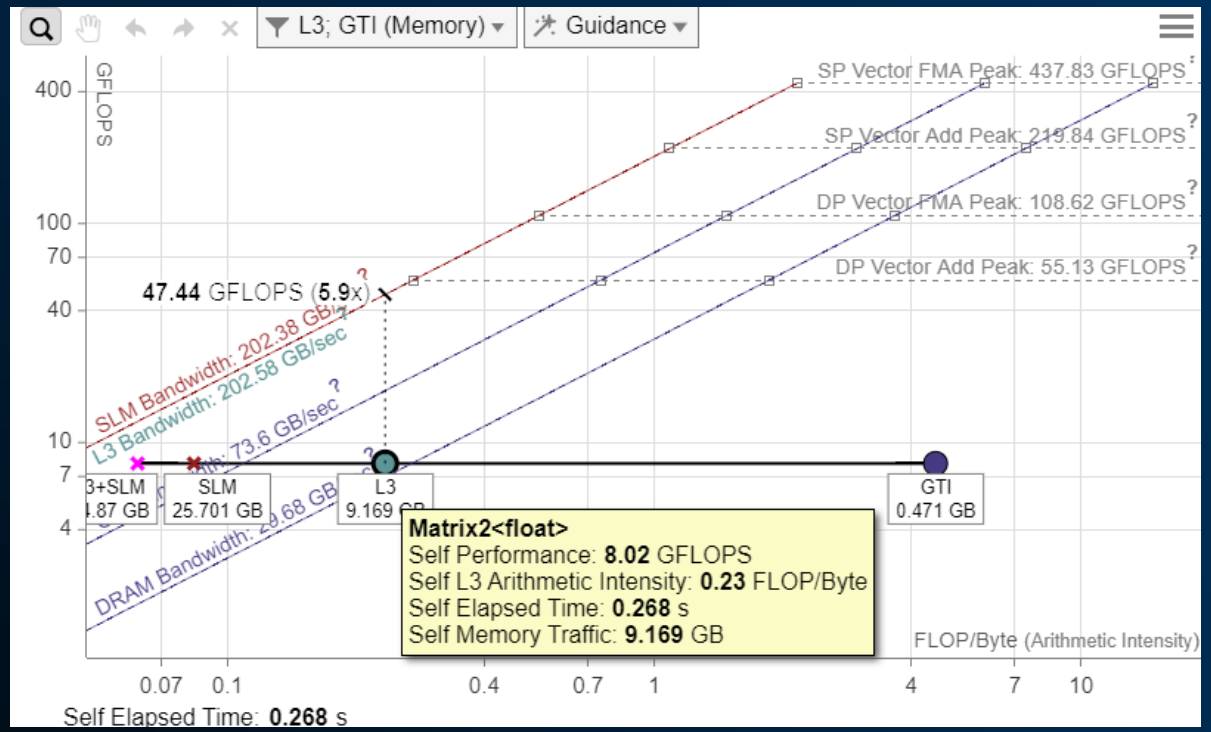
STEP 3: RERUN INTEL® VTUNE™ PROFILER

AFTER MAKING AN OPTIMIZATION, RE-RUN GPU-HOTSPOTS TO SEE WHAT HAS IMPROVED



STEP 4: RUN INTEL® ADVISOR GPU ROOFLINE

TO SEE HOW CLOSE YOU ARE TO THE SYSTEM MAXIMUMS (ROOFLINES)



SUMMARY

Intel® oneAPI includes some powerful analysis tools!

- Intel® Advisor and Intel® VTune™ Profiler now have features for analyzing your GPU.
- Intel® Advisor and Intel® VTune™ Profiler provide an effective workflow for optimizing your workloads for running on Intel® Accelerators.
- The workflow presented here shows how the tools complement each other.

NOTICES AND DISCLAIMERS

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

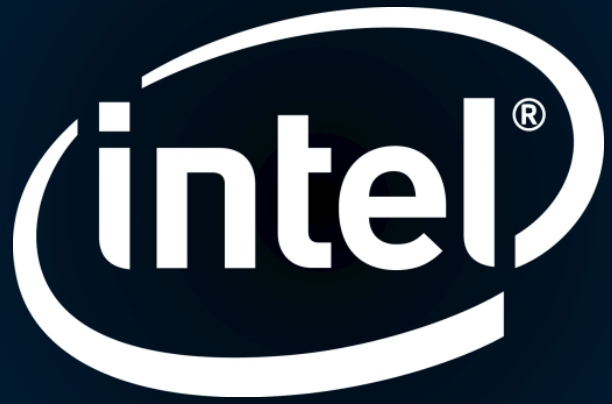
Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Intel, the Intel logo is trademark of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

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

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
Notice Revision #20110804


intel®