LRZ oneAPI Workshop, 8–10 November 2022

Intel® Advisor

Offload Modelling and Analysis

Klaus-Dieter Oertel

Agenda

- Advisor Overview
- Offload Modelling
  - Overview
  - GPU-to-GPU
- Roofline Analysis
  - Recap
  - GPU Code Analysis

Intel® Advisor for High Performance Code Design

Rich Set of Capabilities

Offload Modelling

Design offload strategy and model performance on GPU.

Roofline Analysis

Optimize your application for memory and compute.

Vectorization Optimization

Enable more vector parallelism and improve its efficiency.

Thread Prototyping

Model, tune, and test multiple threading designs.

Build Heterogeneous Algorithms

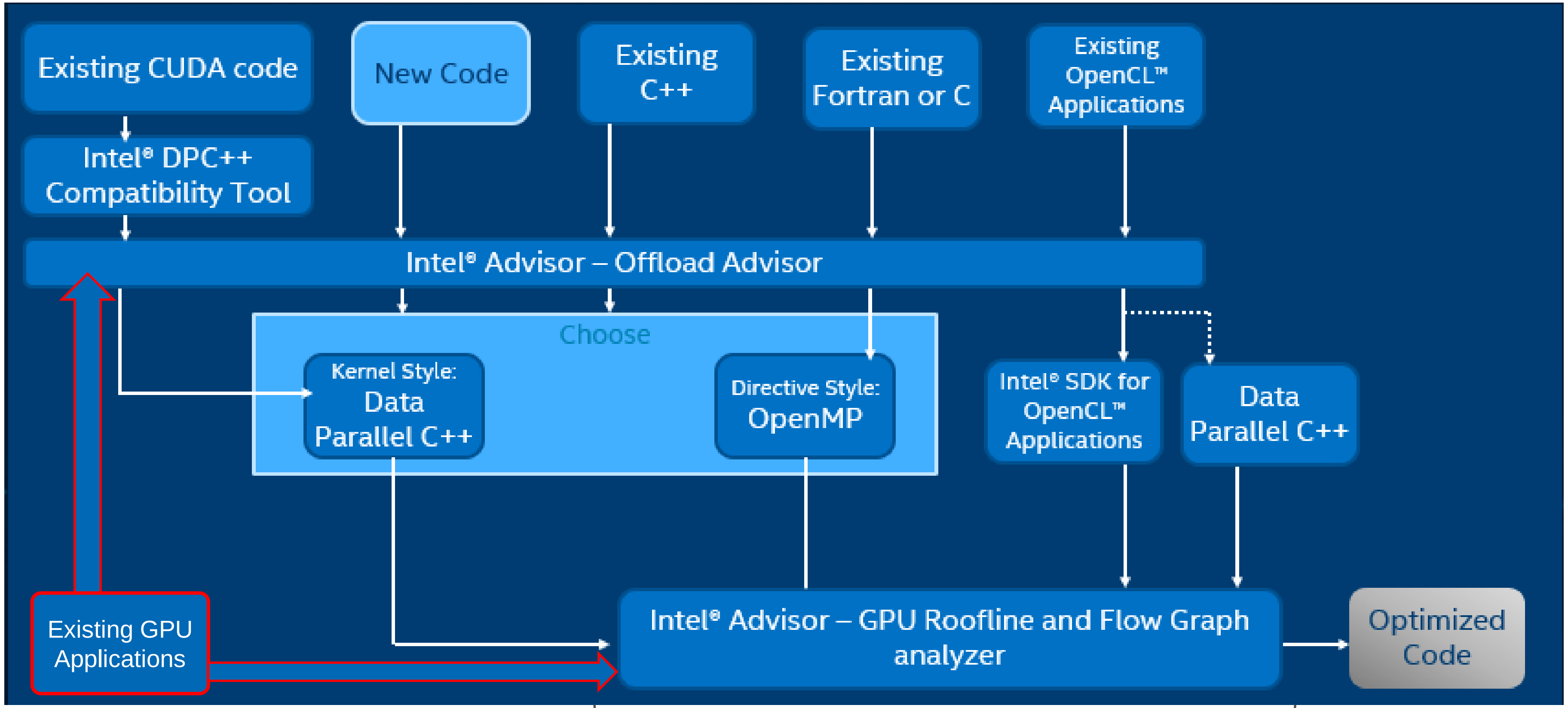
Create and analyze data flow and dependency computation graphs.

Learn More: software.intel.com/advisor

Using Intel® Advisor to increase performance

CPU-to-GPU model
Find kernels to offload and expected ROI from offloading

GPU-to-GPU model
- Estimate GPU kernel performance on new HW
- What-if analysis: model performance with changed software parameters



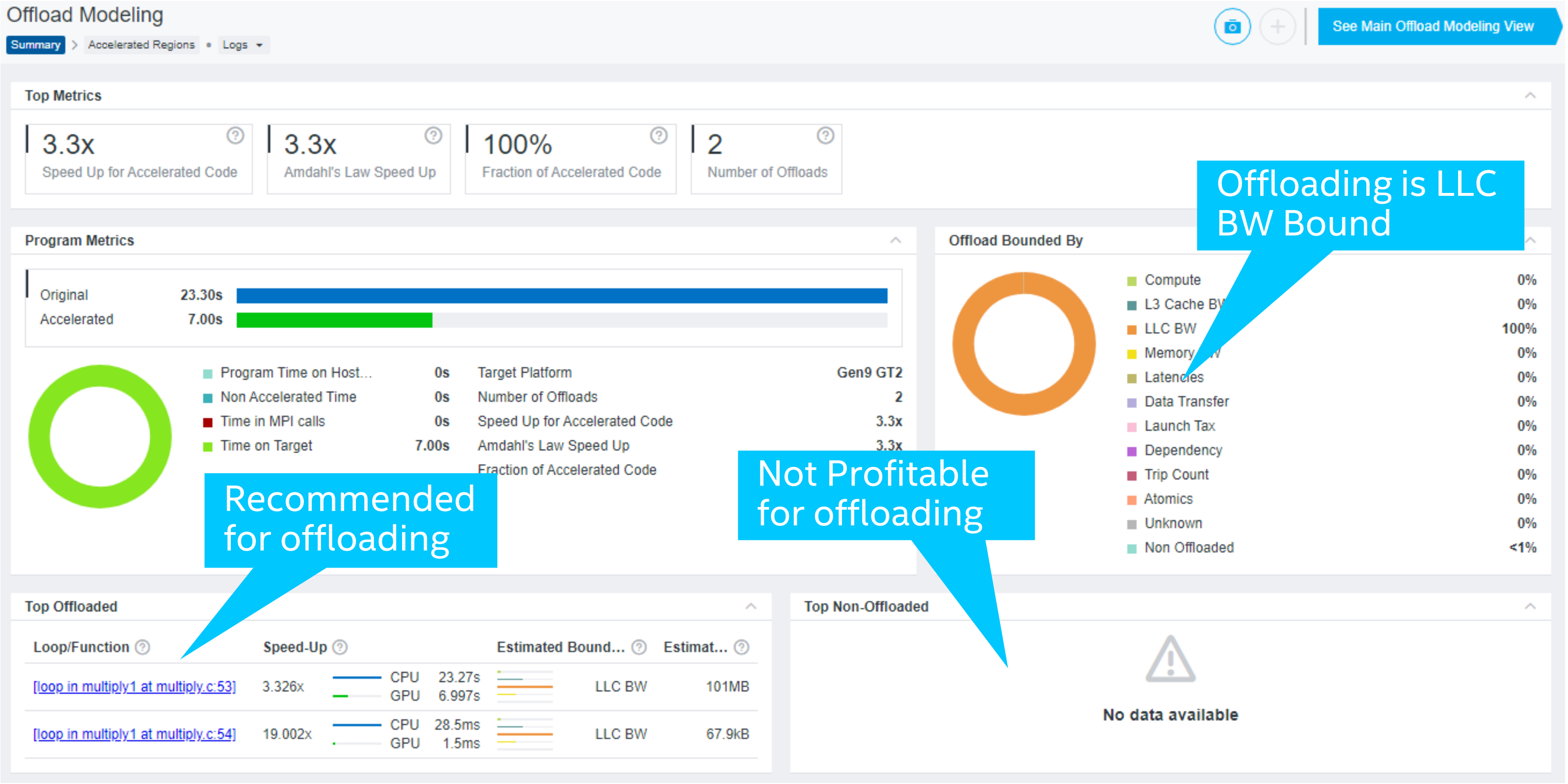
GPU Roofline: Identify performance bottleneck

Offload Modelling

Intel® Advisor - Offload Modeling

Find code that can be profitably offloaded

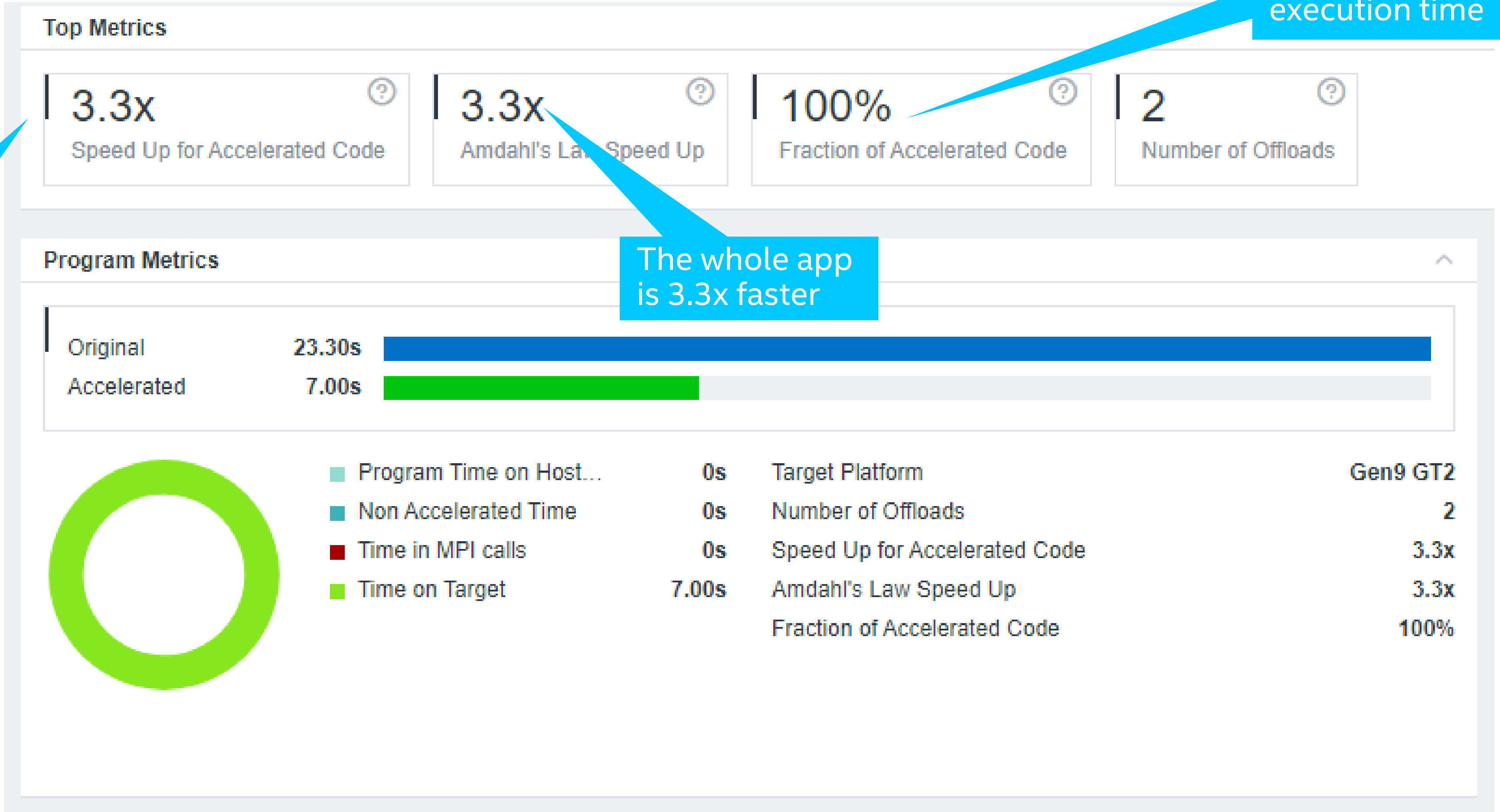
- Run on CPU or GPU – Predict for GPU
- Define which sections of the code should run on given accelerator
- Get performance projection on GPU



Intel® Advisor - Offload Modeling

What can be expected?

Loop takes 100% of the whole app execution time



Loop on GPU is 3.3x faster than on CPU

The whole app is 3.3x faster

Modeling Performance

Using Intel® Advisor – Offload Advisor

	Baseline HW	(Programming model)		Target HW		
1.a	**CPU** measured	(C,C++,Fortran, Py)	➡	**CPU** measured	+	**GPU** estimated
1.b	**CPU** measured	(DPC++, OCL, OMP, "target=host")	➡	**CPU** measured	+	**GPU** estimated
2	**CPU+GPU** measured	(DPC++, OCL, OMP, "target=offload")	➡	**CPU** measured	+	**GPU** Estimated

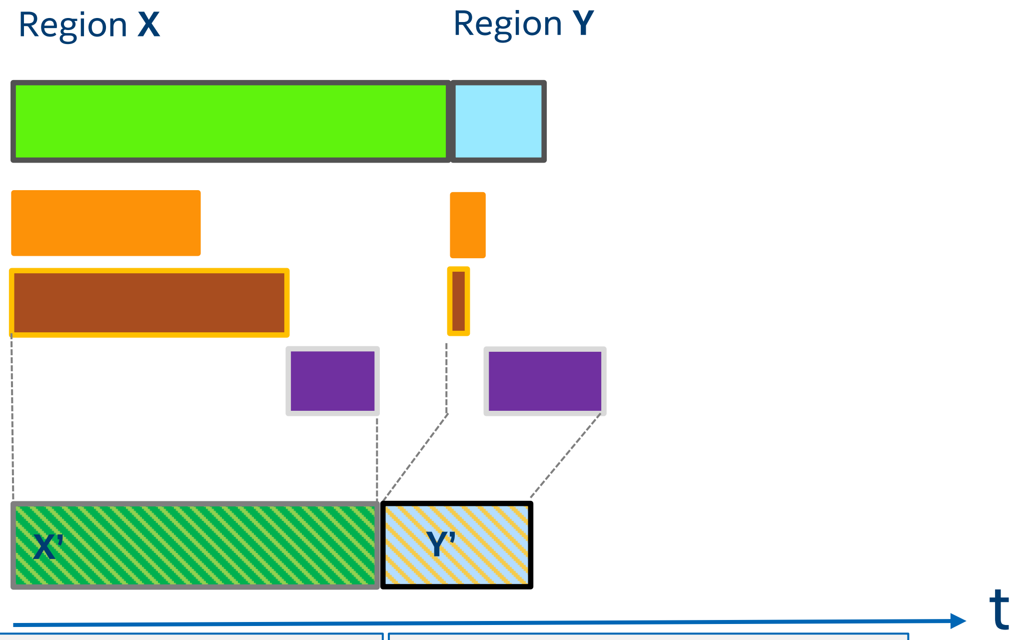
Modeling Performance

Using Intel® Advisor – Offload Advisor

Execution time on baseline platform (CPU)

- Execution time on accelerator. Estimate assuming bounded exclusively by **Compute**
- Execution time on accelerator. Estimate assuming bounded exclusively by **Caches/Memory**
- **Offload Tax** estimate (data transfer + invoke)

Final estimated time on target device (GPU)



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

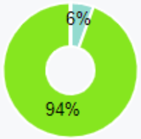
Y - too much overhead, not accelerable, $t(Y) < t(Y')$

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

Will Offload Increase Performance?

Program metrics

Original	13.84s		
Accelerated	1.63s		
Target Platform	Gen9 GT2	Time on Host	0.09s
Number of Offloads	1	Time on Accelerator	1.54s
Speed Up for Accelerated Code	8.9x	Data Transfer Tax	0s
Amdahl's Law Speed Up	8.5x	Invocation Tax	<0.01s
Fraction of Accelerated Code	99%	Code Transfer Tax	<0.01s



Top offloaded

Location	Speed Up		Bounded By	Data Transfer
[loop in iso_3dfdompparallel@52 at iso-3dfd_parallel.cc:53]	8.94x	CPU 13.75s GPU 1.54s	LLC_BW	<0.01MB

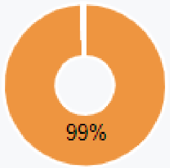
Good Candidates to offload

What is workload bounded by

Bad Candidates

Offloads bounded by

Compute	0%
L3 Cache BW	0%
LLC BW	99%
Memory BW	0%
Data Transfer	0%
Dependency	0%
Trip Count	0%
Unknown	0%
Non Offloaded	<1%



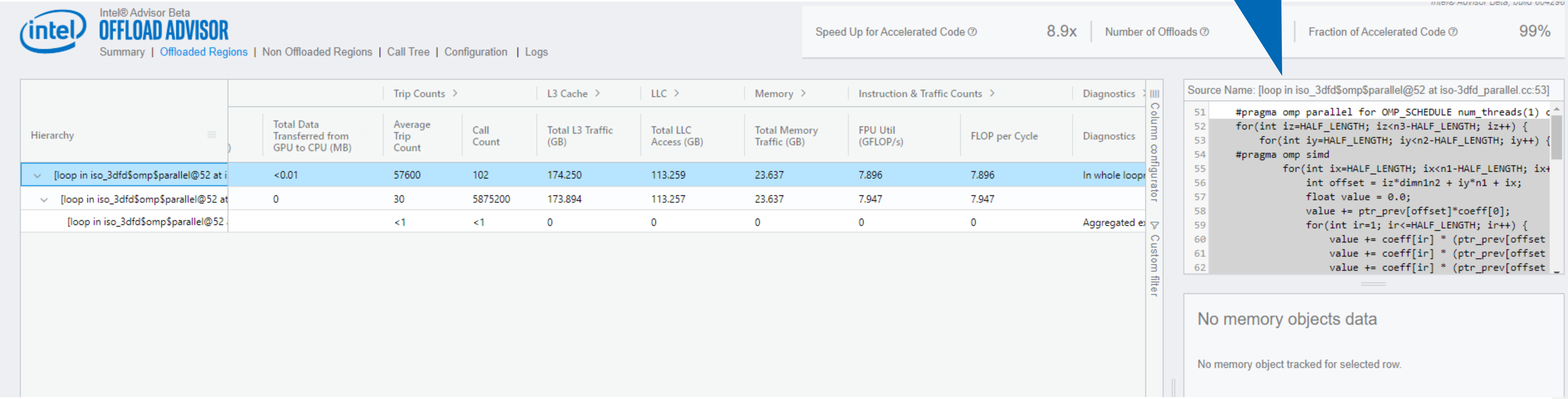
Top non offloaded

Location	Data Transfer	Execution Time	Why Not Offloaded
[loop in iso_3dfd at iso-3dfd_parallel.cc:85]	0.09MB	CPU 13.75s GPU 19.45s	Not profitable.
[loop in main at iso-3dfd_main.cc:194]	0MB	CPU 0.02s GPU <0.01s	Total time is too small for reliable modelling. Use --loop-filter-threshold=0 to model such small offloads.

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 tax 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 DPC++ or OMP offload .



In-Depth Analysis of Top Offload Regions

Loop metrics are matched with Source and Call Tree

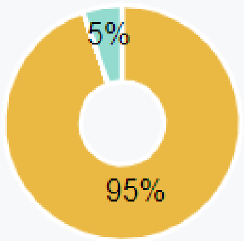
Source × | Top-Down × | Recommendations ×

Loop/Function	Measured	Basic Estimated Metrics			Estimated Bounded By			Estimated Data Transfer With Reuse
	Time	Speed-Up	Time	Offload Summary	Throughput	Taxes With Reuse	Latencies	
▼ Total	23.28s							
▼ func@0x4b2e8759	23.27s							
▼ func@0x4b2e8775	23.27s							
▼ BaseThreadInitThunk	23.27s							
▼ ThreadFunction	23.27s							
▼ multiply1	23.27s							
▶ [loop in multiply1 at multiply.c:53]	23.27s	3.326x	6....	Offloaded	LLC... 6.... L3 ... 3.2...	Launch Tax < 0.1ms All Taxes < 0.1ms	L... < 0....	Read 101MB Write 0B
▶ _scrt_common_main_seh	98.5ms							

What Is My Workload Bounded By?

Offloads bounded by

Compute	0%
L3 Cache BW	95%
LLC BW	0%
Memory BW	0%
Data Transfer	0%
Invoke Tax	0%
Transfer Tax	0%
Dependency	0%
Trip Count	0%
Unknown	0%
Non Offloaded	5%



Predict performance on future GPU hardware.

95% of workload bounded by L3 bandwidth but you may have several bottlenecks.

Offload Information >

Estimated Speed Up	Estimated Time on Target	Total Execution Time by Compute (s)	Total Execution Time by Memory BW Time (s)	Total Execution Time by LLC BW (s)	Total Execution Time by L3 BW (s)	Parallel Threads	Bounded By
1.80x	0.377s (79.05%)	0.0438	0.0436	0.133	0.377	192	L3_BW

Will the Data Transfer Make GPU Offload Worthwhile?

Memory objects

Tracked 3 memory objects with 24.0 MB total size

Size ↓	Source Location	Address	Transfer Direction	Type
8.0 MB	matrix.c:116	0x7f248c6...	host_to_devi...	heap
8.0 MB	matrix.c:121	0x7f248be...	host_to_devi...	heap
8.0 MB	matrix.c:126	0x7f248b6...	device_to_h...	heap

Memory histogram

Memory object histogram

0 - 1 MB
1 MB - 10 MB 3 item(s): 24.0 MB
10 MB - 100 MB
100 MB - 1 GB
> 1 GB

Total data transferred

Overhead >				Data Transfer >		
Offload Taxes	Data Transfer Tax	Invocation Tax	Configuratic Tax	Total Data Transferred (MB)	Total Data Transferred from CPU to GPU (MB)	Total Data Transferred from GPU to CPU (MB)
0.00000520	0	0.00000500	0	25.17MB	25.17	0
0.00512	0	0.00512	0	25.17MB	25.17	0
5.24	0	5.24	0	25.17MB	16.78	8.39

What Kernels 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

Top non offloaded

Location	Data Transfer	Execution Time	Why Not Offloaded
[loop in GSimulation::startompparallel@133 at GSimulation.cpp:134]	1.15MB	CPU 0.14s GPU 0.167	Not profitable: Computation Time is high despite the full use of Target Device capabilities
[loop in GSimulation::startompparallel@133 at GSimulation.cpp:158]	0.96MB	CPU 0.14s GPU 0.193	Not profitable: Computation Time is high despite the full use of Target Device capabilities
[loop in GSimulation::start at GSimulation.cpp:130]	0.87MB	CPU 0.15s GPU 32.027	Not profitable: Parallel execution efficiency is limited due to Dependencies

Compare Acceleration on Different GPUs

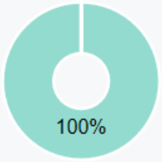
Gen9 – Not profitable to offload kernel

Speed Up for Accelerated Code ⓘ	1.0x	Number of Offloads ⓘ	0	Fraction of Accelerated Code ⓘ	0%

Program metrics ⓘ

Original ⓘ	0.15s
Accelerated ⓘ	0.15s

Target Platform	**Gen9 GT2**	Time on Host ⓘ	0.15s
Number of Offloads ⓘ	0	Time on Accelerator ⓘ	0s
Speed Up for Accelerated Code ⓘ	1.0x	Data Transfer Tax ⓘ	s
Amdahl's Law Speed Up ⓘ	1.0x	Invocation Tax ⓘ	s
Fraction of Accelerated Code ⓘ	0%	Code Transfer Tax ⓘ	s



CPU	0.14s
GPU	0.167

Not profitable: Computation Time is high despite the full use of Target Device capabilities

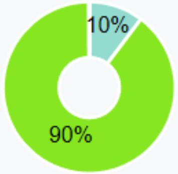
Gen11 – 1.6x speedup

Speed Up for Accelerated Code ⓘ	1.6x	Number of Offloads ⓘ	1	Fraction of Accelerated Code ⓘ	98%

Program metrics ⓘ

Original ⓘ	0.15s
Accelerated ⓘ	0.10s

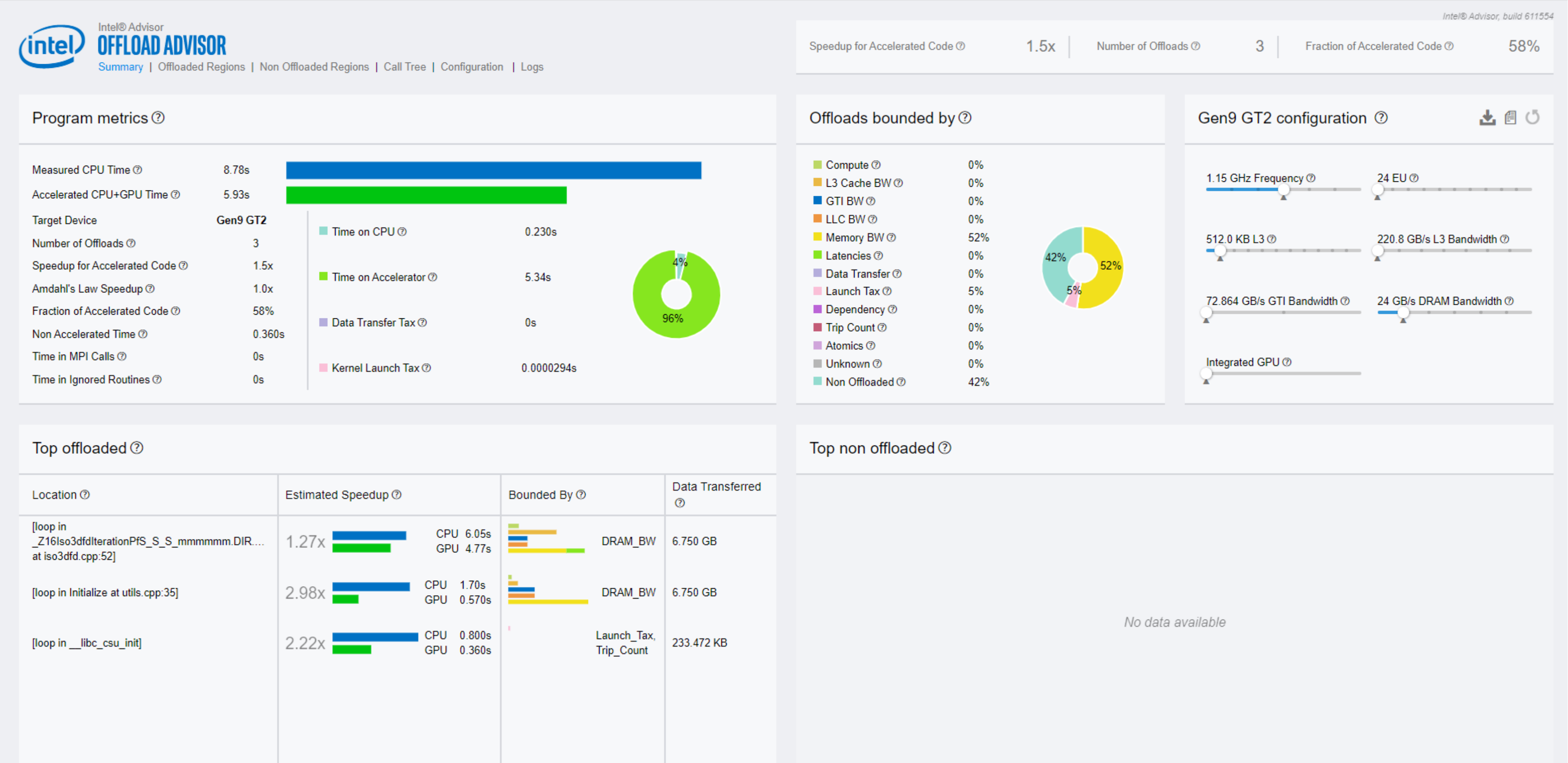
Target Platform	**Gen11 GT2**	Time on Host ⓘ	<0.01s
Number of Offloads ⓘ	1	Time on Accelerator ⓘ	0.09s
Speed Up for Accelerated Code ⓘ	1.6x	Data Transfer Tax ⓘ	0s
Amdahl's Law Speed Up ⓘ	1.7x	Invocation Tax ⓘ	<0.01s
Fraction of Accelerated Code ⓘ	98%	Code Transfer Tax ⓘ	<0.01s



1.60x	CPU 0.14s	GPU 0.09s	Compute	1.15MB

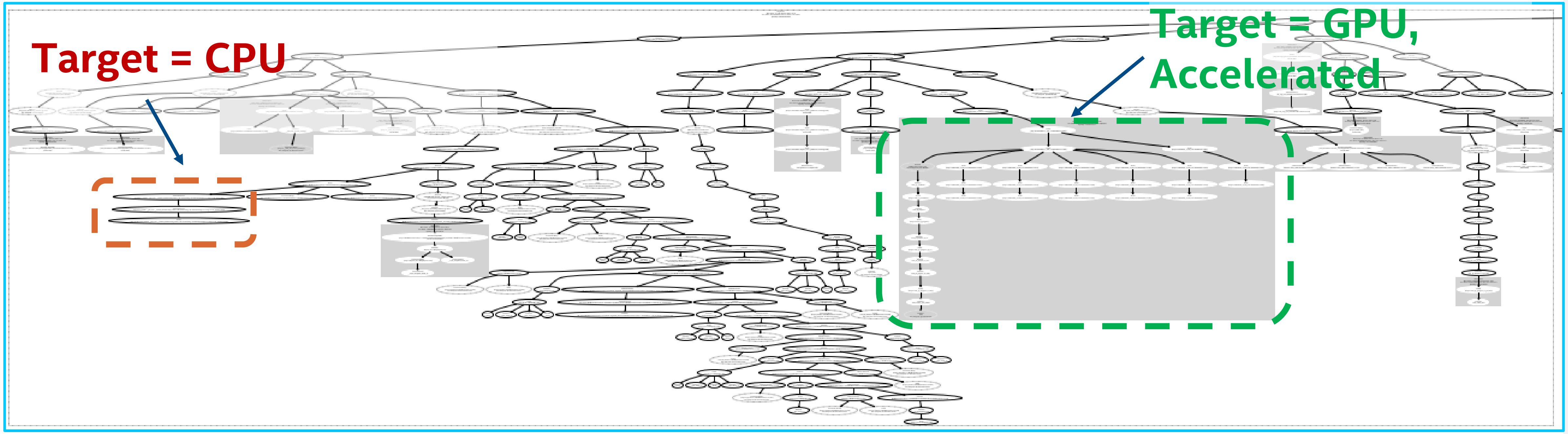
Share or View the results in web browser

With HTML UI, no need to install Intel® Advisor



Program Tree

- The program tree offers another view of the proportion of code that can be offloaded to the accelerator.
  - Generated if the DOT(GraphViz*) utility is installed



Before you start to use 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)

PERFORMANCE ESTIMATION FLOW

Performance estimation steps:

A. Profiling

B. Performance modelling

3 different approaches to get estimation:

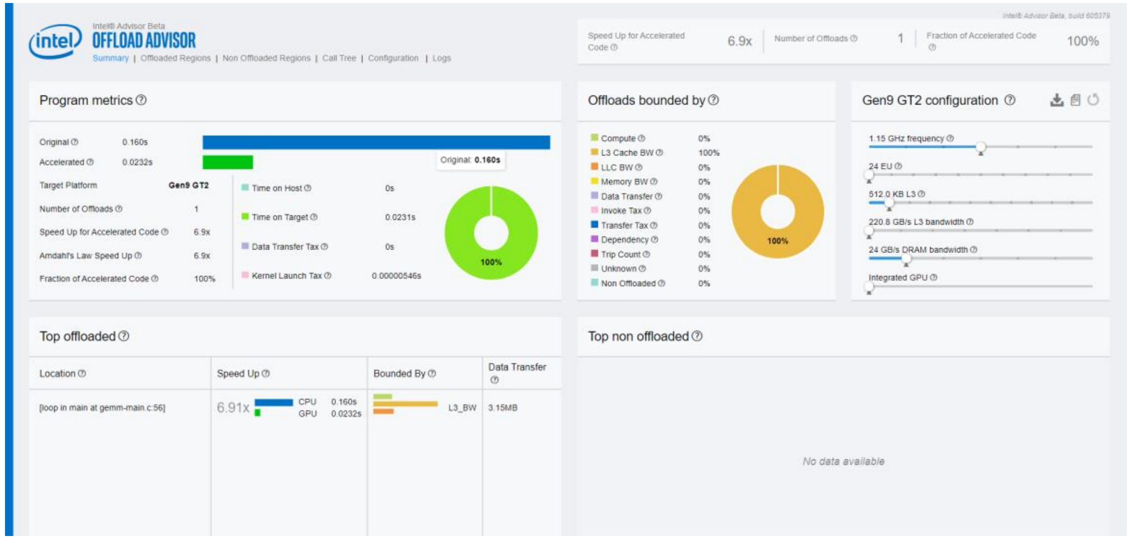
- `run_oa.py` (both A and B), most convenient
- `collect.py` (A) + `analyze.py` (B)
- `advisor` (multiple times, A)
  + `analyze.py` (B), most control

Performance estimation result:

- List of loops to be offloaded
- Estimated speed-up (relative to baseline)

Output:

1. report.html



2. report.csv (whole grid in CSV table)
   For batch processing

Using Python scripts to run Offload Advisor

- Set up the Intel® Advisor environment
  (implicitly done by oneAPI `setvars.sh`)
  `source <advisor_install_dir>/advixe-vars.sh`

  Environment variable `APM` points to `<ADV_INSTALL_DIR>/perfmodels`

Analyze for a specific GPU config

- Run the data collection
  `advisor-python $APM/collect.py <project_dir> --config gen9 -- <app> [app_options]`

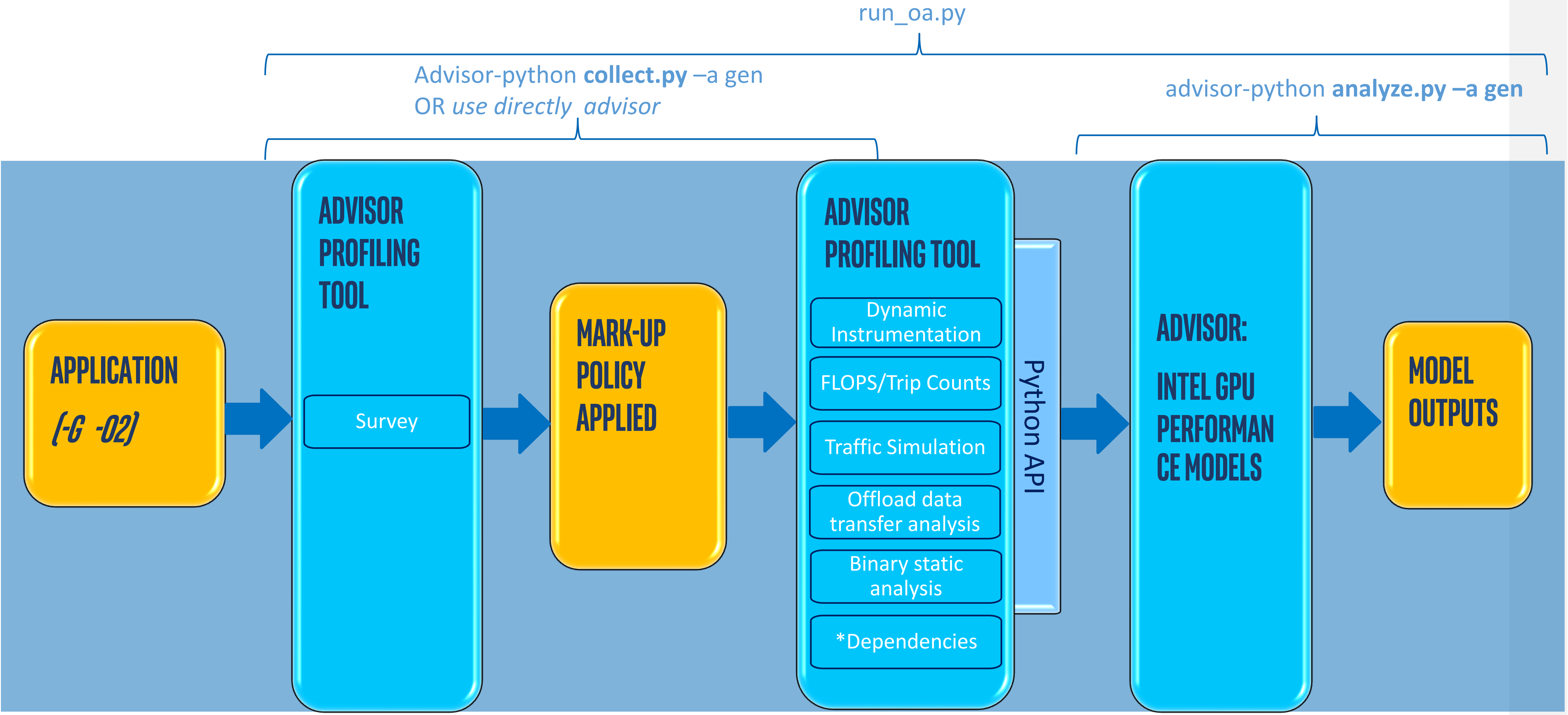
  Also works with other installed python, `advisor-python` only provided for convenience.

- Run the performance modelling
  `advisor-python $APM/analyze.py <project_dir> --config gen9 --out-dir <proj_results>`

  View the report.html generated (or generate a command-line report)

- Alternatives: `run_oa.py` or `advisor + analyze-py`

Run_oa.py: What is running behind?



Avoid Dependency Checking

- Dependency adds a lot of time to the collection and you might want to avoid it
- Add the option `--collect basic` for the collection:

```
advisor-python $APM/run_oa.py <project_dir> –config gen9
–c basic --out-dir <proj_results> [--options] -- <app>
```

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

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

Focus on Specific Loops

- Analyzing all loops adds a lot of time to the collection and you might want to focus on specific loops
- Add the option --markup for the collection, for example to focus on OpenMP loops

```
advisor-python $APM/run_oa.py <project_dir> –config gen9
--markup omp --out-dir <proj_results> [--options] -- <app>
```

Contents of Output Directory

- 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

Offload Modelling: GPU-to-GPU

How to run GPU-to-GPU with Advisor CLI

1. Measure the hardware metrics of GPU-enabled kernels (for example, memory traffic):
   ```
   advisor --collect=survey --profile-gpu -- <my_app> [app_parameters]
   ```
2. Get the number of floating-point and integer operations on Gen9 target device:
   ```
   advisor --collect=tripcounts --flop --data-transfer=medium
            --profile-gpu -- <my_app> [app_parameters]
   ```
3. Model application performance on the specified Gen11 target GPU:
   ```
   advisor --collect=projection --profile-gpu --config=gen11
   ```

Alternative: Use a single command to run performance modeling (non-MPI case)

```
advisor --collect=offload --gpu --config=gen11
         -- <my_app> [app_parameters]
```

How to run GPU-to-GPU with Python scripts

1. Collect characterization metrics with `$APM/collect.py` script (non MPI case):
```
advisor-python $APM/collect.py --gpu
        <my_project_directory> -- <my_app> [app_parameters]
```
2. Run performance model to Gen11with `$APM/analyze.py` script:
```
advisor-python $APM/analyze.py --gpu --config=gen11
        <my_project_directory> -o <path-to-report-dir>
```

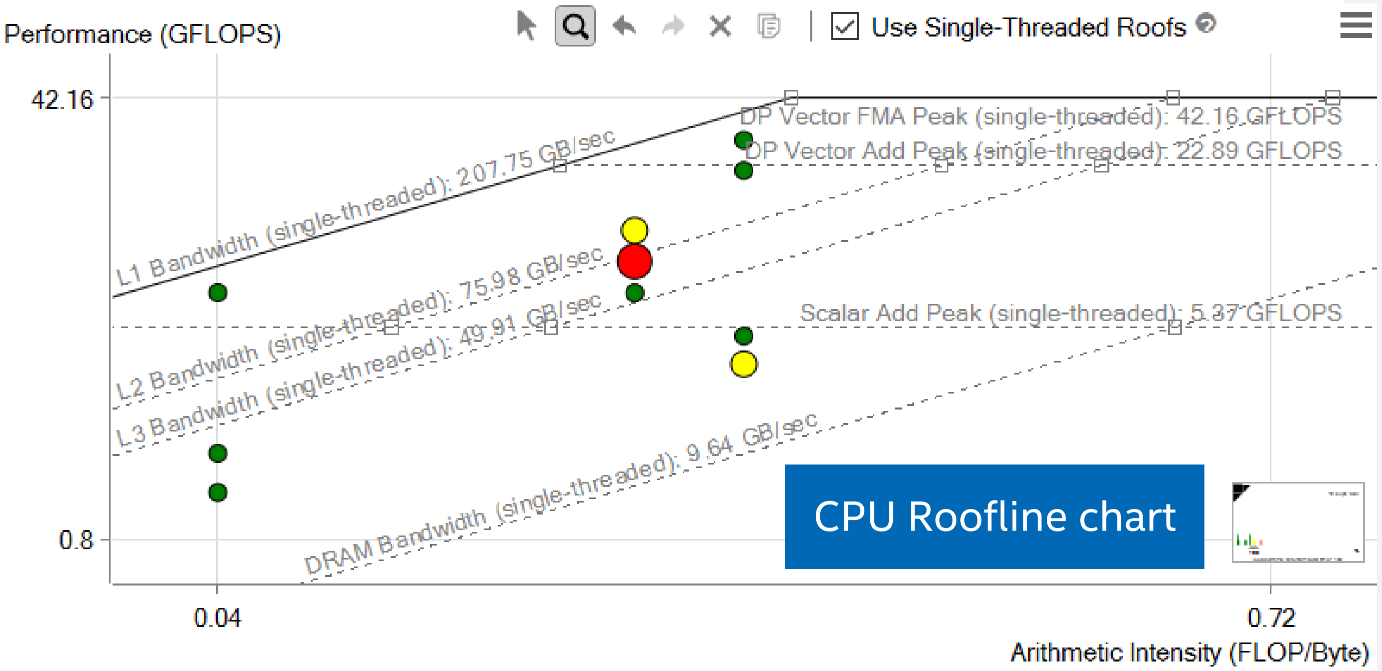
Alternative: Use `$APM/run_oa.py` to run performance model just after other collections.
```
advisor-python $APM/run_oa.py --gpu --config=gen11
        <my_project_directory> -- <my_app> [app_parameters]
```

Use `--set-parameter` option to scale target device, or specify configuration file

Roofline Analysis - Recap

What is a Roofline Chart?

- A Roofline Chart plots application performance against hardware limitations
  - Where are the bottlenecks?
  - How much performance is being left on the table?
  - What are the next steps?
- Values of Rooflines in Intel® Advisor are measured
  - Small benchmarks are run when starting a Roofline Analysis



Roofline first proposed by University of California at Berkeley:
Roofline: An Insightful Visual Performance Model for Multicore Architectures, 2009
Cache-aware variant proposed by University of Lisbon:
Cache-Aware Roofline Model: Upgrading the Loft, 2013

What is the Roofline Model?

Do you know how fast you should run?

- Comes from Berkeley
- Performance is limited by equations/implementation & code generation/hardware
- 2 hardware limitations
  - PEAK Flops
  - PEAK Bandwidth
- The application performance is bounded by hardware specifications

$$\text{Gflop/s} = \boldsymbol{min} \begin{cases} \boldsymbol{Platform\ PEAK} \\ \boldsymbol{Platform\ BW} \ * \boldsymbol{AI} \end{cases}$$

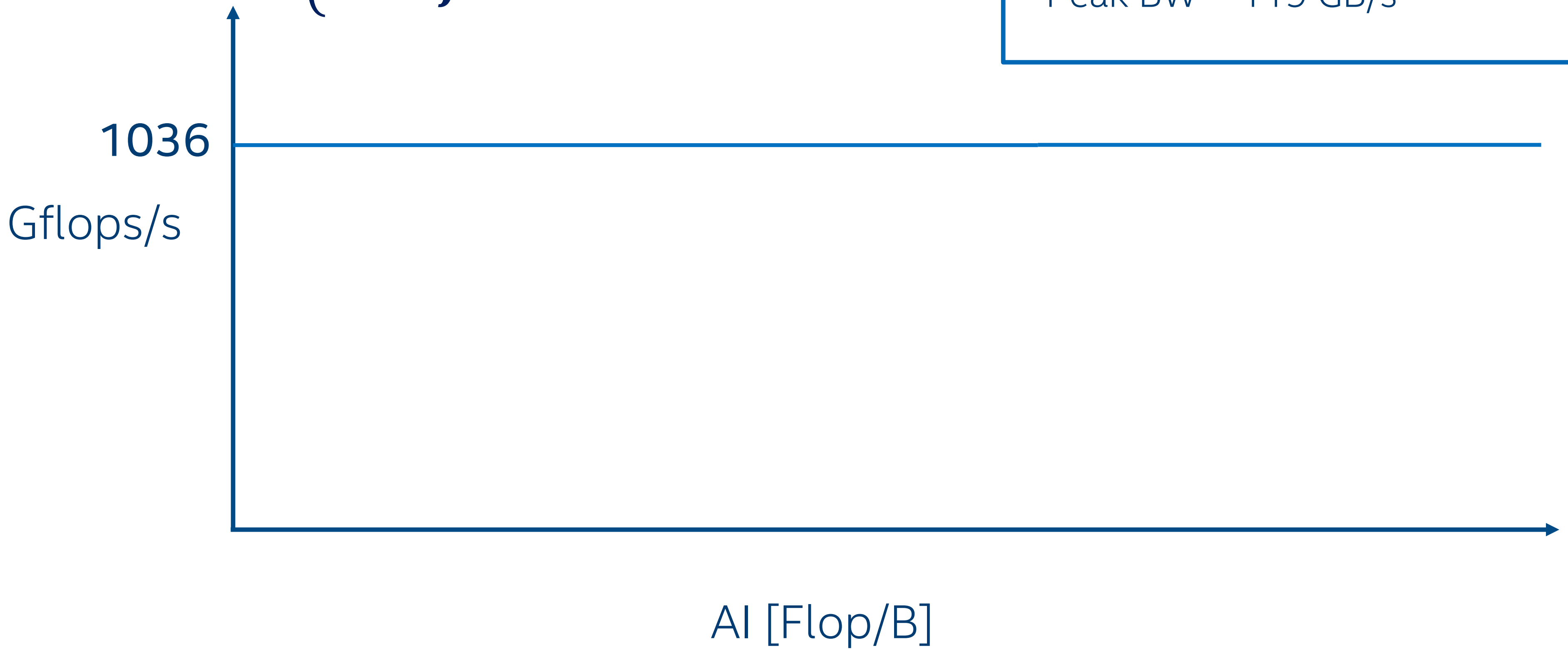
Arithmetic Intensity (Flops/Bytes)

DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = \boldsymbol{min}\begin{cases} \boldsymbol{Platform\ PEAK} \\ \boldsymbol{Platform\ BW} \ * \boldsymbol{AI} \end{cases}$$

2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s

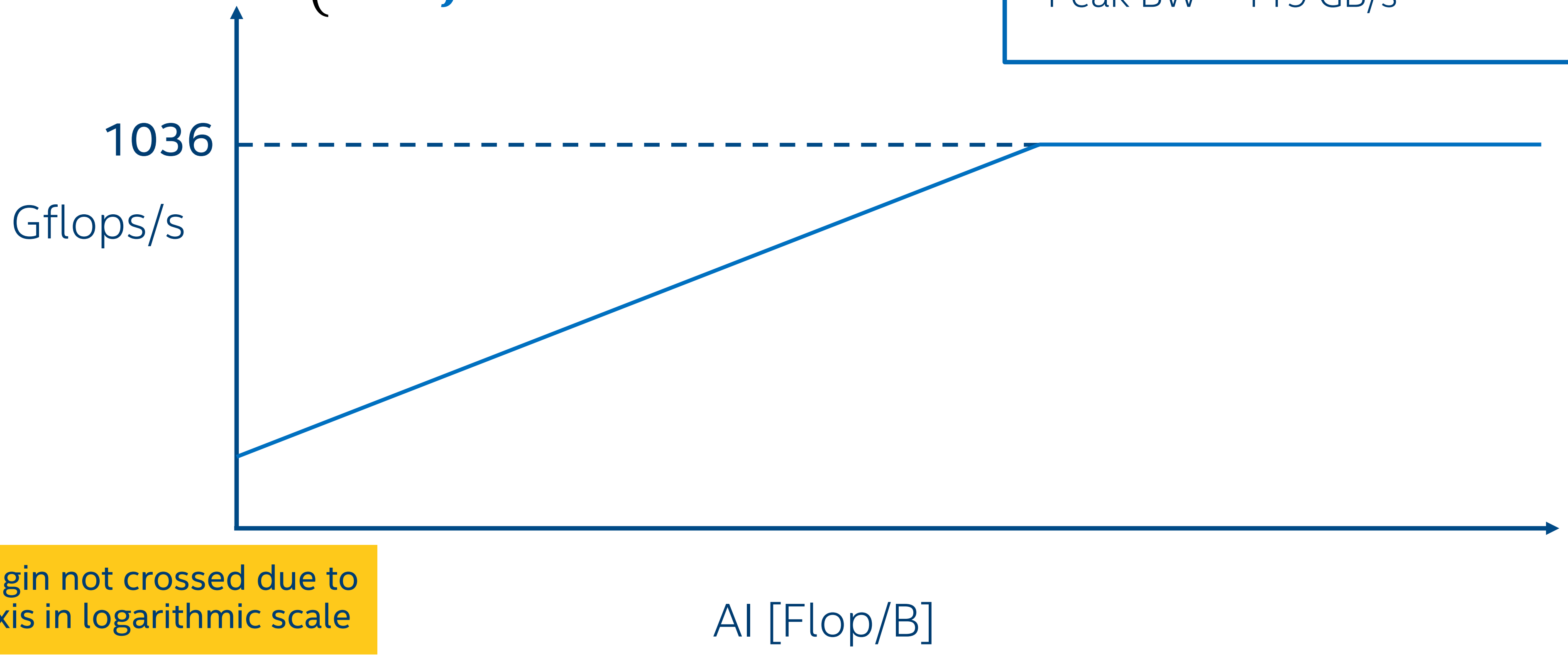


DRAWING THE ROOFLINE

Defining the speed of light

$$\text{Gflop/s} = \boldsymbol{min} \begin{cases} \boldsymbol{Platform\ PEAK} \\ \boldsymbol{Platform\ BW} * \boldsymbol{AI} \end{cases}$$

2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s



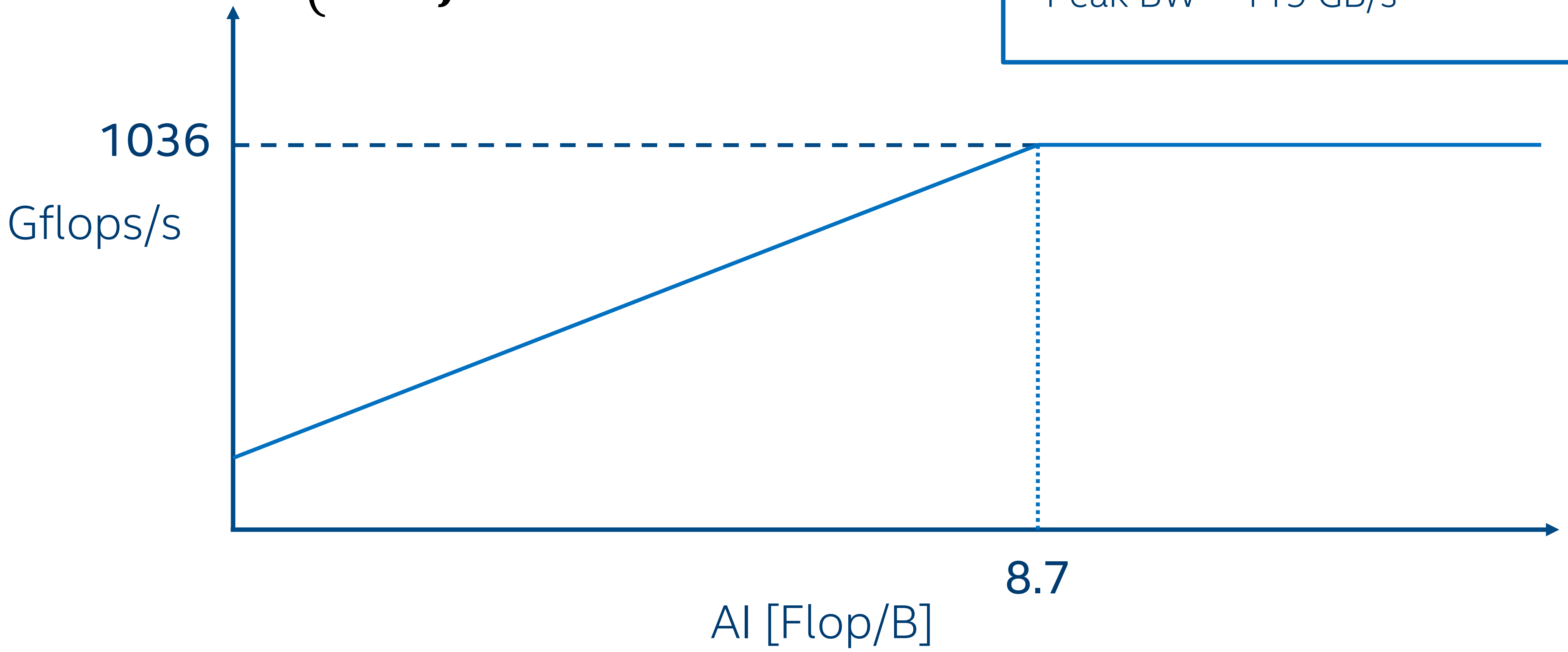
NB: Origin not crossed due to both axis in logarithmic scale

DRAWING THE ROOFLINE

Defining the speed of light

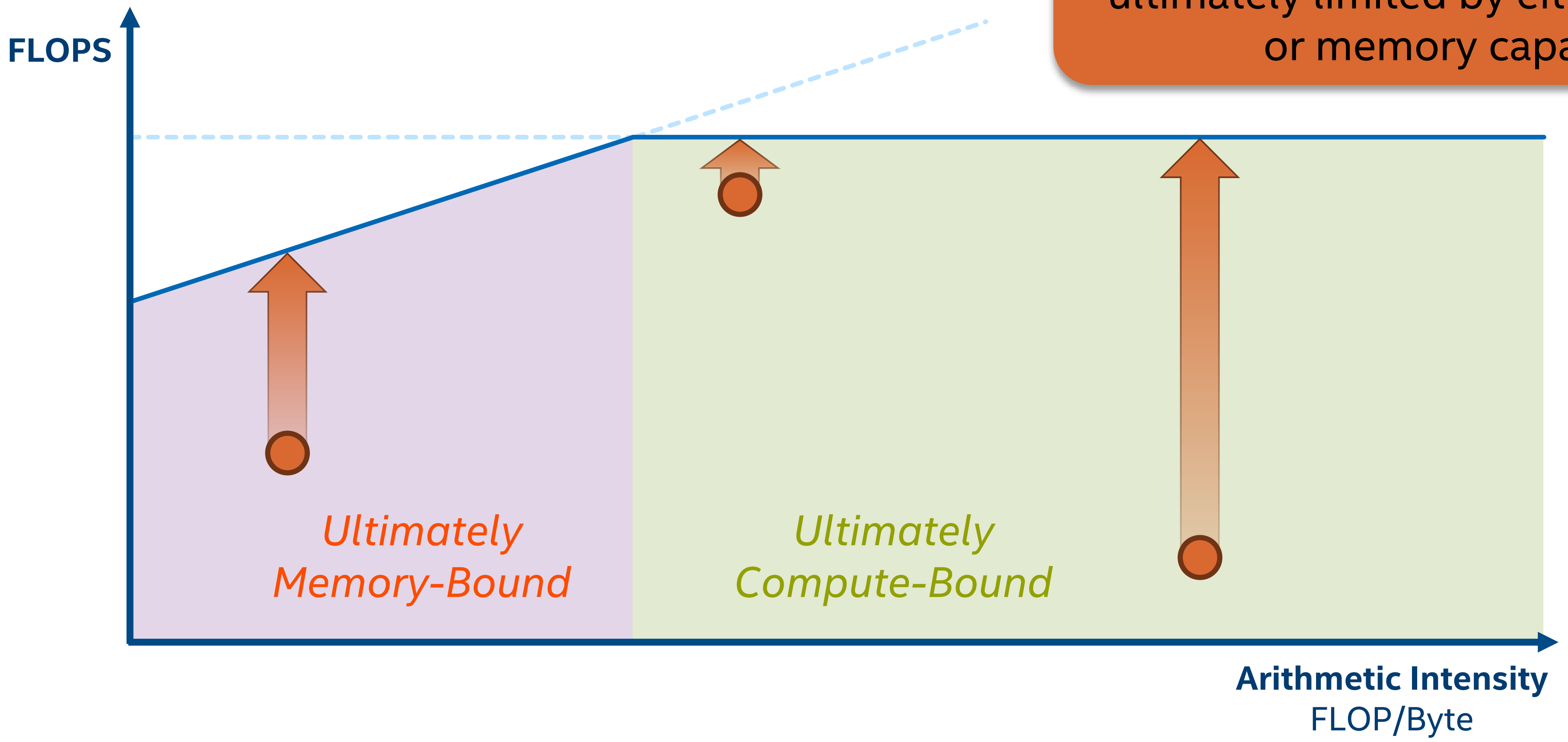
$$\text{Gflop/s} = \boldsymbol{min} \begin{cases} \boldsymbol{Platform\ PEAK} \\ \boldsymbol{Platform\ BW} * \boldsymbol{AI} \end{cases}$$

2 sockets Intel® Xeon® Processor E5-2697 v2
Peak Flop = 1036 Gflop/s
Peak BW = 119 GB/s



Ultimate Performance Limits

Performance cannot exceed the machine's capabilities, so each loop is ultimately limited by either compute or memory capacity.



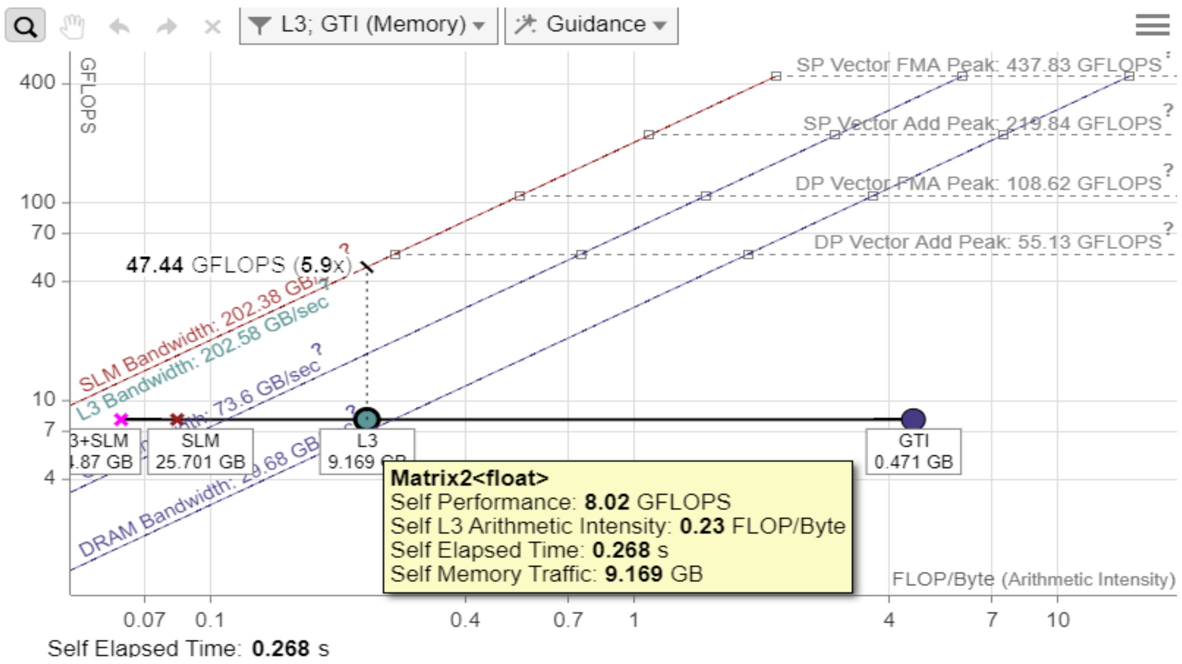
Roofline Analysis for GPU code

Find Effective Optimization Strategies

Intel® Advisor - GPU Roofline

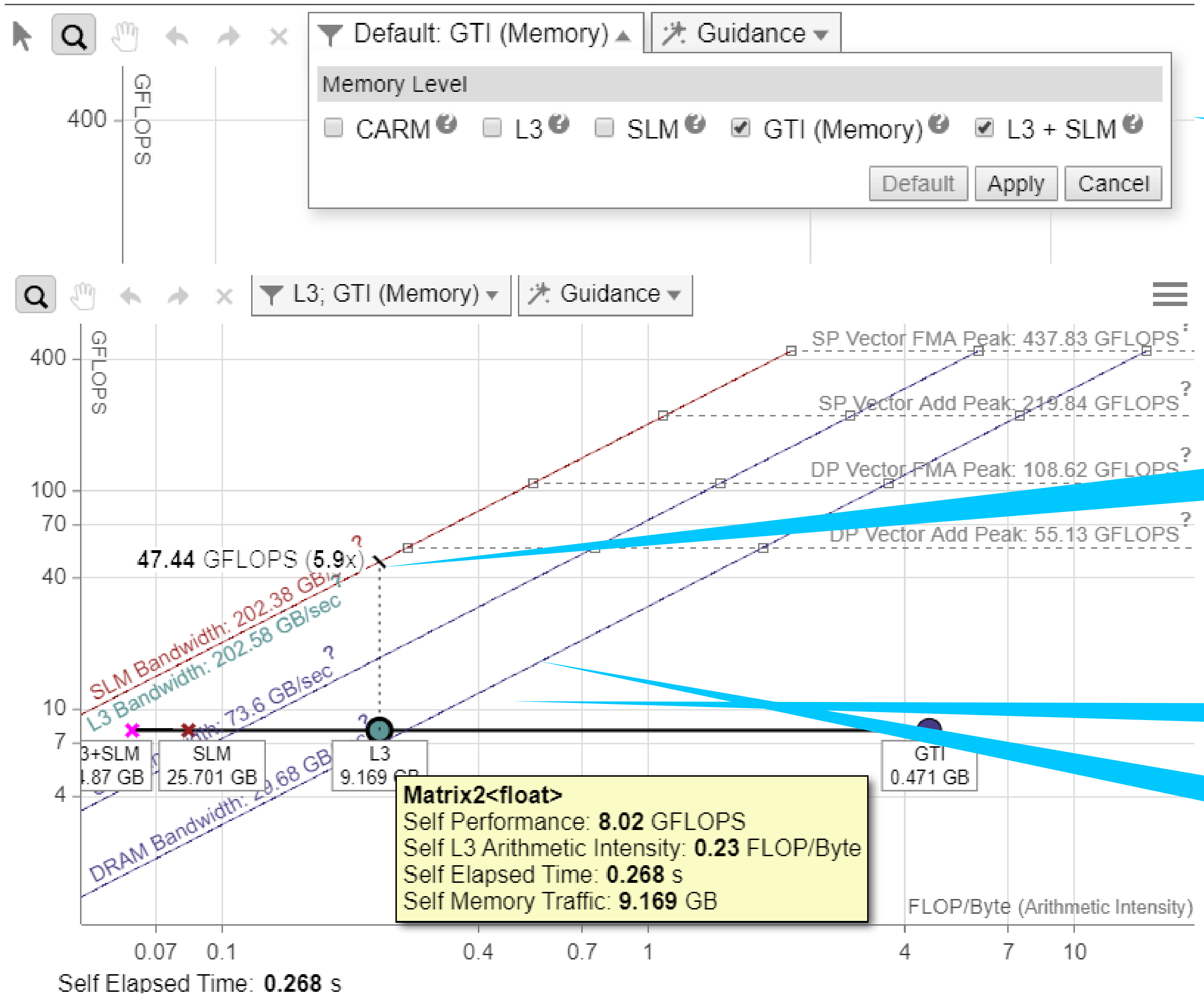
GPU Roofline Performance Insights

- 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



Find Effective Optimization Strategies

Intel® Advisor - GPU Roofline



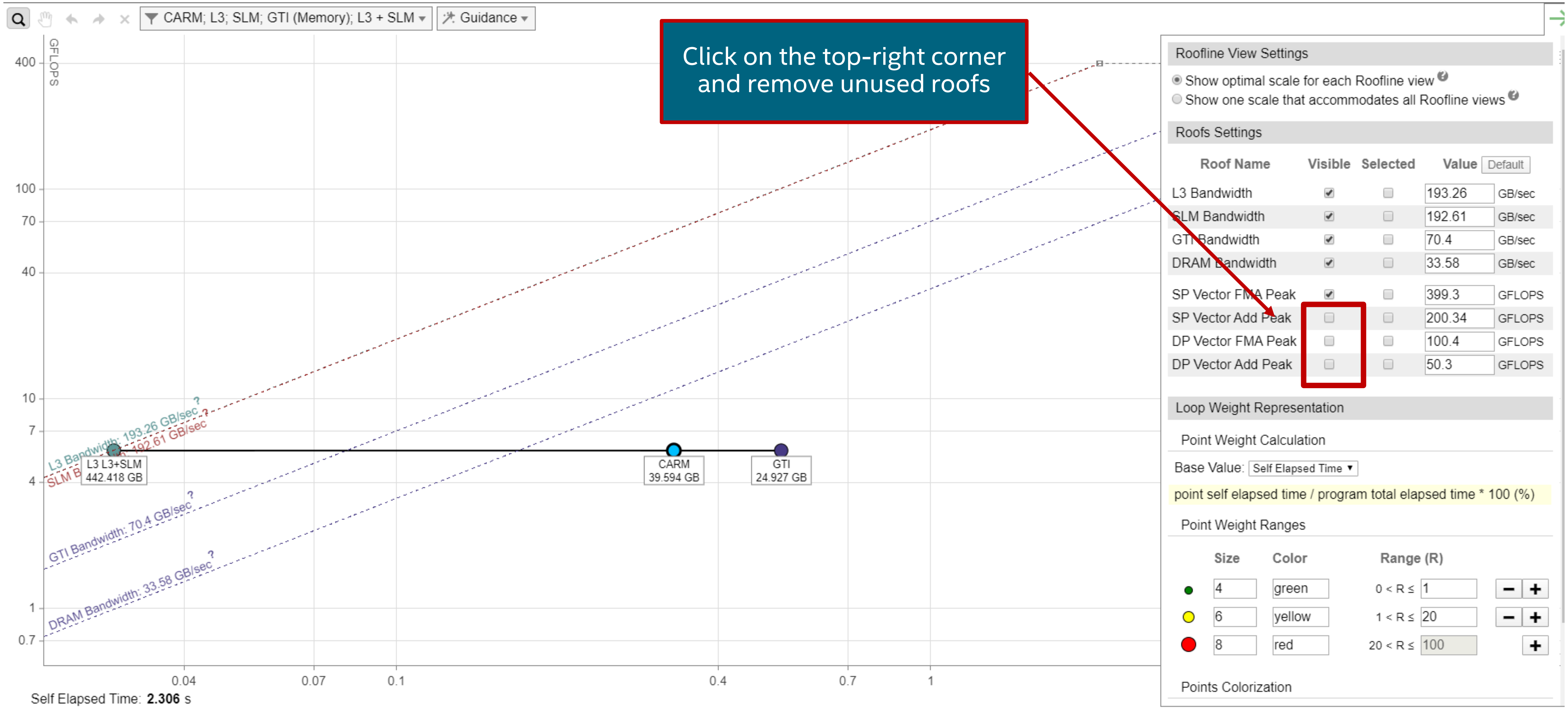
Configure levels to display

Shows performance headroom for each loop

Likely bottlenecks

Suggests optimization next steps

Customize to Display Only Desired Roofs



Click on the top-right corner and remove unused roofs

How to Run Intel® Advisor – GPU Roofline

Run 2 collections with `--profile-gpu` option.

First Survey run will do time measurements with minimized overhead:

```
advisor –collect=survey --profile-gpu
        --project-dir=<my_project_directory>
        --search-dir src:r=<my_source_directory> -- <my_app> [app_parameters]
```

Run the Trip Counts and FLOP data collection :

```
advisor –collect=tripcounts --stacks --flop --profile-gpu
        --project-dir=<my_project_directory>
        --search-dir src:r=<my_source_directory> -- <my_app> [app_parameters]
```

Generate a GPU Roofline report:

```
advisor --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.

Resources

OpenMP Resources

- Release Notes
  https://software.intel.com/content/www/us/en/develop/articles/intel-oneapi-dpc-c-compiler-release-notes.html
  (search for “OpenMP offload” or “OpenMP”)
- Get Started
  https://software.intel.com/content/www/us/en/develop/documentation/get-started-with-cpp-fortran-compiler-openmp
- Programming Guide “Debugging the DPC++ and OpenMP* Offload Process”
  https://software.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/software-development-process/debugging-the-dpc-and-openmp-offload-process.html
  - oneAPI Debug Tools “Debug Environment Variables”
    https://software.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/software-development-process/debugging-the-dpc-and-openmp-offload-process/oneapi-debug-tools.html
  - Trace the Offload Process
    https://software.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/software-development-process/debugging-the-dpc-and-openmp-offload-process/trace-the-offload-process.html
- Example
  https://github.com/oneapi-src/oneAPI-samples/tree/master/DirectProgramming/C++/StructuredGrids/iso3dfd_omp_offload

Advisor Resources

Intel® Advisor

- Product page – overview, features, FAQs…
- What's New?
- Training materials – Cookbook, User Guide, Tutorials
- Support Forum
- Priority Support - Online Service Center

Additional Analysis Tools

- Intel® VTune™ Profiler – performance profiler
- Intel® Inspector – memory and thread checker/ debugger
- Intel® Trace Analyzer and Collector - MPI Analyzer and Profiler

All Development Products

- Intel® oneAPI Toolkits

Backup

Command Line Tips

Source Offload Advisor

- To set up the Intel® Advisor 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 installation directory

Detailed approach (more control)

- 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.

```
advisor-python $APM/run_oa.py <project_dir> –config gen9
--dry-run –c basic --out-dir <path_to_result_dir> [--options]
-- <app>
```

Detailed approach (step 1) - Survey

- We start with the survey

```
advisor --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)

Detailed approach (step 2) – Tripcounts & Caches

- We continue with the trip count and cache simulation

```
advisor --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 speedown

Detailed approach (optional) - Dependencies

- Optional step: Dependency analysis

```
advisor --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

Detailed approach - Report

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

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

  - You didn't run the dependency analysis

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

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. https://software.intel.com/en-us/articles/optimization-notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

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

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.