Intel® Advisor Offload Modeling

Dmitry Tarakanov

Software Technical Consulting Engineer

Notices & Disclaimers

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex.

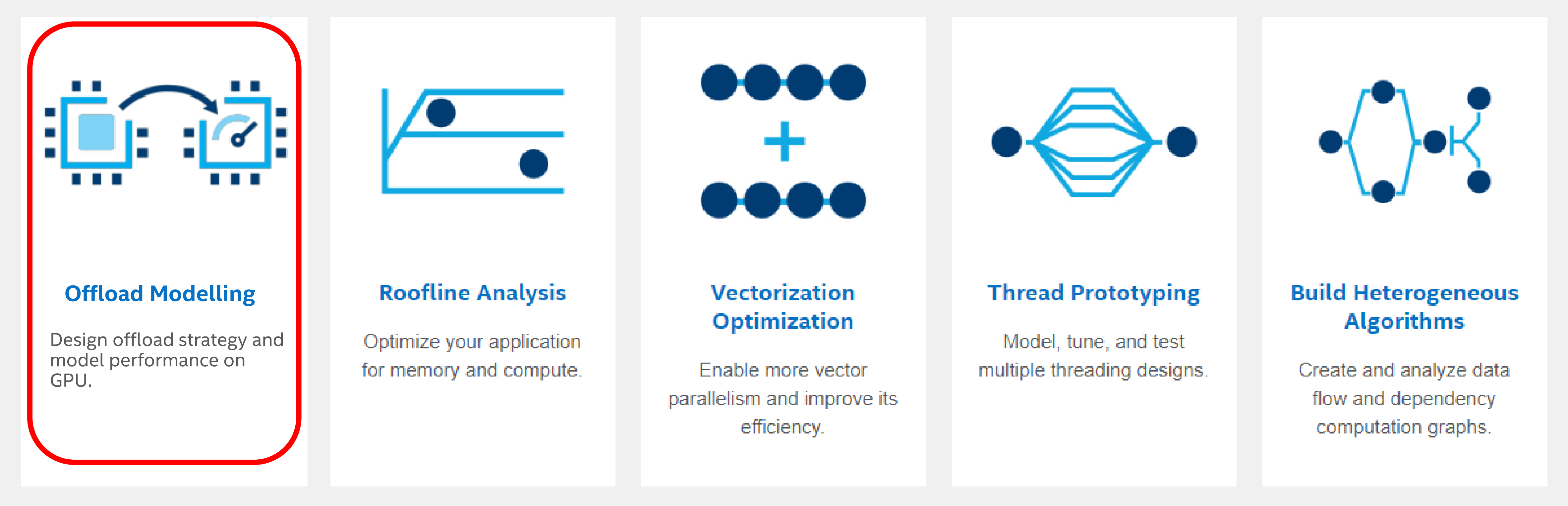
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Rich Set of Capabilities for High Performance Code Design

Intel® Advisor



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.

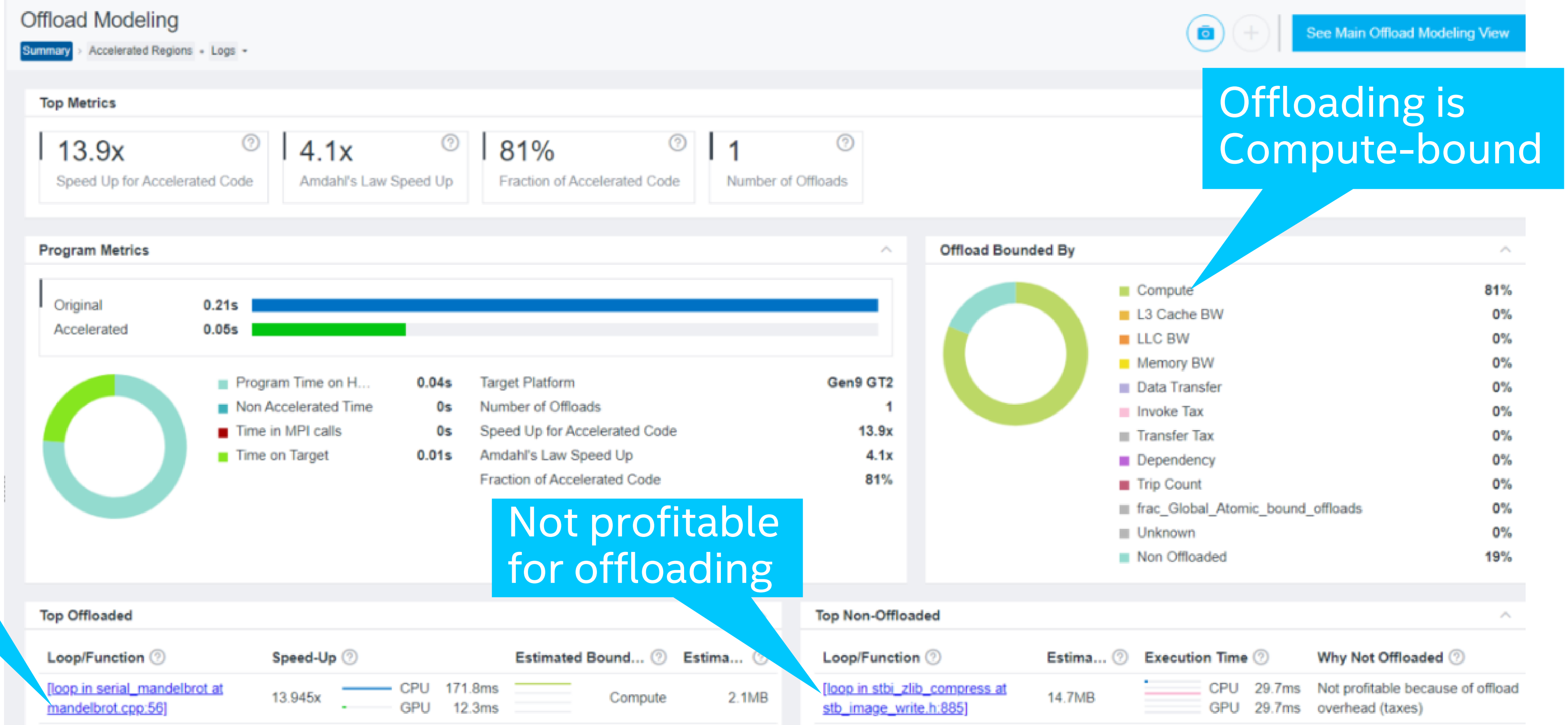
Build Heterogeneous Algorithms

Create and analyze data flow and dependency computation graphs.

Intel® Advisor - Offload Modeling

“Run on CPU or GPU – Predict for GPU”

- Helps to define which sections of the code should run on a given accelerator
- Provides performance projection on accelerators



Offload Modeling

Summary › Accelerated Regions ▾ Logs ▾
See Main Offload Modeling View

Top Metrics

13.9x	4.1x	81%	1
Speed Up for Accelerated Code	Amdahl's Law Speed Up	Fraction of Accelerated Code	Number of Offloads

Program Metrics

Original	0.21s	
Accelerated	0.05s	

Program Time on H...	0.04s
Non Accelerated Time	0s
Time in MPI calls	0s
Time on Target	0.01s

Target Platform	Gen9 GT2
Number of Offloads	1
Speed Up for Accelerated Code	13.9x
Amdahl's Law Speed Up	4.1x
Fraction of Accelerated Code	81%

Offload Bounded By

Compute	81%
L3 Cache BW	0%
LLC BW	0%
Memory BW	0%
Data Transfer	0%
Invoke Tax	0%
Transfer Tax	0%
Dependency	0%
Trip Count	0%
frac_Global_Atomic_bound_offloads	0%
Unknown	0%
Non Offloaded	19%

Top Offloaded

Loop/Function	Speed-Up		Estimated Bound...	Estima...
[loop in serial_mandelbrot at mandelbrot.cpp:56]	13.945x	CPU 171.8ms GPU 12.3ms	Compute	2.1MB

Top Non-Offloaded

Loop/Function	Estima...	Execution Time	Why Not Offloaded
[loop in stbi_zlib_compress at stb_image_write.h:885]	14.7MB	CPU 29.7ms GPU 29.7ms	Not profitable because of offload overhead (taxes)

Offloading is Compute-bound

Recommended for offloading

Not profitable for offloading

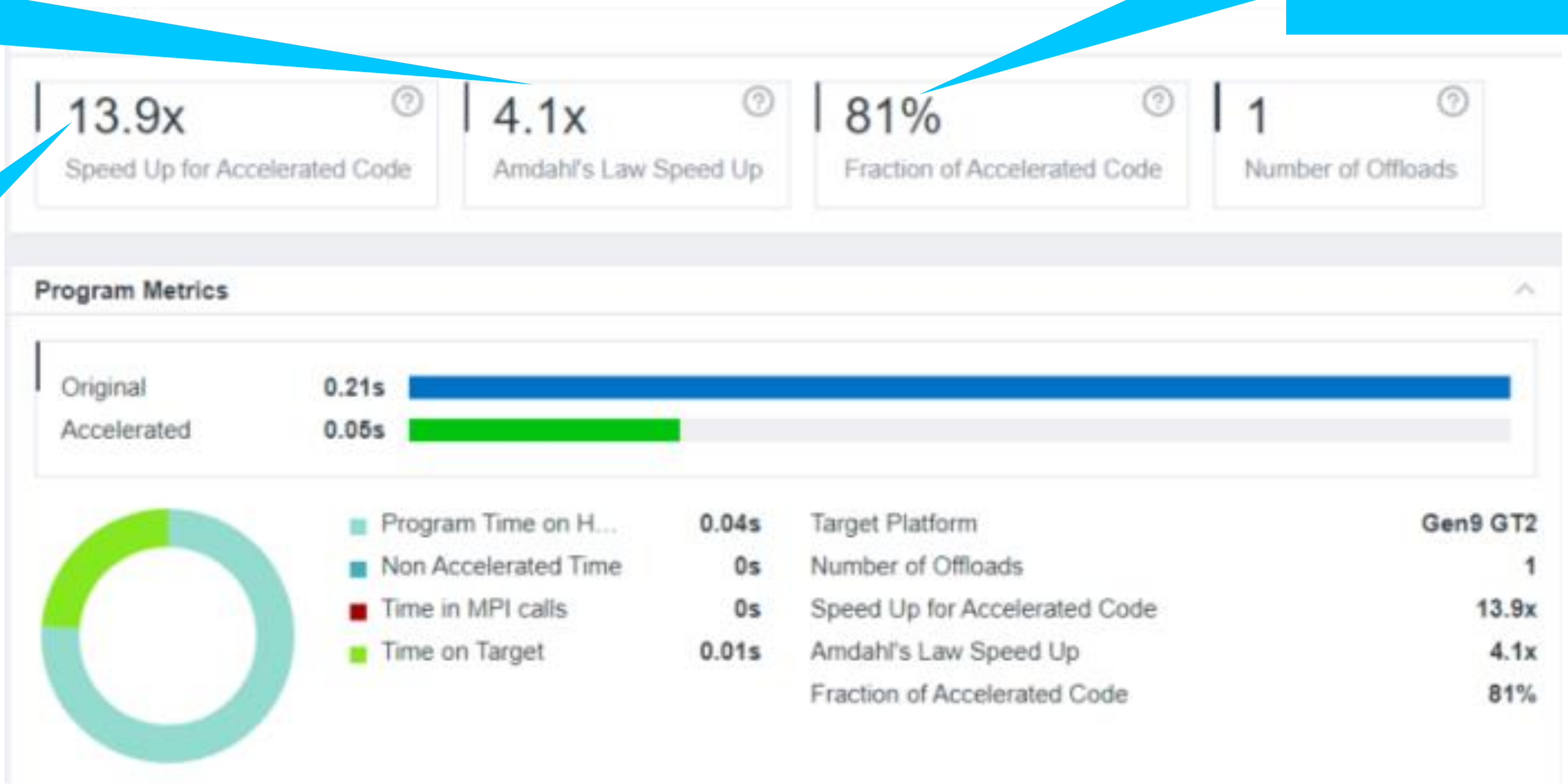
Intel® Advisor - Offload Modeling

Find code that can be profitably offloaded

The whole app is 4.1x faster

Loop takes 81% of the whole app execution time

Loop on GPU is 13.9x faster than on CPU



13.9x
Speed Up for Accelerated Code

4.1x
Amdahl's Law Speed Up

81%
Fraction of Accelerated Code

1
Number of Offloads

Program Metrics

Original	0.21s
Accelerated	0.05s

🟦 Program Time on H...	0.04s
🟦 Non Accelerated Time	0s
🟥 Time in MPI calls	0s
🟩 Time on Target	0.01s

Target Platform	Gen9 GT2
Number of Offloads	1
Speed Up for Accelerated Code	13.9x
Amdahl's Law Speed Up	4.1x
Fraction of Accelerated Code	81%

Intel® Advisor - Offload Modeling

Find code that can be profitably offloaded

	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	(DPC++, OCL, OMP, “target=offload”)	→	**CPU** measured	+	**GPU** estimated

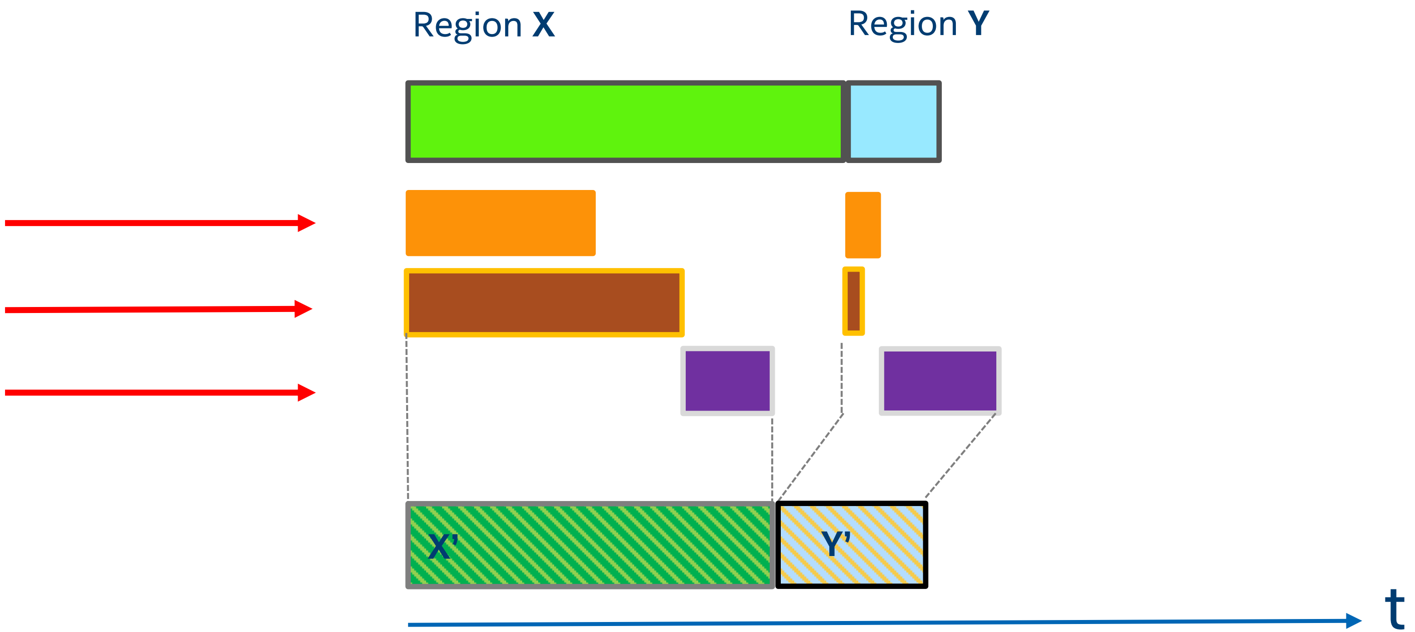
Intel® Advisor - Offload Modeling

Find code that can be profitably offloaded

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, $t(Y)<t(Y')$

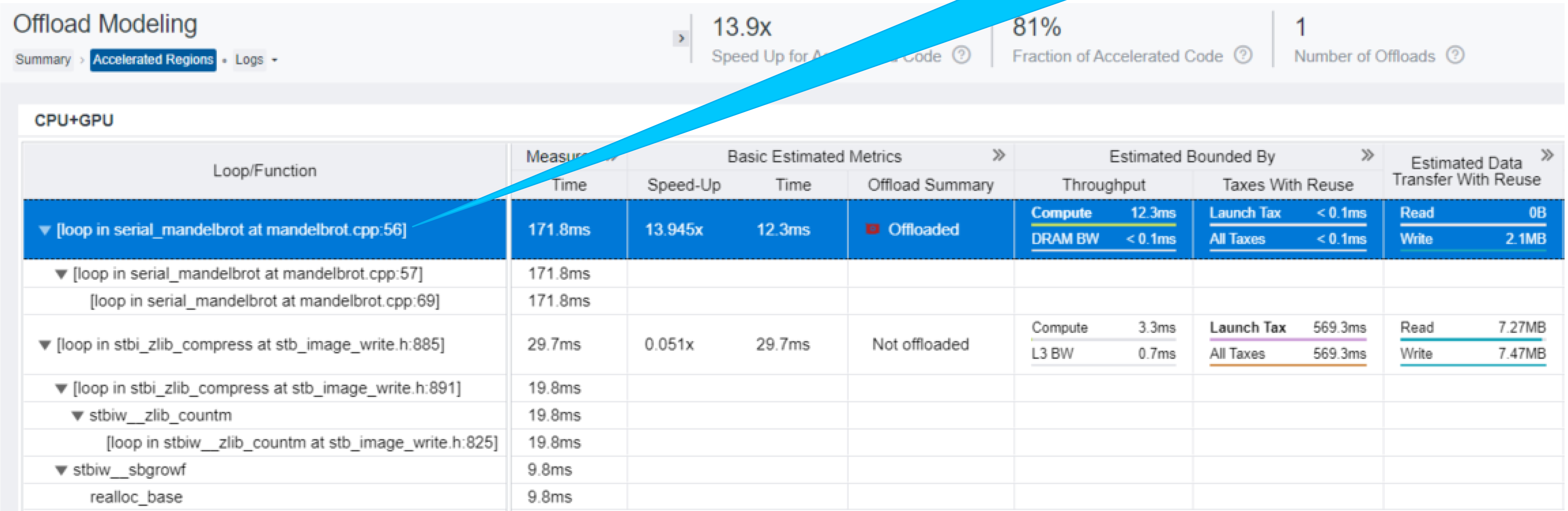
$$t_{\text{region}} = \max(t_{\text{compute}}, t_{\text{memory subsystem}}) + t_{\text{data transfer tax}} + t_{\text{invocation tax}}$$

In-Depth Analysis of Top Offload Regions

- Provides a detailed description of modeling for each loop
  - 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

Loop at mandelbrot.cpp:56 is recommended for offloading

- Compute-bound
- Estimated to run on GPU in 12.3ms
- Transfers 2.1MB of data



Offload Modeling

13.9x Speed Up for A Code

81% Fraction of Accelerated Code

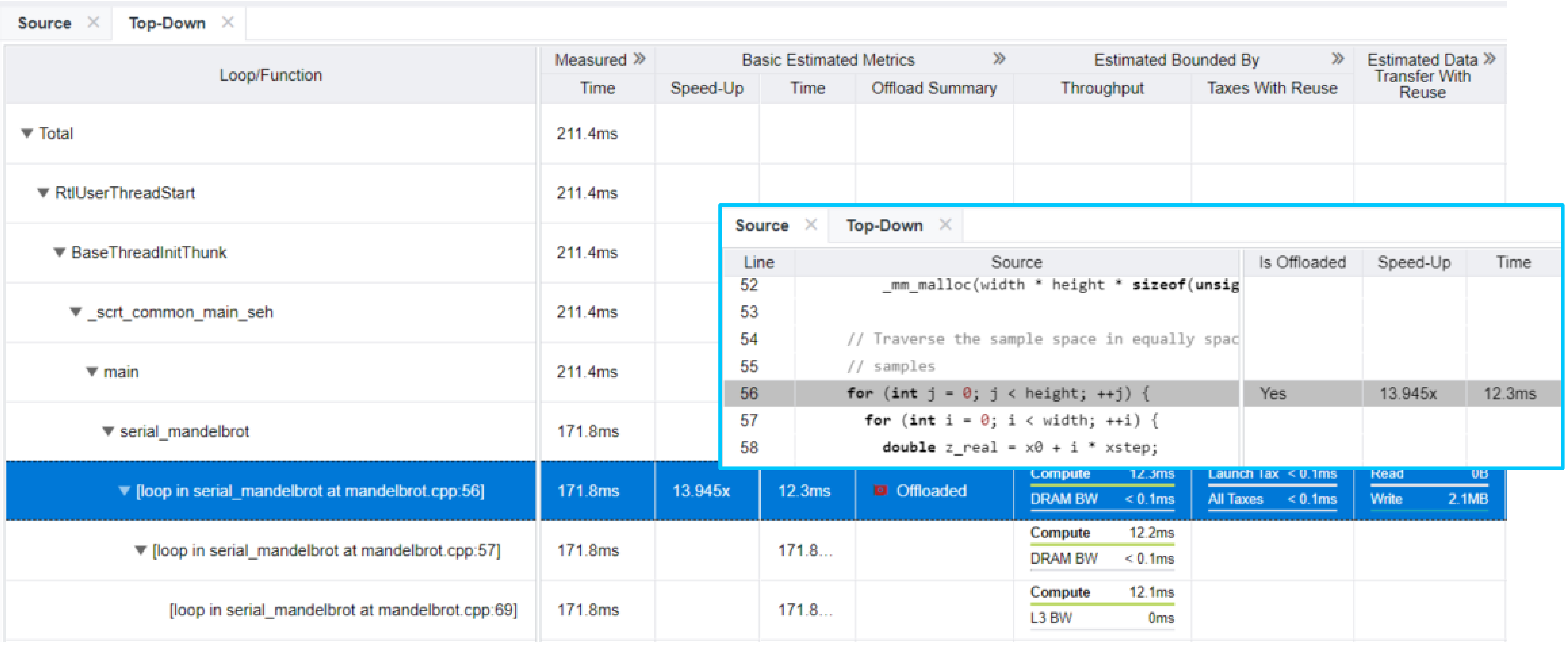
1 Number of Offloads

CPU+GPU

Loop/Function	Measur	Basic Estimated Metrics			Estimated Bounded By		Estimated Data Transfer With Reuse
	Time	Speed-Up	Time	Offload Summary	Throughput	Taxes With Reuse	
▼ [loop in serial_mandelbrot at mandelbrot.cpp:56]	171.8ms	13.945x	12.3ms	Offloaded	Compute 12.3ms DRAM BW < 0.1ms	Launch Tax < 0.1ms All Taxes < 0.1ms	Read 0B Write 2.1MB
▼ [loop in serial_mandelbrot at mandelbrot.cpp:57]	171.8ms						
[loop in serial_mandelbrot at mandelbrot.cpp:69]	171.8ms						
▼ [loop in stbi_zlib_compress at stb_image_write.h:885]	29.7ms	0.051x	29.7ms	Not offloaded	Compute 3.3ms L3 BW 0.7ms	Launch Tax 569.3ms All Taxes 569.3ms	Read 7.27MB Write 7.47MB
▼ [loop in stbi_zlib_compress at stb_image_write.h:891]	19.8ms						
▼ stbiw__zlib_countm	19.8ms						
[loop in stbiw__zlib_countm at stb_image_write.h:825]	19.8ms						
▼ stbiw__sbgrowf	9.8ms						
realloc_base	9.8ms						

In-Depth Analysis of Top Offload Regions

Loop metrics are matched with Sources and Call Tree

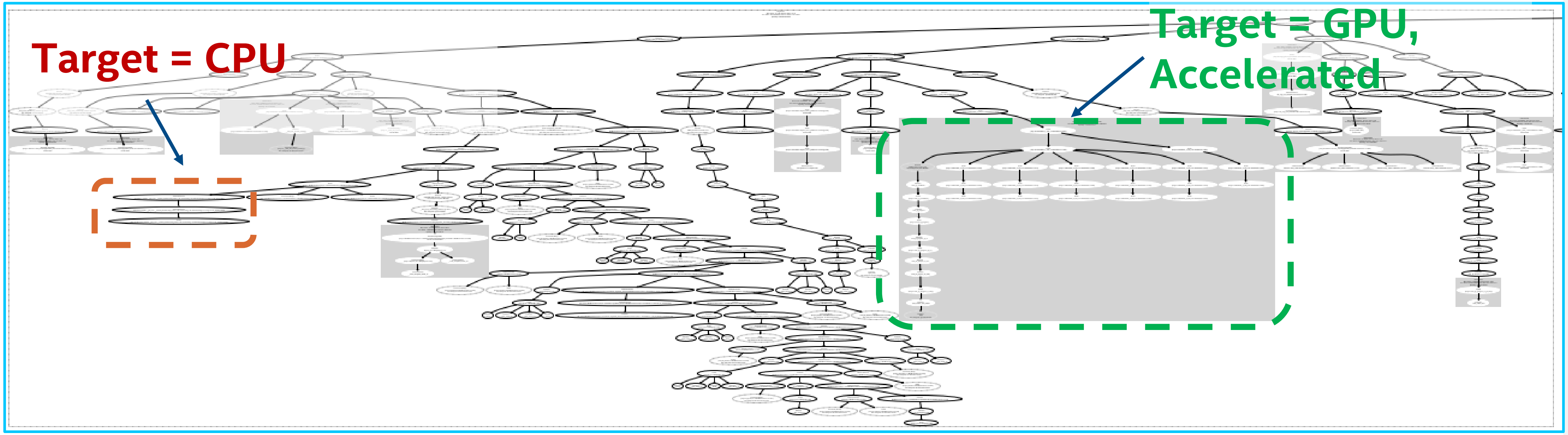


Loop/Function	Measured »	Basic Estimated Metrics		»	Estimated Bounded By	»	Estimated Data Transfer With Reuse »
	Time	Speed-Up	Time	Offload Summary	Throughput	Taxes With Reuse	
▼ Total	211.4ms						
▼ RtlUserThreadStart	211.4ms						
▼ BaseThreadInitThunk	211.4ms						
▼ _scrt_common_main_seh	211.4ms						
▼ main	211.4ms						
▼ serial_mandelbrot	171.8ms						
▼ [loop in serial_mandelbrot at mandelbrot.cpp:56]	171.8ms	13.945x	12.3ms	Offloaded	Compute 12.3ms DRAM BW < 0.1ms	Launch Tax < 0.1ms All Taxes < 0.1ms	Read 0B Write 2.1MB
▼ [loop in serial_mandelbrot at mandelbrot.cpp:57]	171.8ms		171.8...		Compute 12.2ms DRAM BW < 0.1ms		
[loop in serial_mandelbrot at mandelbrot.cpp:69]	171.8ms		171.8...		Compute 12.1ms L3 BW 0ms		

Line	Source	Is Offloaded	Speed-Up	Time
52	`_mm_malloc(width * height * sizeof(unsig`			
53				
54	`// Traverse the sample space in equally spac`			
55	`// samples`			
56	`for (int j = 0; j < height; ++j) {`	Yes	13.945x	12.3ms
57	`for (int i = 0; i < width; ++i) {`			
58	`double z_real = x0 + i * xstep;`			

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

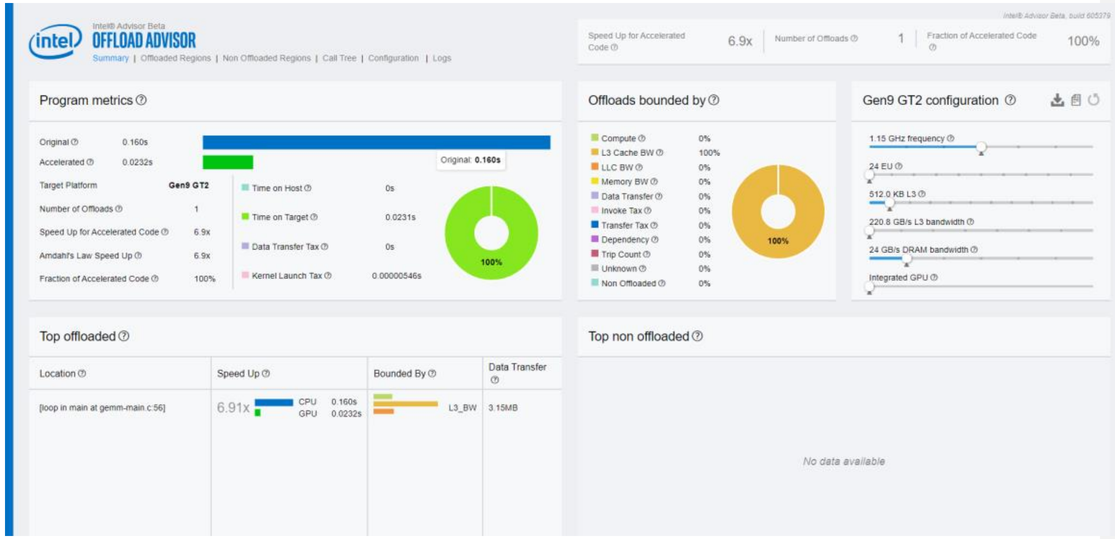
- Performance estimation steps:
  A. Profiling
  B. Performance modelling

- 3 different approaches to get estimation:
  - run_oa.py (both A and B), most convenient
  - collect.py (A) + analyze.py (B)
  - advixe-cl (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



intel
Intel® Advisor Beta
OFFLOAD ADVISOR
Summary | Offloaded Regions | Non Offloaded Regions | Call Tree | Configuration | Logs
Intel® Advisor Beta, build 605379
Original: 0.160s
100%
100%

Speed Up for Accelerated Code ⓘ	6.9x	Number of Offloads ⓘ	1	Fraction of Accelerated Code ⓘ	100%

Program metrics ⓘ

Original ⓘ	0.160s
Accelerated ⓘ	0.0232s
Target Platform	Gen9 GT2
Number of Offloads ⓘ	1
Speed Up for Accelerated Code ⓘ	6.9x
Amdahl's Law Speed Up ⓘ	6.9x
Fraction of Accelerated Code ⓘ	100%

Time on Host ⓘ	0s
Time on Target ⓘ	0.0231s
Data Transfer Tax ⓘ	0s
Kernel Launch Tax ⓘ	0.00000546s

Offloads bounded by ⓘ

Compute ⓘ	0%
L3 Cache BW ⓘ	100%
LLC BW ⓘ	0%
Memory BW ⓘ	0%
Data Transfer ⓘ	0%
Invoke Tax ⓘ	0%
Transfer Tax ⓘ	0%
Dependency ⓘ	0%
Trip Count ⓘ	0%
Unknown ⓘ	0%
Non Offloaded ⓘ	0%

Gen9 GT2 configuration ⓘ

1.15 GHz frequency ⓘ
24 EU ⓘ
512.0 KB L3 ⓘ
220.8 GB/s L3 bandwidth ⓘ
24 GB/s DRAM bandwidth ⓘ
Integrated GPU ⓘ

Top offloaded ⓘ

Location ⓘ	Speed Up ⓘ	Bounded By ⓘ	Data Transfer ⓘ
[loop in main at gemm-main.c:56]	6.91x CPU 0.160s GPU 0.0232s	L3_BW	3.15MB

Top non offloaded ⓘ

No data available

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`

- Run the data collection

  ```
  advixe-python $APM/collect.py advisor_project --config gen9 -- <app> [app_options]
  ```

  Analyze for a specific GPU config

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

- Run the performance modelling

  ```
  advixe-python $APM/analyze.py advisor_project --config gen9 --out-dir proj_results
  ```

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

- Alternatives: `run_oa.py` or `advixe-cl + analyze-py`

How to Run Offload Modeling

- Run Survey analysis to get baseline performance data

```
advisor --collect=survey --stackwalk-mode=online --static-instruction-mix
--project-dir=<my_project_dir> --search-dir sym:r=<my_symbols_dir>
--search-dir bin:r=<my_bin_dir> --search-dir src:r=<my_source_dir>
-- ./myapp [app_parameters]
```

Analyze stacks during collection

Statically calculate the number of instructions

- Run Trip Counts and FLOP analysis to get call count data and model cache for Gen9 GT2 GPU

```
advisor --collect=tripcounts --flop --stacks --enable-cache-simulation
--data-transfer=light --target-device=gen9_gt2
--project-dir=<my_project_dir> --search-dir sym:r=<my_symbols_dir>
--search-dir bin:r=<my_bin_dir> --search-dir src:r=<my_source_dir>
-- ./myapp [app_parameters]
```

Model CPU cache behavior

Model data transfer between host and device memory

Analyze for a specific GPU configuration

- Model performance on Gen9 GT2 GPU

```
advisor --collect=projection --config=gen9_gt2 --no-assume-dependencies
--project-dir=<my_project_dir>
```

Assume loop has no dependencies

Offload Modeling Resources

- User guide
  https://software.intel.com/content/www/us/en/develop/documentation/advisor-user-guide/top/design-for-gpu-offload/offload-modeling-perspective.html
- Cookbook recipes
  https://software.intel.com/content/www/us/en/develop/documentation/advisor-cookbook/top/design-and-optimize-application-with-offload-advisor.html
  https://software.intel.com/content/www/us/en/develop/documentation/advisor-cookbook/top/model-cpp-application-performance-on-a-target-gpu.html
- More user resources
  https://software.intel.com/content/www/us/en/develop/articles/offload-modeling-resources-for-intel-advisor-users.html