Omnitools: Performance Analysis Tools for AMD GPUs

Presenter: Samuel Antao

Contributors: Gina Sitaraman, Suyash Tandon, George Markomanolis, Jonathan Madsen, Austin Ellis, Bob Robey, Xiaomin Lu, Noah Wolfe

LUMI Training Course
May 30 – June 2, 2023

Logistics

- Access to LUMI: ssh username@lumi.csc.fi
- Project account: project_465000524
- Slides: /project/project_465000524/slides/AMD/
- Hackmd: https://hackmd.io/@gmarkoma/cug2023_tutorial_omnitools
  https://hackmd.io/@gmarkoma/lumi_training_ee

AMD
together we advance_

AMD
together we advance_

Profiling

[AMD Official Use Only - General]

Background – AMD Profilers

ROC-profiler (rocprof)

Hardware Counters
- Raw collection of GPU counters and traces
- Counter collection with user input files
- Counter results printed to a CSV

Traces and timelines
- Trace collection support for
  - CPU copy
  - HIP API
  - HSA API
  - GPU Kernels

Visualisation
- Traces visualized with Perfetto

	A	B	C	D	E
1	Name	Calls	TotalDura	AverageN:	Percentage
2	hipMemcpyAsync	99	3.22E+10	3.25E+08	44.14872
3	hipEventSynchronize	330	2.42E+10	73394557	33.225
4	hipMemsetAsync	87	7.76E+09	89232696	10.64953
5	hipHostMalloc	9	5.41E+09	6.01E+08	7.415198
6	hipDeviceSynchronize	28	1.32E+09	47006288	1.805515
7	hipHostFree	17	1.05E+09	61534688	1.435014
8	hipMemcpy	41	8.11E+08	19791876	1.113161
9	hipLaunchKernel	1856	58082083	31294	0.079676
10	hipStreamCreate	2	46380834	23190417	0.063625
11	hipMemset	2	18847246	9423623	0.025854
12	hipStreamDestroy	2	15183338	7591669	0.020828
13	hipFree	38	8269713	217624	0.011344
14	hipEventRecord	330	2520035	7636	0.003457
15	hipMalloc	30	1484804	49493	0.002037
16	__hipPopCallConfigura	1856	229159	123	0.000314
17	__hipPushCallConfigur	1856	224177	120	0.000308
18	hipGetLastError	1494	100458	67	0.000138
19	hipEventCreate	330	76675	232	0.000105
20	hipEventDestroy	330	64671	195	8.87E-05
21	hipGetDevicePropertie	47	51808	1102	7.11E-05
22	hipGetDevice	64	11611	181	1.59E-05
23	hipSetDevice	1	401	401	5.50E-07
24	hipGetDeviceCount	1	220	220	3.02E-07

Omnitrace

Trace collection
- Comprehensive trace collection
  - CPU
  - GPU

Supports
- CPU copy
- HIP API
- HSA API
- GPU Kernels
- OpenMP®
- MPI
- Kokkos
- p-threads
- multi-GPU

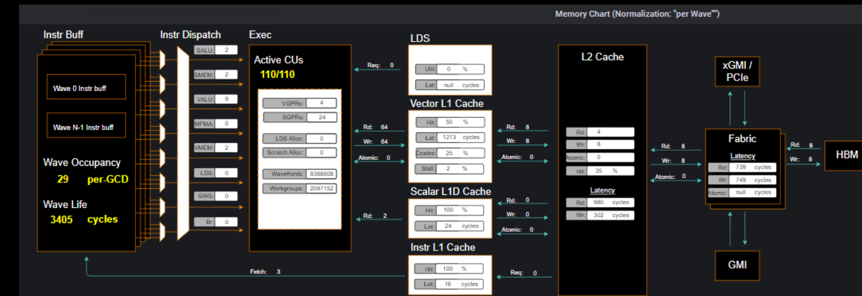
Visualisation
- Traces visualized with Perfetto

Omniperf

Performance Analysis
- Automated collection of hardware counters
  - Analysis
  - Visualisation

Supports
- Speed of Light
- Memory chart
- Rooflines
- Kernel comparison

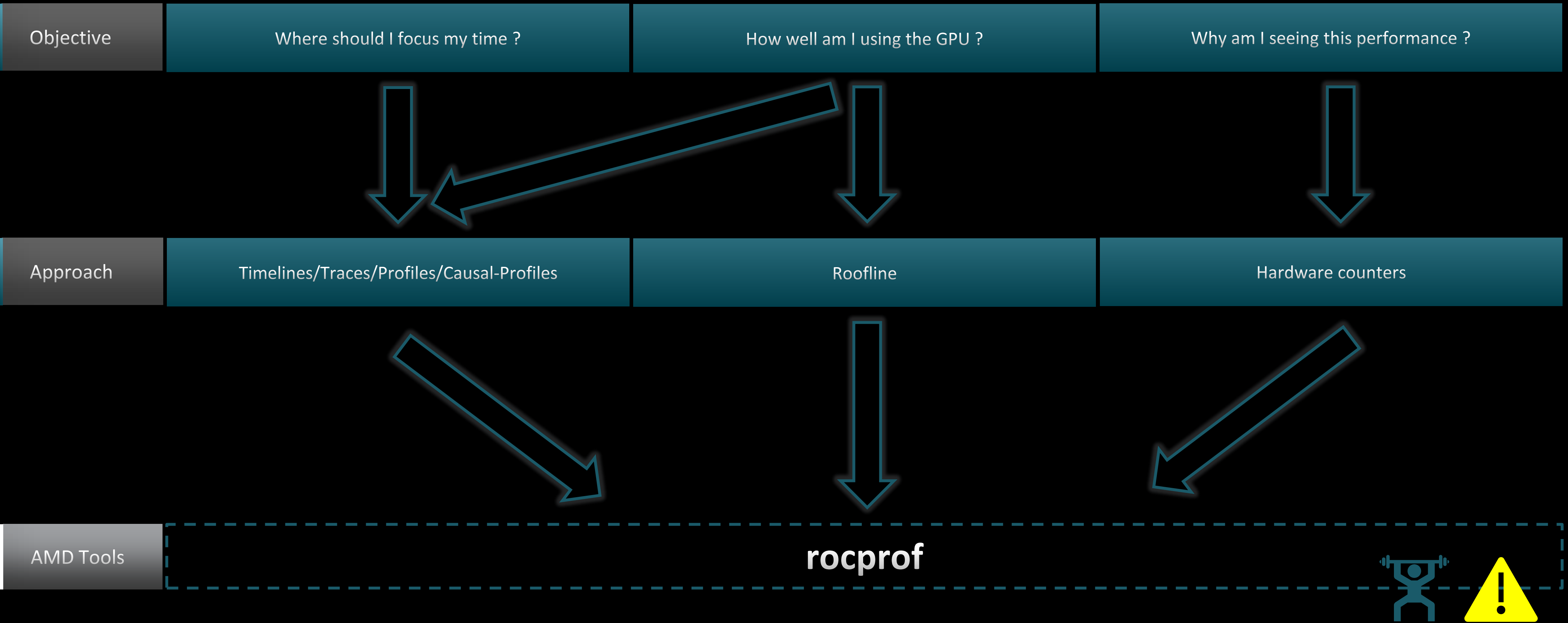
Visualisation
- With Grafana or standalone GUI



AMD
together we advance_

[AMD Official Use Only - General]

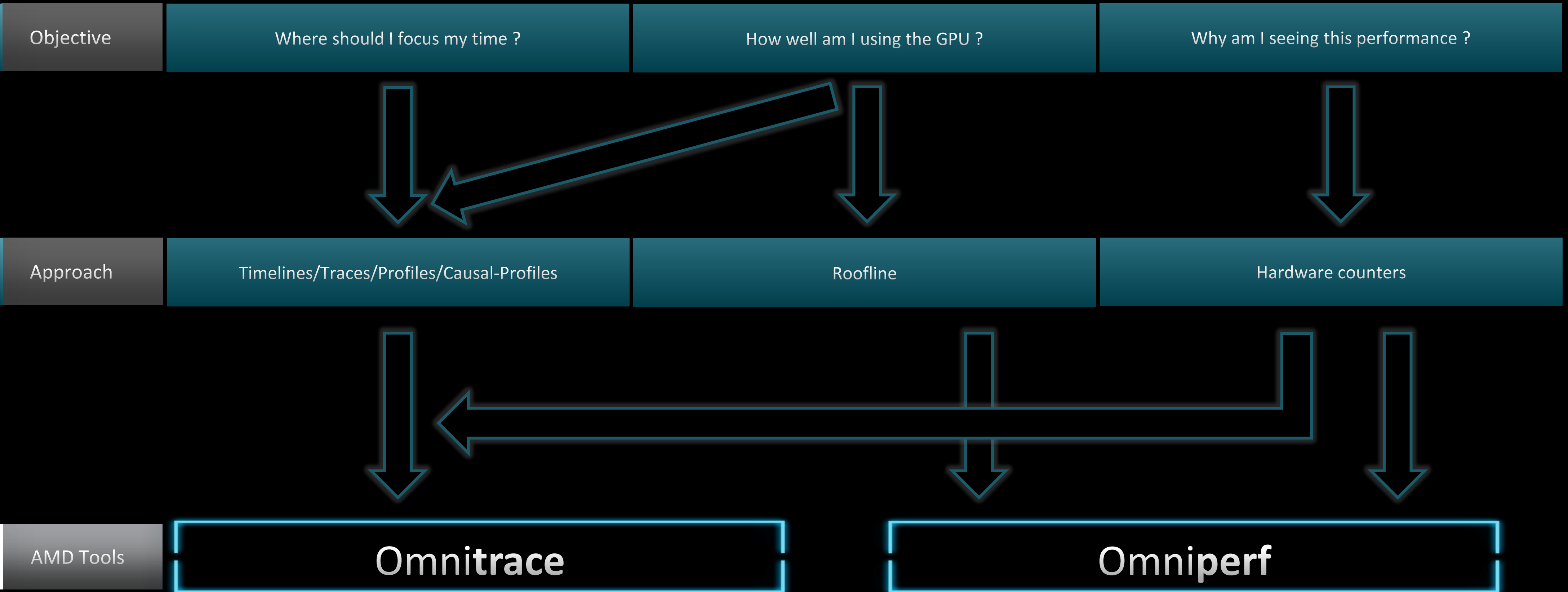
Background – AMD Profilers



AMD
together we advance_

[AMD Official Use Only - General]

Background – AMD Profilers



AMD
together we advance_

OmniTrace

[AMD Official Use Only - General]

Omnitrace: Application Profiling, Tracing, and Analysis

AMD Research Tool	Repository: https://github.com/AMDResearch/omnitrace					
	Not part of ROCm stack					
Language Support	C/C++	Fortran	Python	OpenCL ™		
Data Collection Modes	Dynamic instrumentation	Statistical/process sampling	Causal Profiling			
Data Analysis	High-level summary	Comprehensive trace	Critical trace analysis			
Parallelism Support	MPI	OpenMP®	Pthreads	HIP	HSA	Kokkos
GPU Metrics	HW counters	HSA API	HIP API	HIP trace	HSA trace	Memory & thermal
CPU Metrics	HW counters	Timing metrics	Memory access	Network	I/O	more…

Refer to current documentation for recent updates

[AMD Official Use Only - General]

Installation (if required)

To use pre-built binaries, select the version that matches your operating system, ROCm version, etc.

Select OpenSuse operating system for HPE/AMD system:

omnitrace-1.7.4-opensuse-15.4-ROCm-50400-PAPI-OMPT-Python3.sh

There are .rpm and .deb files for installation also. In future versions, binary installers for RHEL also available.

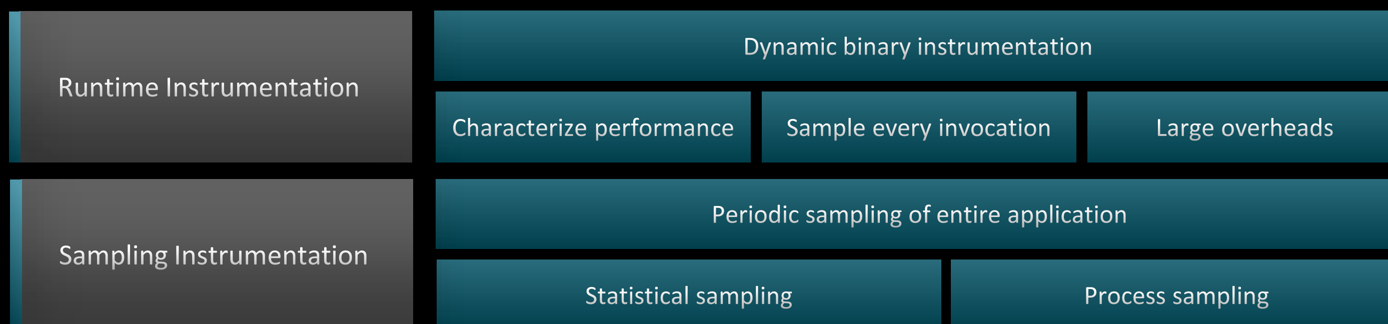
Full documentation: https://amdresearch.github.io/omnitrace/

```
export OMNITRACE_VERSION=latest
export ROCM_VERSION=5.4.3
export OMNITRACE_INSTALL_DIR=</path/to/your/omnitrace/install>
wget https://github.com/AMDResearch/omnitrace/releases/${OMNITRACE_VERSION}/download/omnitrace-install.py
python3 omnitrace-install.py -p ${OMNITRACE_INSTALL_DIR} --rocm ${ROCM_VERSION}

Set up environment:
source ${OMNITRACE_INSTALL_DIR}/share/omnitrace/setup-env.sh
```

Note: If installing from source, remember to clone the omnitrace repo recursively

AMD
together we advance_

Omnitrace instrumentation Modes



Basic command-line syntax:

```
$ omnitrace [omnitrace-options] -- <CMD> <ARGS>
```

For more information or help use -h/--help/? flags:

```
$ omnitrace -h
```

Can also execute on systems using a job scheduler. For example, with SLURM, an interactive session can be used as:

```
$ srun [options] omnitrace [omnitrace-options] -- <CMD> <ARGS>
```

For problems, create an issue here: https://github.com/AMDResearch/omnitrace/issues
Documentation: https://amdresearch.github.io/omnitrace/

AMD
together we advance_

[AMD Official Use Only - General]

Omnitrace Configuration

```
$ omnitrace-avail --categories [options]
```

Get more information about run-time settings, data collection capabilities, and available hardware counters. For more information or help use -h/--help flags:

```
$ omnitrace-avail -h
```

Collect information for omnitrace-related settings using shorthand -c for --categories :

```
$ omnitrace-avail –c perfetto
```

```
$ omnitrace-avail -c perfetto
|-------------------------------------|-----------------|----------------------------------------------------------------------|
|        ENVIRONMENT VARIABLE         |      VALUE      |                              CATEGORIES                              |
|-------------------------------------|-----------------|----------------------------------------------------------------------|
| OMNITRACE_PERFETTO_BACKEND          |    inprocess    | custom, libomnitrace, omnitrace, perfetto                            |
| OMNITRACE_PERFETTO_BUFFER_SIZE_KB   |     1024000     | custom, data, libomnitrace, omnitrace, perfetto                      |
| OMNITRACE_PERFETTO_FILL_POLICY      |     discard     | custom, data, libomnitrace, omnitrace, perfetto                      |
| OMNITRACE_TRACE_DELAY               |        0        | custom, libomnitrace, omnitrace, perfetto, profile, timemory, trace |
| OMNITRACE_TRACE_DURATION            |        0        | custom, libomnitrace, omnitrace, perfetto, profile, timemory, trace |
| OMNITRACE_TRACE_PERIODS             |                 | custom, libomnitrace, omnitrace, perfetto, profile, timemory, trace |
| OMNITRACE_TRACE_PERIOD_CLOCK_ID     | CLOCK_REALTIME  | custom, libomnitrace, omnitrace, perfetto, profile, timemory, trace |
| OMNITRACE_USE_PERFETTO              |      true       | backend, custom, libomnitrace, omnitrace, perfetto                   |
|-------------------------------------|-----------------|----------------------------------------------------------------------|
```

Shows all runtime settings that may be tuned for perfetto

AMD
together we advance_

Omnitrace Configuration

```
$ omnitrace-avail --categories [options]
```

Get more information about run-time settings, data collection capabilities, and available hardware counters. For more information or help use -h/--help/? flags:

```
$ omnitrace-avail -h
```

Collect information for omnitrace-related settings using shorthand -c for --categories :

```
$ omnitrace-avail -c omnitrace
```

For brief description, use the options:

```
$ omnitrace-avail -bd
```

ENVIRONMENT VARIABLE	DESCRIPTION
OMNITRACE_CAUSAL_BINARY_EXCLUDE	Excludes binaries matching the list of provided regexes from causal experiments (separated by tab, sem...
OMNITRACE_CAUSAL_BINARY_SCOPE	Limits causal experiments to the binaries matching the provided list of regular expressions (separated...
OMNITRACE_CAUSAL_DELAY	Length of time to wait (in seconds) before starting the first causal experiment
OMNITRACE_CAUSAL_DURATION	Length of time to perform causal experimentation (in seconds) after the first experiment has started. ...
OMNITRACE_CAUSAL_FUNCTION_EXCLUDE	Excludes functions matching the list of provided regexes from causal experiments (separated by tab, se...
OMNITRACE_CAUSAL_FUNCTION_SCOPE	List of <function> regex entries for causal profiling (separated by tab, semi-colon, and/or quotes (si...
OMNITRACE_CAUSAL_RANDOM_SEED	Seed for random number generator which selects speedups and experiments -- please note that the lines ...
OMNITRACE_CAUSAL_SOURCE_EXCLUDE	Excludes source files or source file + lineno pair (i.e. <file> or <file>:<line>) matching the list of...
OMNITRACE_CAUSAL_SOURCE_SCOPE	Limits causal experiments to the source files or source file + lineno pair (i.e. <file> or <file>:<lin...
OMNITRACE_CONFIG_FILE	Configuration file for omnitrace
OMNITRACE_CRITICAL_TRACE	Enable generation of the critical trace
OMNITRACE_ENABLED	Activation state of timemory
OMNITRACE_OUTPUT_PATH	Explicitly specify the output folder for results
OMNITRACE_OUTPUT_PREFIX	Explicitly specify a prefix for all output files
OMNITRACE_PAPI_EVENTS	PAPI presets and events to collect (see also: papi_avail)
OMNITRACE_PERFETTO_BACKEND	Specify the perfetto backend to activate. Options are: 'inprocess', 'system', or 'all'
OMNITRACE_PERFETTO_BUFFER_SIZE_KB	Size of perfetto buffer (in KB)
OMNITRACE_PERFETTO_FILL_POLICY	Behavior when perfetto buffer is full. 'discard' will ignore new entries, 'ring_buffer' will overwrite...
OMNITRACE_PROCESS_SAMPLING_DURATION	If > 0.0, time (in seconds) to sample before stopping. If less than zero, uses OMNITRACE_SAMPLING_DURA...
OMNITRACE_PROCESS_SAMPLING_FREQ	Number of measurements per second when OMNITTRACE_USE_PROCESS_SAMPLING=ON. If set to zero, uses OMNITR...
OMNITRACE_ROCM_EVENTS	ROCm hardware counters. Use ':device=N' syntax to specify collection on device number N, e.g. ':device...
OMNITRACE_SAMPLING_CPUS	CPUs to collect frequency information for. Values should be separated by commas and can be explicit or...
OMNITRACE_SAMPLING_DELAY	Time (in seconds) to wait before the first sampling signal is delivered, increasing this value can fix...
OMNITRACE_SAMPLING_DURATION	If > 0.0, time (in seconds) to sample before stopping
OMNITRACE_SAMPLING_FREQ	Number of software interrupts per second when OMNITRACE_USE_SAMPLING=ON
OMNITRACE_SAMPLING_GPUS	Devices to query when OMNITRACE_USE_ROCM_SMI=ON. Values should be separated by commas and can be expli...

Create a config file

Create a config file in $HOME:

```
$ omnitrace-avail -G $HOME/.omnitrace.cfg
```

To add description of all variables and settings, use:

```
$ omnitrace-avail -G $HOME/.omnitrace.cfg --all
```

Modify the config file $HOME/.omnitrace.cfg as desired to enable and change settings:

```
<snip>
OMNITRACE_USE_PERFETTO          = true
OMNITRACE_USE_TIMEMORY          = true
OMNITRACE_USE_SAMPLING          = false
OMNITRACE_USE_ROCTRACER         = true
OMNITRACE_USE_ROCM_SMI          = true
OMNITRACE_USE_KOKKOSP           = false
OMNITRACE_USE_CAUSAL            = false
OMNITRACE_USE_MPIP              = true
OMNITRACE_USE_PID               = true
OMNITRACE_USE_ROCPROFILER       = true
OMNITRACE_USE_ROCTX
<snip>
```

Contents of the config file

Declare which config file to use by setting the environment:

```
$ export OMNITRACE_CONFIG_FILE=/path-to/.omnitrace.cfg
```


AMD
together we advance_

Dynamic Instrumentation

Runtime Instrumentation

[AMD Official Use Only - General]

Dynamic Instrumentation – Jacobi Example

Clone jacobi example:

```
$ git clone https://github.com/amd/HPCTrainingExamples.git
$ cd HPCTrainingExamples/HIP/jacobi
```

Requires ROCm and MPI install, compile:

```
$ make
```

Run the non-instrumented code on a single GPU as:

```
 $ time mpirun -np 1 ./Jacobi_hip -g 1 1
real     0m2.115s
```

Dynamic instrumentation

```
 $ time mpirun -np 1 omnitrace-instrument -- ./Jacobi_hip
-g 1 1

 real 1m45.742s
```

Extra time is the overhead of dyninst reading every binary that is loaded, not overhead of omnitrace during app execution

Parsing libraries

```
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libutil-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libz.so.1.2.11'...
[omnitrace][exe] [internal] binary info processing required 0.322 sec and 70.724 MB
[omnitrace][exe] Processing 72 modules...
[omnitrace][exe] Processing 72 modules... Done (0.101 sec, 12.084 MB)
[omnitrace][exe] Found 'MPI_Init' in '/home/ssitaram/git/HPCTrainingExamples/HIP/jacobi/Jacobi_hip'. Enabling MPI support
[omnitrace][exe] Finding instrumentation functions...
[omnitrace][exe]    2 instrumented funcs in ../../orte/orted/orted_submit.c
[omnitrace][exe]    1 instrumented funcs in libamd_comgr.so.2.4.50403
[omnitrace][exe]   15 instrumented funcs in libamdhip64.so.5.4.50403
[omnitrace][exe]    1 instrumented funcs in libm-2.28.so
[omnitrace][exe]   10 instrumented funcs in libmpi.so.40.20.3
[omnitrace][exe]    8 instrumented funcs in libopen-pal.so.40.20.3
[omnitrace][exe]   17 instrumented funcs in libopen-rte.so.40.20.3
[omnitrace][exe]    2 instrumented funcs in libtinfo.so.5.9
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-14_17.24/instrumentation/available.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-14_17.24/instrumentation/available.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-14_17.24/instrumentation/instrumented.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-14_17.24/instrumentation/instrumented.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-14_17.24/instrumentation/excluded.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-14_17.24/instrumentation/excluded.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-14_17.24/instrumentation/overlapping.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-14_17.24/instrumentation/overlapping.txt'... Done
[omnitrace][exe] Executing...
[omnitrace][1649192][omnitrace_init_tooling] Instrumentation mode: Trace

    omnitrace v1.8.0
```

Functions instrumented

Outputs that will be created

AMD
together we advance_

[AMD Official Use Only - General]

Dynamic Instrumentation – Jacobi Example

Clone jacobi example:

```
$ git clone https://github.com/amd/HPCTrainingExamples.git
$ cd HPCTrainingExamples/HIP/jacobi
```

Requires ROCm and MPI install, compile:

```
$ make
```

Run the non-instrumented code on a single GPU as:

```
$ time mpirun -np 1 ./Jacobi_hip -g 1 1
real    0m2.115s
```

Dynamic instrumentation

```
$ time mpirun -np 1 omnitrace-instrument -- ./Jacobi_hip
-g 1 1

real 1m45.742s
```

Available functions to instrument:

```
$ mpirun -np 1 omnitrace-instrument -v 1 --simulate --
print-available functions -- ./Jacobi_hip -g 1 1
```

Here, -v gives a verbose output from omnitrace

The simulate flag does not run the executable, but only demonstrates the available functions

```
[available] HaloExchange.cpp:
[available]    [HaloExchange.cold.21][14]
[available]    [HaloExchange][1267]
[available]    [_GLOBAL__sub_I_HaloExchange.cpp][8]

[available] Input.cpp:
[available]    [ExtractNumber][19]
[available]    [FindAndClearArgument][38]
[available]    [ParseCommandLineArguments][206]
[available]    [PrintUsage][12]

[available] JacobiIteration.cpp:
[available]    [JacobiIteration][71]

[available] JacobiMain.cpp:
[available]    [main.cold.0][5]
[available]    [main][35]

[available] JacobiRun.cpp:
[available]    [Jacobi_t::Run][155]

[available] JacobiSetup.cpp:
[available]    [FormatNumber][53]
[available]    [Jacobi_t::ApplyTopology][234]
[available]    [Jacobi_t::CreateMesh][459]
[available]    [Jacobi_t::InitializeData][552]
[available]    [Jacobi_t::Jacobi_t.cold.30][15]
[available]    [Jacobi_t::Jacobi_t][1043]
[available]    [Jacobi_t::PrintResults][107]
[available]    [Jacobi_t::~Jacobi_t][167]
[available]    [PrintPerfCounter][34]
[available]    [_GLOBAL__sub_I_JacobiSetup.cpp][8]
[available]    [std::__cxx11::basic_stringbuf<char, std::char_traits<char>, std::allocator
<char> >::~basic_stringbuf][16]
[available]    [std::__cxx11::basic_stringbuf<char, std::char_traits<char>, std::allocator
<char> >::~basic_stringbuf][18]
```

Functions found in each module detected by omnitrace

AMD
together we advance_

[AMD Official Use Only - General]

Dynamic Instrumentation – Jacobi Example

Clone jacobi example:

```
$ git clone https://github.com/amd/HPCTrainingExamples.git
$ cd HPCTrainingExamples/HIP/jacobi
```

Requires ROCm and MPI install, compile:

```
$ make
```

Run the non-instrumented code on a single GPU as:

```
 $ time mpirun -np 1 ./Jacobi_hip -g 1 1
real    0m2.115s
```

Dynamic instrumentation

```
 $ time mpirun -np 1 omnitrace-instrument -- ./Jacobi_hip
-g 1 1

 real 1m45.742s
```

Available functions to instrument:

```
 $ mpirun -np 1 omnitrace-instrument -v 1 --simulate --
print-available functions -- ./Jacobi_hip -g 1 1
```

Custom include/exclude functions* with -I or -E, resp. For e.g:

```
 $ mpirun -np 1 omnitrace-instrument -v 1 -I
'Jacobi_t::Run' 'JacobiIteration' -- ./Jacobi_hip -g 1 1
```

Include two functions to instrument

```
[omnitrace][exe] [internal] parsing library: '/opt/rocm-5.4.3/lib/librocm_smi64.so.5.0.50403'...
[omnitrace][exe] [internal] parsing library: '/opt/rocm-5.4.3/lib/librocmtools.so.1.5.0'...
[omnitrace][exe] [internal] parsing library: '/opt/rocm-5.4.3/lib/librocprofiler64.so.1.0.50403'...
[omnitrace][exe] [internal] parsing library: '/opt/rocm-5.4.3/lib/libroctracer64.so.4.1.0'...
[omnitrace][exe] [internal] parsing library: '/opt/rocm-5.4.3/lib/libroctx64.so.4.1.0'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/libomnitrace-dl.so.1.8.0'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/libomnitrace-rt.so.11.0.1'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/libomnitrace-user.so.1.8.0'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/omnitrace/libcommon.so.11.0.1'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/omnitrace/libdw-0.182.so'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/omnitrace/libelf-0.182.so'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/omnitrace/libgotcha.so.2.0.2'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/omnitrace/libpfm.so.4.11.1'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/omnitrace/libtbb.so.2'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/omnitrace/libtbbmalloc.so.2'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/omnitrace/libtbbmalloc_proxy.so.2'...
[omnitrace][exe] [internal] parsing library: '/share/contrib-modules/omnitrace/omnitrace1.8.0/lib/omnitrace/libunwind.so.99.0.0'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/ld-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libBrokenLocale-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libanl-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libc-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libcrypt.so.1.1.0'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libdl-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libgcc_s-8-20210514.so.1'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libnss_compat-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libnss_dns-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libnss_files-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libpthread-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libresolv-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/librt-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libstdc++.so.6.0.25'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libthread_db-1.0.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libutil-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libz.so.1.2.11'...
[omnitrace][exe] [internal] binary info processing required 0.257 sec and 66.740 MB
[omnitrace][exe] Processing 72 modules...
[omnitrace][exe] Processing 72 modules... Done (0.089 sec, 11.080 MB)
[omnitrace][exe] Found 'MPI_Init' in '/home/ssitaram/git/HPCTrainingExamples/HIP/jacobi/Jacobi_hip'. Enabling MPI support...
[omnitrace][exe] Finding instrumentation functions...
[omnitrace][exe]    1 instrumented funcs in JacobiIteration.cpp
[omnitrace][exe]    1 instrumented funcs in JacobiRun.cpp
[omnitrace][exe]    1 instrumented funcs in Jacobi_hip
[omnitrace][exe]    1 instrumented funcs in libamdhip64.so.5.4.50403
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_12.40/instrumentation/available.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_12.40/instrumentation/available.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_12.40/instrumentation/instrumented.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_12.40/instrumentation/instrumented.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_12.40/instrumentation/excluded.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_12.40/instrumentation/excluded.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_12.40/instrumentation/overlapping.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_12.40/instrumentation/overlapping.txt'... Done
```

Only these two functions are shown to be instrumented

AMD
together we advance_

Dynamic Instrumentation

Binary Rewrite

[AMD Official Use Only - General]

Binary Rewrite – Jacobi Example

Binary Rewrite

```
$ omnitrace-instrument [omnitrace-options] -o <new-name-of-exec> -- <CMD> <ARGS>
```

Generating a new executable/library with instrumentation built-in:

```
$ omnitrace-instrument -o Jacobi_hip.inst -- ./Jacobi_hip
```

This new binary will have instrumented functions

Subroutine Instrumentation

Default instrumentation is main function and functions of 1024 instructions and more (for CPU)

To instrument routines with 50 or more cycles, add option "-i 50" (more overhead)

```
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libgcc_s-8-20210514.so.1'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libnss_compat-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libnss_dns-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libnss_files-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libpthread-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libresolv-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/librt-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libstdc++.so.6.0.25'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libthread_db-1.0.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libutil-2.28.so'...
[omnitrace][exe] [internal] parsing library: '/usr/lib64/libz.so.1.2.11'...
[omnitrace][exe] [internal] binary info processing required 0.666 sec and 110.500 MB
[omnitrace][exe] Processing 9 modules...
[omnitrace][exe] Processing 9 modules... Done (0.001 sec, 0.000 MB)
[omnitrace][exe] Found 'MPI_Init' in '/home/ssitaram/git/HPCTrainingExamples/HIP/jacobi/Jacobi_hip'. Enabling MPI support...
[omnitrace][exe] Finding instrumentation functions...
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_12.57/instrumentation/available.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_12.57/instrumentation/available.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_12.57/instrumentation/instrumented.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_12.57/instrumentation/instrumented.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_12.57/instrumentation/excluded.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_12.57/instrumentation/excluded.txt'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_12.57/instrumentation/overlapping.json'... Done
[omnitrace][exe] Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_12.57/instrumentation/overlapping.txt'... Done
[omnitrace][exe]
[omnitrace][exe] The instrumented executable image is stored in '/home/ssitaram/git/HPCTrainingExamples/HIP/jacobi/Jacobi_hip.inst'
[omnitrace][exe] Getting linked libraries for /home/ssitaram/git/HPCTrainingExamples/HIP/jacobi/Jacobi_hip...
[omnitrace][exe] Consider instrumenting the relevant libraries...
[omnitrace][exe]
[omnitrace][exe]        /lib64/libgcc_s.so.1
[omnitrace][exe]        /lib64/libpthread.so.0
[omnitrace][exe]        /lib64/libm.so.6
[omnitrace][exe]        /lib64/librt.so.1
[omnitrace][exe]        /home/ssitaram/cp2k-hip/libs/install/openmpi/lib/libmpi.so.40
[omnitrace][exe]        /opt/rocm-5.4.3//lib/libroctx64.so.4
[omnitrace][exe]        /opt/rocm-5.4.3//lib/libroctracer64.so.4
[omnitrace][exe]        /opt/rocm-5.4.3/hip/lib/libamdhip64.so.5
[omnitrace][exe]        /lib64/libstdc++.so.6
[omnitrace][exe]        /lib64/libc.so.6
[omnitrace][exe]        /lib64/ld-linux-x86-64.so.2
```

Path to new instrumented binary

AMD
together we advance_

[AMD Official Use Only - General]

Binary Rewrite – Jacobi Example

Binary Rewrite

```
$ omnitrace-instrument [omnitrace-options] -o <new-name-of-exec> -- <CMD> <ARGS>
```

Generating a new /library with instrumentation built-in:

```
$ omnitrace-instrument -o Jacobi_hip.inst -- ./Jacobi_hip
```

Run the instrumented binary:

```
$ mpirun -np 1 omnitrace-run -- ./Jacobi_hip.inst -g 1 1
```

subroutine instrumentation

Default instrumentation is main function and functions of 1024 instructions and more (for CPU)

To instrument routines with 50 or more cycles, add option "`-i 50`" (more overhead)

Binary rewrite is recommended for runs with multiple ranks as omnitrace produces separate output files for each rank

```
[omnitrace][3624331][omnitrace_init_tooling] Instrumentation mode: Trace

    omnitrace v1.8.0
[953.765]       perfetto.cc:58656 Configured tracing session 1, #sources:1, duration:0 ms, #buffers:1, total buffer size:1024000 KB, total sessions:1, uid:0 session name: ""
Topology size: 1 x 1
Local domain size (current node): 4096 x 4096
[omnitrace][0][pid=3624331] MPI rank: 0 (0), MPI size: 1 (1)
Global domain size (all nodes): 4096 x 4096
Rank 0 selecting device 0 on host TheraC60
Starting Jacobi run.
Iteration:    0 - Residual: 0.022108
Iteration:  100 - Residual: 0.000625
Iteration:  200 - Residual: 0.000371
Iteration:  300 - Residual: 0.000274
Iteration:  400 - Residual: 0.000221
Iteration:  500 - Residual: 0.000187
Iteration:  600 - Residual: 0.000163
Iteration:  700 - Residual: 0.000145
Iteration:  800 - Residual: 0.000131
Iteration:  900 - Residual: 0.000120
Iteration: 1000 - Residual: 0.000111
Stopped after 1000 iterations with residue 0.000111
Total Jacobi run time: 1.5470 sec.
Measured lattice updates: 10.84 GLU/s (total), 10.84 GLU/s (per process)
Measured FLOPS: 184.36 GFLOPS (total), 184.36 GFLOPS (per process)
Measured device bandwidth: 1.04 TB/s (total), 1.04 TB/s (per process)

[omnitrace][3624331][0][omnitrace_finalize] finalizing...
[omnitrace][3624331][0][omnitrace_finalize]
[omnitrace][3624331][0][omnitrace_finalize] omnitrace/process/3624331 : 2.364423 sec wall_clock,  645.964 MB peak_rss,  388.739 MB page_rss, 4.330000 sec cpu_clock,  183.1 % cpu_util [laps: 1]
[omnitrace][3624331][0][omnitrace_finalize] omnitrace/process/3624331/thread/0 : 2.355893 sec wall_clock, 1.293230 sec thread_cpu_clock,   54.9 % thread_cpu_util,  645.964 MB peak_rss [laps: 1]
[omnitrace][3624331][0][omnitrace_finalize] omnitrace/process/3624331/thread/1 : 2.345084 sec wall_clock, 0.000261 sec thread_cpu_clock,    0.0 % thread_cpu_util,  642.676 MB peak_rss [laps: 1]
[omnitrace][3624331][0][omnitrace_finalize]
[omnitrace][3624331][0][omnitrace_finalize] Finalizing perfetto...
```

Generates traces for application run

AMD together we advance_

[AMD Official Use Only - General]

List of Instrumented GPU Functions

```
$ cat omnitrace-Jacobi_hip.inst-output/2023-03-15_13.57/roctracer-0.txt
```

ROCM TRACER (ACTIVITY API)

LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	% SELF
|0>>> pthread_create	1	0	roctracer	sec	0.000353	0.000353	0.0
|1>>> |_start_thread	1	1	roctracer	sec	2.344864	2.344864	100.0
|0>>> hipInit	1	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipGetDeviceCount	1	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipSetDevice	1	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipHostMalloc	3	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipMalloc	7	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipMemset	1	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipStreamCreate	2	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipMemcpy	1005	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> |_LocalLaplacianKernel(int, int, int, double, double, double const*, double*)	999	1	roctracer	sec	0.279368	0.000280	100.0
|0>>> |_HaloLaplacianKernel(int, int, int, double, double, double const*, double const*, double*)	990	1	roctracer	sec	0.014761	0.000015	100.0
|0>>> |_JacobiIterationKernel(int, double, double, double const*, double const*, double*, double*)	959	1	roctracer	sec	0.531156	0.000554	100.0
|0>>> |_NormKernel1(int, double, double, double const*, double*)	997	1	roctracer	sec	0.430196	0.000431	100.0
|0>>> |_NormKernel2(int, double const*, double*)	999	1	roctracer	sec	0.004342	0.000004	100.0
|0>>> hipEventCreate	2	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipLaunchKernel	5002	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> |_JacobiIterationKernel(int, double, double, double const*, double const*, double*, double*)	1	1	roctracer	sec	0.000552	0.000552	100.0
|0>>> |_NormKernel1(int, double, double, double const*, double*)	1	1	roctracer	sec	0.000425	0.000425	100.0
|0>>> hipDeviceSynchronize	1001	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> |_NormKernel1(int, double, double, double const*, double*)	2	1	roctracer	sec	0.000850	0.000425	100.0
|0>>> |_NormKernel2(int, double const*, double*)	1	1	roctracer	sec	0.000004	0.000004	100.0
|0>>> |_HaloLaplacianKernel(int, int, int, double, double, double const*, double const*, double*)	9	1	roctracer	sec	0.000133	0.000015	100.0
|0>>> |_JacobiIterationKernel(int, double, double, double const*, double const*, double*, double*)	40	1	roctracer	sec	0.022204	0.000555	100.0
|0>>> |_LocalLaplacianKernel(int, int, int, double, double, double const*, double*)	1	1	roctracer	sec	0.000281	0.000281	100.0
|0>>> hipEventRecord	2000	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipStreamSynchronize	2000	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipEventElapsedTime	1000	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> |_HaloLaplacianKernel(int, int, int, double, double, double const*, double const*, double*)	1	1	roctracer	sec	0.000015	0.000015	100.0
|0>>> hipFree	4	0	roctracer	sec	0.000000	0.000000	0.0
|0>>> hipHostFree	2	0	roctracer	sec	0.000000	0.000000	0.0

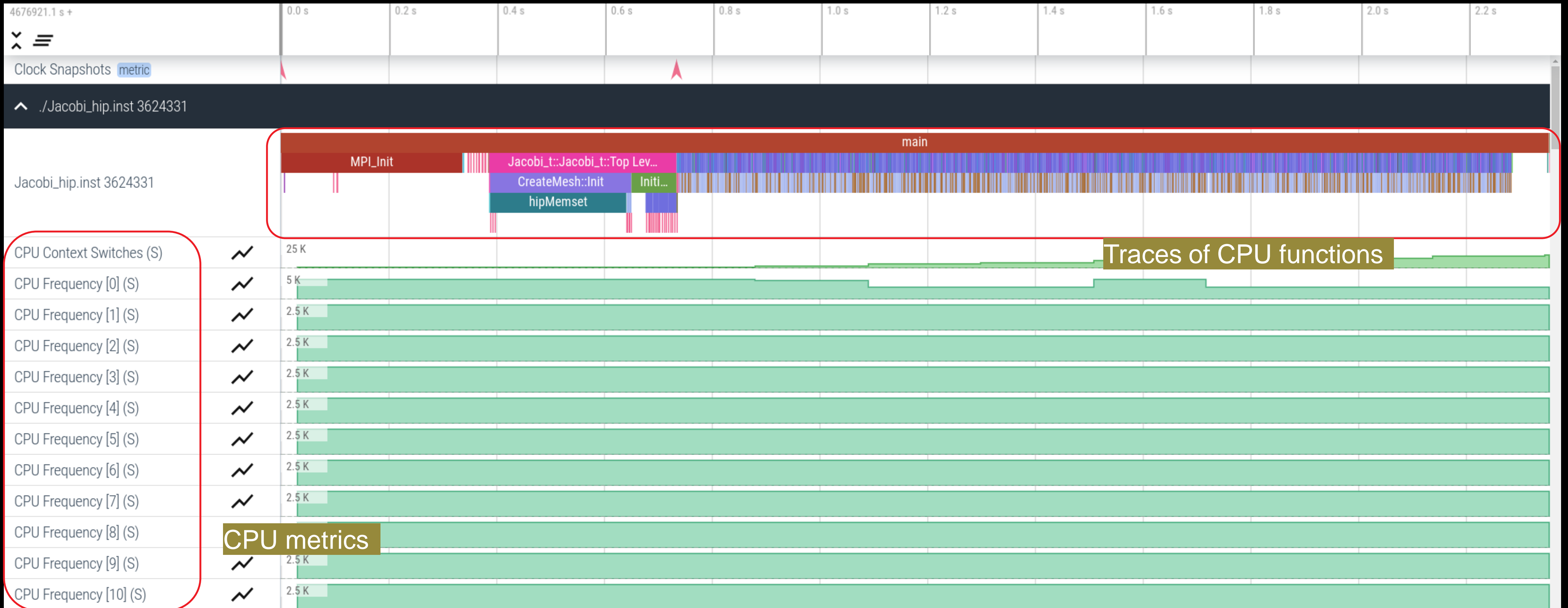
Roctracer-0.txt shows duration of HIP API calls and GPU kernels

AMD
together we advance_

Visualizing Trace

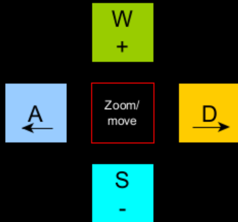
Use Perfetto

Copy `perfetto-trace-0.proto` to your laptop, go to https://ui.perfetto.dev/, Click "Open trace file", select perfetto-trace-0.proto



AMD
together we advance_

Visualizing Trace



Use Perfetto

Zoom in to investigate regions of interest

./Jacobi_hip.inst 3624331

Jacobi_hip.inst 3624331

main

MPI_All... hipDe... hipEven... hipEventRecord hipLaunchK... hipEve... Halo D2H:... MPI Exchange::Hal... Halo H2D::... hipLaunc... hipLau... hipLau... hipLa... hipMemcpy

hipS... MPI_Waita... hipSt...

Zoomed in

CPU Context Switches (S) 25 K

CPU Frequency [0] (S) 5 K

CPU Frequency [1] (S) 2.5 K

CPU Frequency [2] (S) 2.5 K

CPU Frequency [3] (S) 2.5 K

CPU Frequency [4] (S) 2.5 K

CPU Frequency [5] (S) 2.5 K

CPU Frequency [6] (S) 2.5 K

CPU Frequency [7] (S) 2.5 K

CPU Frequency [8] (S) 2.5 K

CPU Frequency [9] (S) 2.5 K

CPU Frequency [10] (S) 2.5 K

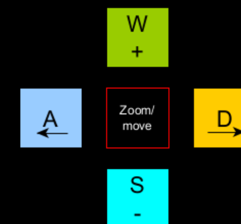
AMD
together we advance_

[AMD Official Use Only - General]

Visualizing Trace

Use Perfetto

Zoom in to investigate regions of interest



4676921.1 s +

0.75236 s 0.75238 s 0.75240 s 0.75242 s 0.75244 s 0.75246 s 0.75248 s 0.75250 s 0.75252 s 0.75254 s 0.75256 s 0.75258 s 0.75260 s

./Jacobi_hip.inst 3624331

HIP Activity Device 8, Queue 0

HIP Activity Device 8, Queue 1

Jacobi_hip.inst 3624331

CPU Context Switches (S)

CPU Frequency [0] (S)

CPU Kernel Time (S)

CPU Memory Usage (S)

CPU Page Faults (S)

CPU Peak Memory (S)

CPU User Time (S)

CPU Virtual Memory Usage (S)

GPU Busy [0] (S)

GPU Memory Usage [0] (S)

GPU Temperature [0] (S)

GPU Power [0] (S)

Flow Events

Marker

LocalLaplacianKernel(int, int, int, double, double, double const*, double*)

main

MPI_All... hipDe... hipEven... hipEventRecord hipLaunchK... hipEve... Halo D2H:... MPI Exchange::Hal... Halo H2D::... hipLaunc... hipLau... hipLau... hipLa... hipMemcpy

hipS... MPI_Waita... hipSt...

Select metrics of interest to view close together

5 K

5

0.75 K

50 K

0.75 K

2.5

50 K

100

1 K

GPU characteristics

0.25 K

AMD
together we advance_

Larger Traces with Perfetto

- There is a memory limit in the Chrome browser. There is a way to read in the trace for the browser before starting it up.

Linux

- curl -LO https://get.perfetto.dev/trace_processor
- chmod +x ./trace_processor
- ./trace_processor –httpd <path to trace file>
- Open up Chrome browser and go to https://ui.perfetto.dev
- When prompted, click on "Yes, use loaded trace"

Windows

- Open up https://get.perfetto.dev/trace_processor in a browser to download the python script
- py trace_processor --httpd <trace file>
  - You may need to download and install python on your windows system
- Open up Chrome browser and go to https://ui.perfetto.dev
- When prompted, click on "Yes, use loaded trace"

AMD
together we advance_

Hardware Counters

[AMD Official Use Only - General]

Hardware Counters – List All

```
$ mpirun –np 1 omnitrace-avail --all
```

Components, Categories

COMPONENT	AVAILABLE	VALUE_TYPE	STRING_IDS	FILENAME	DESCRIPTION	CATEGORY
allinea_map	false	void	"allinea", "allinea_map", "forge"		Controls the AllineaMAP sampler.	category::external, os::supports_linux, t...
caliper_marker	false	void	"cali", "caliper", "caliper_marker"		Generic forwarding of markers to Caliper ...	category::external, os::supports_unix, tp...
caliper_config	false	void	"caliper_config"		Caliper configuration manager.	category::external, os::supports_unix, tp...
caliper_loop_marker	false	void	"caliper_loop_marker"		Variant of caliper_marker with support fo...	category::external, os::supports_unix, tp...
cpu_clock	true	long	"cpu_clock"	cpu_clock	Total CPU time spent in both user- and ke...	project::timemory, category::timing, os::...
cpu_util	true	std::pair<long, long>	"cpu_util", "cpu_utilization"	cpu_util	Percentage of CPU-clock time divided by w...	project::timemory, category::timing, os::...
craypat_counters	false	std::vector<unsigned long, std::allocato...	"craypat_counters"	craypat_counters	Names and value of any counter events tha...	category::external, os::supports_linux, t...

Environment Variables

ENVIRONMENT VARIABLE	VALUE	DATA TYPE	DESCRIPTION	CATEGORIES
OMNITRACE_CAUSAL_BINARY_EXCLUDE		string	Excludes binaries matching the list of pr...	analysis, causal, custom, libomnitrace, o...
OMNITRACE_CAUSAL_BINARY_SCOPE	%MAIN%	string	Limits causal experiments to the binaries...	analysis, causal, custom, libomnitrace, o...
OMNITRACE_CAUSAL_DELAY	0	double	Length of time to wait (in seconds) befor...	analysis, causal, custom, libomnitrace, o...
OMNITRACE_CAUSAL_DURATION	0	double	Length of time to perform causal experime...	analysis, causal, custom, libomnitrace, o...
OMNITRACE_CAUSAL_FUNCTION_EXCLUDE		string	Excludes functions matching the list of p...	analysis, causal, custom, libomnitrace, o...
OMNITRACE_CAUSAL_FUNCTION_SCOPE		string	List of <function> regex entries for caus...	analysis, causal, custom, libomnitrace, o...
OMNITRACE_CAUSAL_RANDOM_SEED	0	unsigned long	Seed for random number generator which se...	analysis, causal, custom, libomnitrace, o...
OMNITRACE_CAUSAL_SOURCE_EXCLUDE		string	Excludes source files or source file + li...	analysis, causal, custom, libomnitrace, o...
OMNITRACE_CAUSAL_SOURCE_SCOPE		string	Limits causal experiments to the source f...	analysis, causal, custom, libomnitrace, o...

CPU Hardware Counters

HARDWARE COUNTER	AVAILABLE	DESCRIPTION
CPU		
PAPI_L1_DCM	true	Level 1 data cache misses
PAPI_L1_ICM	false	Level 1 instruction cache misses
PAPI_L2_DCM	true	Level 2 data cache misses
PAPI_L2_ICM	true	Level 2 instruction cache misses
PAPI_L3_DCM	false	Level 3 data cache misses
PAPI_L3_ICM	false	Level 3 instruction cache misses
PAPI_L1_TCM		Level 1 cache misses

HARDWARE COUNTER	AVAILABLE	DESCRIPTION
perf::CYCLES	true	PERF_COUNT_HW_CPU_CYCLES
perf::CYCLES:u=0	true	perf::CYCLES + monitor at user level
perf::CYCLES:k=0	true	perf::CYCLES + monitor at kernel level
perf::CYCLES:h=0	true	perf::CYCLES + monitor at hypervisor level
perf::CYCLES:period=0	true	perf::CYCLES + sampling period
perf::CYCLES:freq=0	true	perf::CYCLES + sampling frequency (Hz)
perf::CYCLES:precise=0	true	perf::CYCLES + precise event sampling
perf::CYCLES:excl=0	true	perf::CYCLES + exclusive access

GPU Hardware Counters

HARDWARE COUNTER	AVAILABLE	DESCRIPTION
TCC_NORMAL_WRITEBACK_sum:device=0	true	Number of writebacks due to requests that...
TCC_ALL_TC_OP_WB_WRITEBACK_sum:device=0	true	Number of writebacks due to all TC_OP wri...
TCC_NORMAL_EVICT_sum:device=0	true	Number of evictions due to requests that ...
TCC_ALL_TC_OP_INV_EVICT_sum:device=0	true	Number of evictions due to all TC_OP inva...
TCC_EA_RDREQ_DRAM_sum:device=0	true	Number of TCC/EA read requests (either 32...
TCC_EA_WRREQ_DRAM_sum:device=0	true	Number of TCC/EA write requests (either 3...
FETCH_SIZE:device=0	true	The total kilobytes fetched from the vide...
WRITE_SIZE:device=0	true	The total kilobytes written to the video ...
WRITE_REQ_32B:device=0	true	The total number of 32-byte effective mem...
GPUBusy:device=0	true	The percentage of time GPU was busy.
Wavefronts:device=0		Total wavefronts.
VALUInsts:device=0		The average number of vector ALU instruct...
SALUInsts:device=0	true	The average number of scalar ALU instruct...
SFetchInsts:device=0	true	The average number of scalar fetch instru...
GDSInsts:device=0	true	The average number of GDS read or GDS wri...
MemUnitBusy:device=0	true	The percentage of GPUTime the memory unit...
ALUStalledByLDS:device=0	true	The percentage of GPUTime ALU units are s...

A very small subset of the counters shown here

AMD
together we advance_

[AMD Official Use Only - General]

Commonly Used GPU Counters

Counter	Description
VALUUtilization	The percentage of ALUs active in a wave. Low VALUUtilization is likely due to high divergence or a poorly sized grid
VALUBusy	The percentage of GPUTime vector ALU instructions are processed. Can be thought of as something like compute utilization
FetchSize	The total kilobytes fetched from global memory
WriteSize	The total kilobytes written to global memory
L2CacheHit	The percentage of fetch, write, atomic, and other instructions that hit the data in L2 cache
MemUnitBusy	The percentage of GPUTime the memory unit is active. The result includes the stall time
MemUnitStalled	The percentage of GPUTime the memory unit is stalled
WriteUnitStalled	The percentage of GPUTime the write unit is stalled

Modify config file

Create a config file in $HOME:

```
$ omnitrace-avail -G $HOME/.omnitrace.cfg
```

Modify the config file $HOME/.omnitrace.cfg to add desired metrics and for concerned GPU#ID:

```
…
OMNITRACE_ROCM_EVENTS = GPUBusy:device=0,
Wavefronts:device=0, MemUnitBusy:device=0
…
```

To profile desired metrics for all participating GPUs:

```
…
OMNITRACE_ROCM_EVENTS = GPUBusy, Wavefronts,
MemUnitBusy
…
```

Full list at: https://github.com/ROCm-Developer-Tools/rocprofiler/blob/amd-master/test/tool/metrics.xml

AMD
together we advance_

[AMD Official Use Only - General]

Execution with Hardware Counters

(after modifying cfg file to set up OMNITRACE_ROCM_EVENTS with GPU metrics)

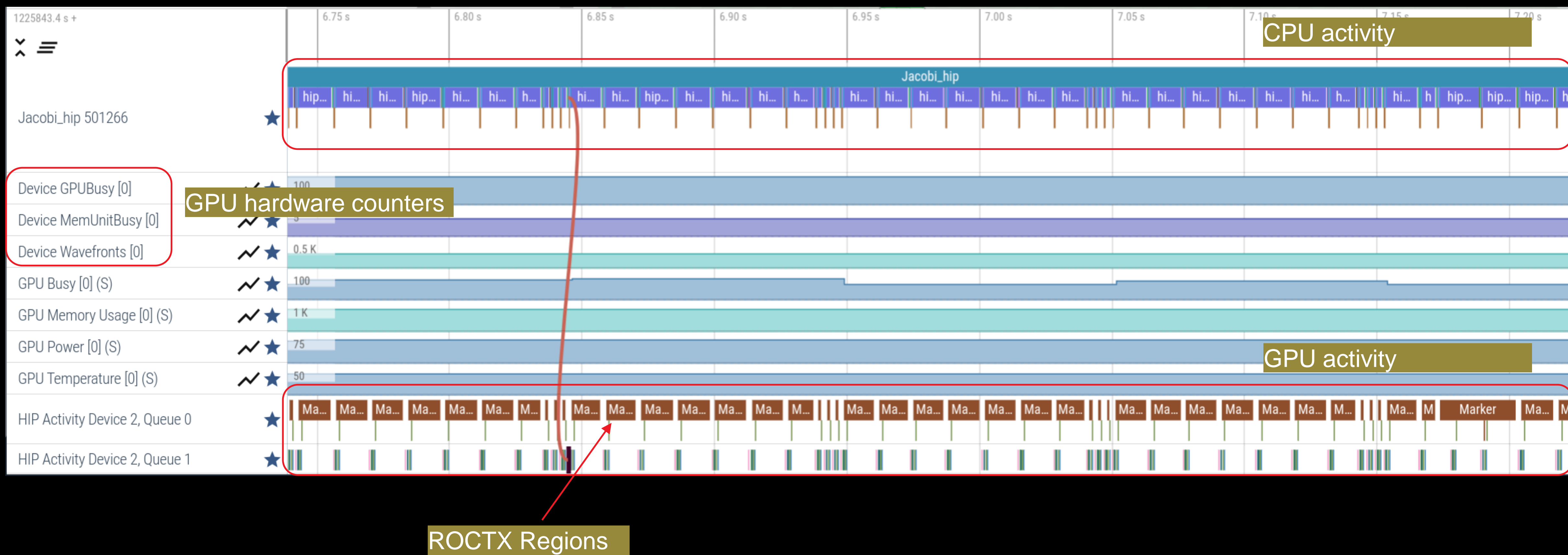
```
$ mpirun -np 1 omnitrace-run -- ./Jacobi_hip.inst -g 1 1
```

```
[omnitrace][501266][0][omnitrace_finalize] Finalizing perfetto...
[omnitrace][501266][perfetto]> Outputting '/shared/prod/home/ssitaram/HPCTrainingExamples/HIP/jacobi/omnitrace-Jacobi_hip-output/2023-03-15_22.57/perfetto-trace-0.proto' (11
.. Done
[omnitrace][501266][rocprof-device-0-GPUBusy]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/rocprof-device-0-GPUBusy-0.json'
[omnitrace][501266][rocprof-device-0-GPUBusy]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/rocprof-device-0-GPUBusy-0.txt'
[omnitrace][501266][rocprof-device-0-Wavefronts]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/rocprof-device-0-Wavefronts-0.json'
[omnitrace][501266][rocprof-device-0-Wavefronts]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/rocprof-device-0-Wavefronts-0.txt'
[omnitrace][501266][rocprof-device-0-MemUnitBusy]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/rocprof-device-0-MemUnitBusy-0.json'
[omnitrace][501266][rocprof-device-0-MemUnitBusy]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/rocprof-device-0-MemUnitBusy-0.txt'
[omnitrace][501266][trip_count]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/trip_count-0.json'
[omnitrace][501266][trip_count]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/trip_count-0.txt'
[omnitrace][501266][wall_clock]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/wall_clock-0.json'
[omnitrace][501266][wall_clock]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/wall_clock-0.txt'
[omnitrace][501266][roctracer]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/roctracer-0.json'
[omnitrace][501266][roctracer]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/roctracer-0.txt'
[omnitrace][501266][sampling_percent]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_percent-0.json'
[omnitrace][501266][sampling_percent]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_percent-0.txt'
[omnitrace][501266][sampling_cpu_clock]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_cpu_clock-0.json'
[omnitrace][501266][sampling_cpu_clock]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_cpu_clock-0.txt'
[omnitrace][501266][sampling_wall_clock]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_wall_clock-0.json'
[omnitrace][501266][sampling_wall_clock]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_wall_clock-0.txt'
[omnitrace][501266][sampling_gpu_memory_usage]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_gpu_memory_usage-0.json'
[omnitrace][501266][sampling_gpu_memory_usage]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_gpu_memory_usage-0.txt'
[omnitrace][501266][sampling_gpu_power]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_gpu_power-0.json'
[omnitrace][501266][sampling_gpu_power]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_gpu_power-0.txt'
[omnitrace][501266][sampling_gpu_temperature]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_gpu_temperature-0.json'
[omnitrace][501266][sampling_gpu_temperature]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_gpu_temperature-0.txt'
[omnitrace][501266][sampling_gpu_busy_percent]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_gpu_busy_percent-0.json'
[omnitrace][501266][sampling_gpu_busy_percent]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/sampling_gpu_busy_percent-0.txt'
[omnitrace][501266][metadata]> Outputting 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/metadata-0.json' and 'omnitrace-Jacobi_hip-output/2023-03-15_22.57/functions-0.json'
[omnitrace][501266][0][omnitrace_finalize] Finalized: 31.657272 sec wall_clock,    0.000 MB peak_rss,  179.700 MB page_rss, 29.950000 sec cpu_clock,   94.6 % cpu_util
[889.832]       perfetto.cc:60129 Tracing session 1 ended, total sessions:0
```

GPU hardware counters

AMD
together we advance_

Visualization with Hardware Counters



AMD
together we advance_

Tracing Multiple Ranks

[AMD Official Use Only - General]

Profiling Multiple MPI Ranks – Jacobi Example

Binary Rewrite

Generating a new /library with instrumentation built-in:

```
$ omnitrace-instrument -o Jacobi_hip.inst --
./Jacobi_hip
```

Run the instrumented binary with 2 ranks:

```
$ mpirun -np 2 omnitrace-run --./Jacobi_hip.inst -g
2 1
```

```
[omnitrace][3628199][perfetto]> Outputting '/home/ssitaram/git/HPCTrainingExamples/HIP/jacobi/omnitrace-Jacobi_hip.inst-output/2023-03-15_18.02/perfetto-trace-1.proto'
[perfetto]> Outputting '/home/ssitaram/git/HPCTrainingExamples/HIP/jacobi/omnitrace-Jacobi_hip.inst-output/2023-03-15_18.02/perfetto-trace-0.proto' (7856.71 KB / 7.86 M

[omnitrace][3628199][wall_clock]> Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_18.02/wall_clock-1.json'
[omnitrace][3628196][wall_clock]> Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_18.02/wall_clock-0.json'
[omnitrace][3628199][wall_clock]> Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_18.02/wall_clock-1.txt'
[omnitrace][3628196][wall_clock]> Outputting 'omnitrace-Jacobi_hip.inst-output/2023-03-15_18.02/wall_clock-0.txt'
```

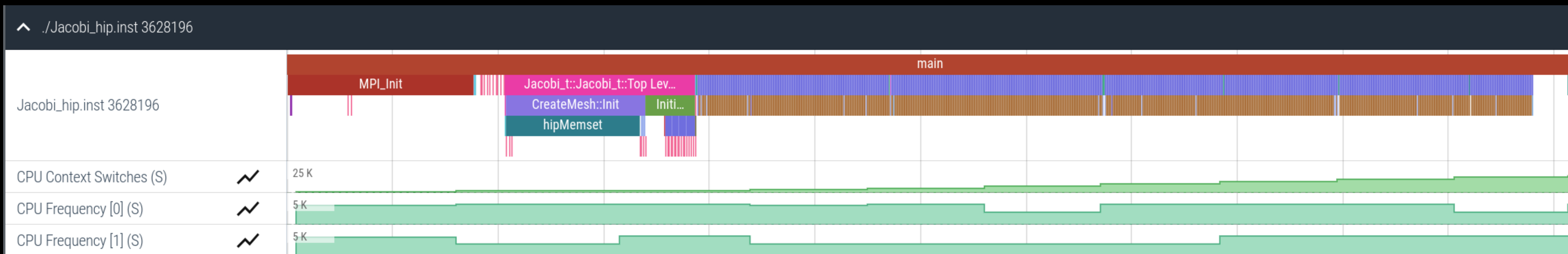
All output files are generated for each rank

AMD
together we advance_

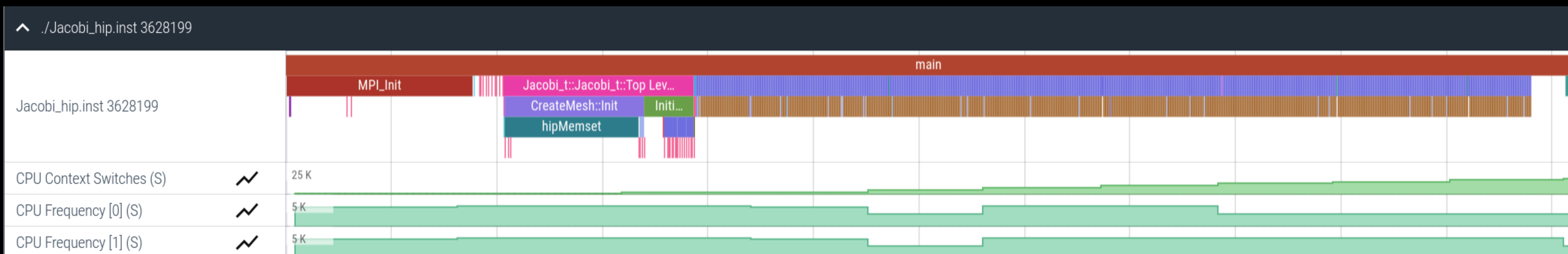
[AMD Official Use Only - General]

Visualizing Traces from Multiple Ranks - Separately

MPI 0



MPI 1



AMD
together we advance_

[AMD Official Use Only - General]

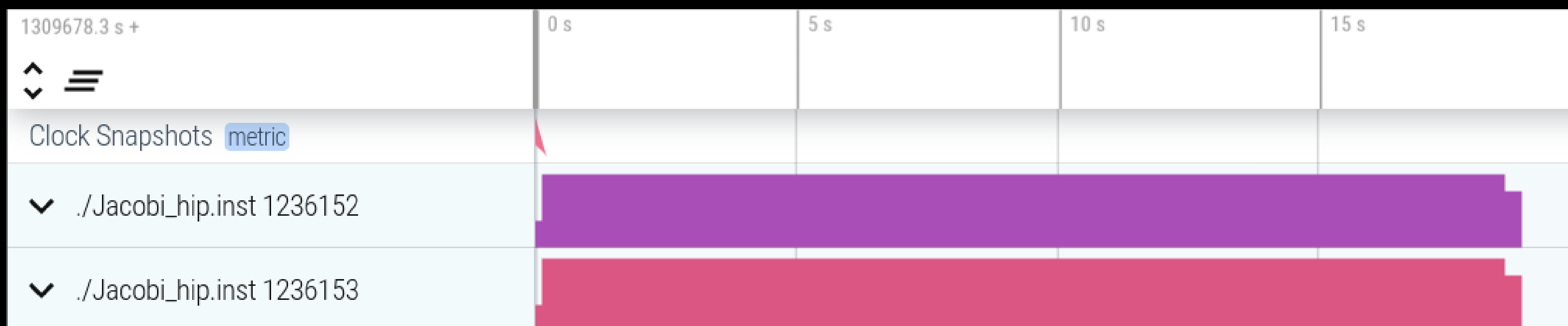
Visualizing Traces from Multiple Ranks - Combined

Merge Perfetto

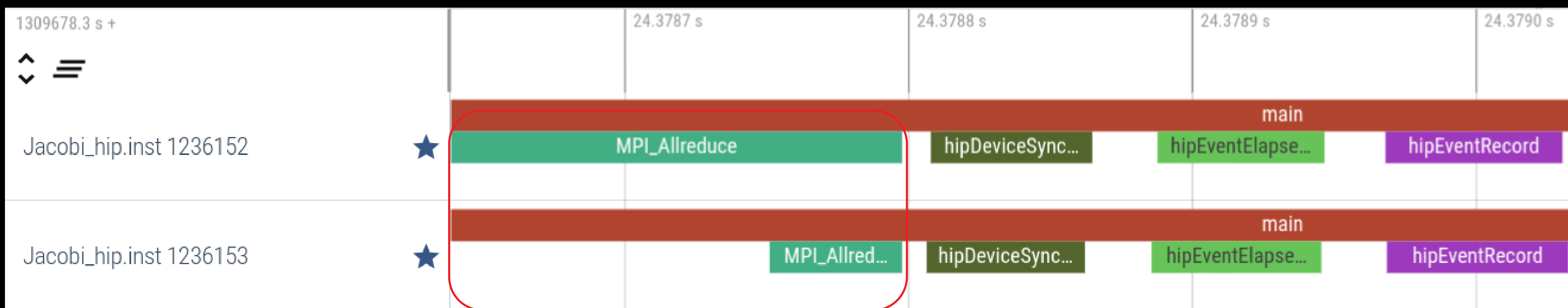
Use the following command to merge and concatenate multiple traces:

```
$ cat perfetto-trace-0.proto perfetto-trace-1.proto > allprocesses.proto
```

It seems there is an issue with newer Perfetto to visualize all the MPI processes



Two processes seen in combined trace file



Zooming in helps understand load balance issues

AMD
together we advance_

Statistical Sampling

[AMD Official Use Only - General]

Sampling Call-Stack (I)

`OMNITRACE_USE_SAMPLING = false`

Jacobi_hip.inst 3624331
main
MPI_Init
Jacobi_t::Jacobi_t::Top Lev...
CreateMesh::Init
Initi...
hipMemset

`OMNITRACE_USE_SAMPLING = true; OMNITRACE_SAMPLING_FREQ = 100` (100 samples per second)

samples [omnitrace]
Jacobi_t::Jacobi_t(grid_t&, mesh_t&)
Jacobi_t::CreateMesh()
hipMemset
hipApiName
hipDeviceGetByPCIBusId
hipExtStreamGetCUMask
hipExtStreamGetCUMask
hiprtcLinkAddData
hiprtcLinkAddData
hiprtcLinkAddData
hiprtcLinkAddData
hiprtcLinkAddData
hiprtcLinkAddData
hiprtcLinkAddData
hiprtcLinkAddData
hiprtcLinkAddData
hiprtcLinkAddData
amd_comgr_do_action
amd_comgr_data_set_remove
amd_comgr_data_set_remove
amd_comgr_data_set_remove
amd_comgr_data_set_remove
amd_comgr_data_set_remove

Each sample shows the call stack at that time

Scroll down all the way in Perfetto to see the sampling output!

AMD
together we advance_

[AMD Official Use Only - General]

Sampling Call-Stack (II)

Zoom in call-stack sampling

samples [omnitrace]										
Jacobi_…	Jacobi_t::Run()	Jacobi_t::Run()	Jacobi_t::Run()	Jacobi_t::Run()	Jacobi_t::Run()	Jacobi_t::Run()	Jacobi_t::Run()	Jacobi_t::Run()	Jacobi_t::Run()	Jacobi_t::Ru…
Norm(gr…	LocalLaplacian(gri…	Norm(grid_t&, me…	Norm(grid_t&, me…	hipEventRecord	Norm(grid_t&, me…	JacobiIteration(…	HaloExchange(gri…	LocalLaplacian(g…	HaloExchange(grid_…	Norm(grid_t&…
hipMemc…	hipLaunchKernel	hipMemcpy	hipMemcpy	std::basic_string<…	hipMemcpy	hipLaunchKernel	hipStreamSynchro…	hipLaunchKernel	hipStreamSynchroni…	hipMemcpy
hipApiN…	std::basic_string<…	hipApiName	hipApiName	OnUnload	hipApiName	std::basic_strin…	std::basic_strin…	hipMemPoolGetAtt…	hipLaunchHostFunc	hipApiName
hiprtcL…	OnUnload	hiprtcLinkAddData	hiprtcLinkAddData	OnUnload	hiprtcLinkAddData	OnUnload	OnUnload	hip_impl::hipLau…	OnUnload	hiprtcLinkAd…
hiprtcL…	OnUnload	hiprtcLinkAddData	hiprtcLinkAddData		hiprtcLinkAddData		OnUnload	hipGetCmdName	OnUnload	hiprtcLinkAd…
hiprtcL…	OnUnload	hiprtcLinkAddData	hiprtcLinkAddData		hiprtcLinkAddData			__hipGetPCH	OnUnload	hiprtcLinkAd…
hiprtcL…	std::ostream& std:…	hiprtcLinkAddData	hiprtcLinkAddData		hiprtcLinkAddData			hipIpcGetEventHa…		hiprtcLinkAd…
hiprtcL…	std::ostreambuf_it…	hiprtcLinkAddData	hiprtcLinkAddData		hiprtcLinkAddData					hiprtcLinkAd…
hiprtcL…		hiprtcLinkAddData	hiprtcLinkAddData		hiprtcLinkAddData					hiprtcLinkAd…
hiprtcL…		hiprtcLinkAddData	hiprtcLinkAddData		hiprtcLinkAddData					hiprtcLinkAd…
hiprtcL…		hiprtcLinkAddData	hiprtcLinkAddData		hiprtcLinkAddData					hiprtcLinkAd…
roctrac…		roctracer_disabl…	roctracer_disabl…		roctracer_disabl…					roctracer_di…
hsa_amd…		hsa_amd_image_ge…	hsa_amd_image_ge…		hsa_amd_image_ge…					hsa_amd_imag…

Thread 0 (S) 3625610 ← Sampling data is annotated with (S)

AMD
together we advance_

Other Features

[AMD Official Use Only - General]

Kernel Durations

```
$ cat omnitrace-Jacobi_hip.inst-output/2023-03-15_13.57/wall_clock-0.txt
```

If you do not see a wall_clock.txt dumped by omnitrace, try modify the config file `$HOME/.omnitrace.cfg` and enable `OMNITRACE_USE_TIMEMORY`:

```
…
OMNITRACE_USE_PERFETTO                              = true
OMNITRACE_USE_TIMEMORY                              = true
OMNITRACE_USE_SAMPLING                              = false
…
```

Durations

```
| |0>>>         |_MPI_Allreduce                                                                    |       1 |        5 | wall_clock | sec    | 0.000012 | 0.000012 | 0.000012 | 0.000012 | 0.000000 | 0.000000 |  100.0 |
| |0>>>         |_hipDeviceSynchronize                                                             |       1 |        5 | wall_clock | sec    | 0.000019 | 0.000019 | 0.000019 | 0.000019 | 0.000000 | 0.000000 |   94.4 |
| |0>>>           |_NormKernel1(int, double, double, double const*, double*)                      |       1 |        6 | wall_clock | sec    | 0.000001 | 0.000001 | 0.000001 | 0.000001 | 0.000000 | 0.000000 |  100.0 |
| |0>>>           |_NormKernel2(int, double const*, double*)                                       |       1 |        6 | wall_clock | sec    | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |  100.0 |
| |0>>>         |_MPI_Barrier                                                                      |       1 |        5 | wall_clock | sec    | 0.000001 | 0.000001 | 0.000001 | 0.000001 | 0.000000 | 0.000000 |  100.0 |
| |0>>>         |_hipEventRecord                                                                   |       2 |        5 | wall_clock | sec    | 0.000027 | 0.000014 | 0.000011 | 0.000016 | 0.000000 | 0.000003 |  100.0 |
| |0>>>         |_Halo D2H::Halo Exchange                                                          |       1 |        5 | wall_clock | sec    | 1.628420 | 1.628420 | 1.628420 | 1.628420 | 0.000000 | 0.000000 |    0.0 |
| |0>>>           |_hipStreamSynchronize                                                           |       1 |        6 | wall_clock | sec    | 0.000003 | 0.000003 | 0.000003 | 0.000003 | 0.000000 | 0.000000 |  100.0 |
| |0>>>           |_MPI Exchange::Halo Exchange                                                    |       1 |        6 | wall_clock | sec    | 1.628395 | 1.628395 | 1.628395 | 1.628395 | 0.000000 | 0.000000 |    0.0 |
| |0>>>             |_MPI_Waitall                                                                  |       1 |        7 | wall_clock | sec    | 0.000002 | 0.000002 | 0.000002 | 0.000002 | 0.000000 | 0.000000 |  100.0 |
| |0>>>             |_Halo H2D::Halo Exchange                                                      |       1 |        7 | wall_clock | sec    | 1.628104 | 1.628104 | 1.628104 | 1.628104 | 0.000000 | 0.000000 |    0.0 |
| |0>>>               |_hipStreamSynchronize                                                       |       1 |        8 | wall_clock | sec    | 0.000003 | 0.000003 | 0.000003 | 0.000003 | 0.000000 | 0.000000 |  100.0 |
| |0>>>               |_hipLaunchKernel                                                            |       5 |        8 | wall_clock | sec    | 0.000615 | 0.000123 | 0.000005 | 0.000578 | 0.000000 | 0.000254 |   99.6 |
| |0>>>                 |_mbind                                                                    |       1 |        9 | wall_clock | sec    | 0.000003 | 0.000003 | 0.000003 | 0.000003 | 0.000000 | 0.000000 |  100.0 |
| |0>>>               |_hipMemcpy                                                                  |       1 |        8 | wall_clock | sec    | 0.001122 | 0.001122 | 0.001122 | 0.001122 | 0.000000 | 0.000000 |   99.9 |
| |0>>>                 |_LocalLaplacianKernel(int, int, int, double, double, double const*, double*) |       1 |        9 | wall_clock | sec    | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |  100.0 |
| |0>>>                 |_HaloLaplacianKernel(int, int, int, double, double, double const*, double const*, double*) | 1 | 9 | wall_clock | sec | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 100.0 |
| |0>>>                 |_JacobiIterationKernel(int, double, double, double const*, double const*, double*, double*) | 1 | 9 | wall_clock | sec | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 100.0 |
```

Call Stack

Text file is for quick reference. JSON output is easy to script for and can be read by Hatchet, a Python package (https://hatchet.readthedocs.io/en/latest/)

AMD
together we advance_

Kernel Durations (flat profile)

Edit in your omnitrace.cfg:

```
OMNITRACE_USE_TIMEMORY       = true
OMNITRACE_FLAT_PROFILE       = true
```

Use flat profile to see aggregate duration of kernels and functions

REAL-CLOCK TIMER (I.E. WALL-CLOCK TIMER)

LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	MIN	MAX	VAR	STDDEV	% SELF
|0>>> main	1	0	wall_clock	sec	82.739099	82.739099	82.739099	82.739099	0.000000	0.000000	100.0
|0>>> MPI_Init	1	0	wall_clock	sec	34.056610	34.056610	34.056610	34.056610	0.000000	0.000000	100.0
|0>>> pthread_create	3	0	wall_clock	sec	0.014644	0.004881	0.001169	0.011974	0.000038	0.006145	100.0
|0>>> mbind	285	0	wall_clock	sec	0.001793	0.000006	0.000005	0.000020	0.000000	0.000002	100.0
|0>>> MPI_Comm_dup	1	0	wall_clock	sec	0.000212	0.000212	0.000212	0.000212	0.000000	0.000000	100.0
|0>>> MPI_Comm_rank	1	0	wall_clock	sec	0.000041	0.000041	0.000041	0.000041	0.000000	0.000000	100.0
|0>>> MPI_Comm_size	1	0	wall_clock	sec	0.000004	0.000004	0.000004	0.000004	0.000000	0.000000	100.0
|0>>> hipInit	1	0	wall_clock	sec	0.000372	0.000372	0.000372	0.000372	0.000000	0.000000	100.0
|0>>> hipGetDeviceCount	1	0	wall_clock	sec	0.000017	0.000017	0.000017	0.000017	0.000000	0.000000	100.0
|0>>> MPI_Allgather	1	0	wall_clock	sec	0.000009	0.000009	0.000009	0.000009	0.000000	0.000000	100.0
|0>>> hipSetDevice	1	0	wall_clock	sec	0.000024	0.000024	0.000024	0.000024	0.000000	0.000000	100.0
|0>>> hipHostMalloc	3	0	wall_clock	sec	0.126827	0.042276	0.000176	0.126453	0.005314	0.072900	100.0
|0>>> hipMalloc	7	0	wall_clock	sec	0.000458	0.000065	0.000024	0.000178	0.000000	0.000052	100.0
|0>>> hipMemset	1	0	wall_clock	sec	35.770403	35.770403	35.770403	35.770403	0.000000	0.000000	100.0
|0>>> hipStreamCreate	2	0	wall_clock	sec	0.016750	0.008375	0.005339	0.011412	0.000018	0.004295	100.0
|0>>> hipMemcpy	1005	0	wall_clock	sec	8.506781	0.008464	0.000610	0.039390	0.000023	0.004844	100.0
|0>>> hipEventCreate	2	0	wall_clock	sec	0.000037	0.000018	0.000016	0.000021	0.000000	0.000003	100.0
|0>>> hipLaunchKernel	5002	0	wall_clock	sec	0.181301	0.000036	0.000025	0.012046	0.000000	0.000278	100.0
|0>>> MPI_Allreduce	1003	0	wall_clock	sec	0.002009	0.000002	0.000001	0.000022	0.000000	0.000001	100.0
|0>>> hipDeviceSynchronize	1001	0	wall_clock	sec	0.016813	0.000017	0.000015	0.000043	0.000000	0.000004	100.0
|0>>> MPI_Barrier	3	0	wall_clock	sec	0.000007	0.000002	0.000001	0.000004	0.000000	0.000001	100.0
|0>>> hipEventRecord	2000	0	wall_clock	sec	0.046701	0.000023	0.000020	0.000225	0.000000	0.000006	100.0
|0>>> hipStreamSynchronize	2000	0	wall_clock	sec	0.030366	0.000015	0.000013	0.000382	0.000000	0.000009	100.0
|0>>> MPI_Waitall	1000	0	wall_clock	sec	0.001665	0.000002	0.000002	0.000007	0.000000	0.000000	100.0
|0>>> NormKernel1(int, double, double, double const*, double*)	1001	0	wall_clock	sec	0.001502	0.000002	0.000001	0.000006	0.000000	0.000000	100.0
|0>>> NormKernel2(int, double const*, double*)	1000	0	wall_clock	sec	0.001972	0.000002	0.000001	0.000003	0.000000	0.000001	100.0
|0>>> LocalLaplacianKernel(int, int, int, double, double, double const*, double*)	1000	0	wall_clock	sec	0.001488	0.000001	0.000001	0.000007	0.000000	0.000000	100.0
|0>>> HaloLaplacianKernel(int, int, int, double, double, double const*, double const*, double*)	1000	0	wall_clock	sec	0.001465	0.000001	0.000001	0.000007	0.000000	0.000000	100.0
|0>>> hipEventElapsedTime	1000	0	wall_clock	sec	0.015060	0.000015	0.000014	0.000041	0.000000	0.000002	100.0
|0>>> JacobiIterationKernel(int, double, double, double const*, double const*, double*, double*)	1000	0	wall_clock	sec	0.002598	0.000003	0.000001	0.000006	0.000000	0.000001	100.0
|0>>> pthread_join	1	0	wall_clock	sec	0.000396	0.000396	0.000396	0.000396	0.000000	0.000000	100.0
|0>>> hipFree	4	0	wall_clock	sec	0.000526	0.000131	0.000021	0.000243	0.000000	0.000091	100.0
|0>>> hipHostFree	2	0	wall_clock	sec	0.000637	0.000318	0.000287	0.000350	0.000000	0.000044	100.0
|3>>> start_thread	1	0	wall_clock	sec	0.004802	0.004802	0.004802	0.004802	0.000000	0.000000	100.0
|1>>> start_thread	1	0	wall_clock	sec	81.987779	81.987779	81.987779	81.987779	0.000000	0.000000	100.0
|2>>> start_thread	-	0	-	-	-	-	-	-	-	-	-

AMD
together we advance_

[AMD Official Use Only - General]

User API

Omnitrace provides an API to control the instrumentation

API Call	Description
`int` **`omnitrace_user_start_trace(void)`**	Enable tracing on this thread and all subsequently created threads
`int` **`omnitrace_user_stop_trace(void)`**	Disable tracing on this thread and all subsequently created threads
`int` **`omnitrace_user_start_thread_trace(void)`**	Enable tracing on this specific thread. Does not apply to subsequently created threads
`int` **`omnitrace_user_stop_thread_trace(void)`**	Disable tracing on this specific thread. Does not apply to subsequently created threads
`int` **`omnitrace_user_push_region(void)`**	Start user defined region
`int` **`omnitrace_user_pop_region(void)`**	End user defined region, FILO (first in last out) is expected

All the API calls: https://amdresearch.github.io/omnitrace/user_api.html

AMD
together we advance_

[AMD Official Use Only - General]

OpenMP®

We use the example omnitrace/examples/openmp/

Build the code with CMake:

```
$ cmake -B build
```

Use the openmp-lu binary, which can be executed with:

```
$ export OMP_NUM_THREADS=4
$ srun -n 1 -c 4 ./openmp-lu
```

Create a new instrumented binary:

```
$ srun -n 1 omnitrace-instrument -o openmp-lu.inst --
./openmp-lu
```

Execute the new binary:

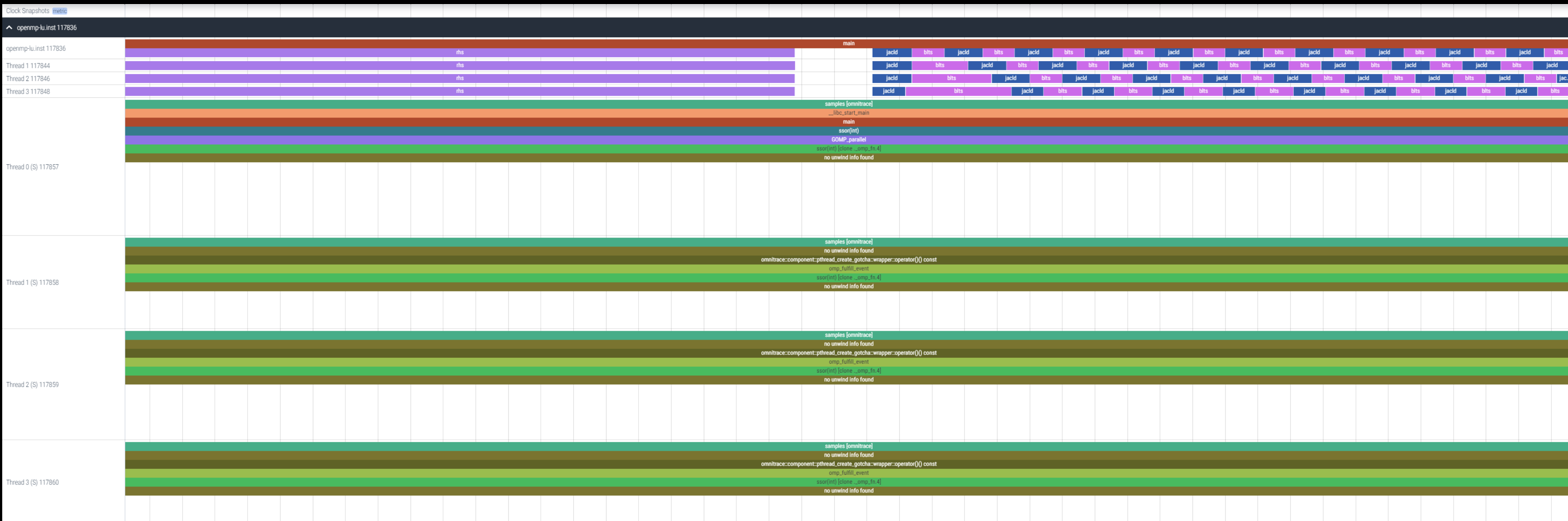
```
$ srun -n 1 -c 4 omnitrace-run -- ./openmp-lu.inst
```

```
|--------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                     REAL-CLOCK TIMER (I.E. WALL-CLOCK TIMER)                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|
|         LABEL          | COUNT  | DEPTH  |   METRIC   | UNITS  |   SUM    |   MEAN   |   MIN    |   MAX    |   VAR    |  STDDEV  | % SELF |
|------------------------|--------|--------|------------|--------|----------|----------|----------|----------|----------|----------|--------|
| |0>>> main             |      1 |      0 | wall_clock | sec    | 1.096702 | 1.096702 | 1.096702 | 1.096702 | 0.000000 | 0.000000 |    9.2 |
| |0>>> |_pthread_create |      3 |      1 | wall_clock | sec    | 0.002931 | 0.000977 | 0.000733 | 0.001420 | 0.000000 | 0.000385 |    0.0 |
| |3>>>   |_start_thread |      1 |      2 | wall_clock | sec    | 2.451520 | 2.451520 | 2.451520 | 2.451520 | 0.000000 | 0.000000 |   57.7 |
| |3>>>     |_erhs       |      1 |      3 | wall_clock | sec    | 0.001906 | 0.001906 | 0.001906 | 0.001906 | 0.000000 | 0.000000 |  100.0 |
| |3>>>     |_rhs        |    153 |      3 | wall_clock | sec    | 0.229893 | 0.001503 | 0.001410 | 0.001893 | 0.000000 | 0.000116 |  100.0 |
| |3>>>     |_jacld      |   3473 |      3 | wall_clock | sec    | 0.170568 | 0.000049 | 0.000047 | 0.000135 | 0.000000 | 0.000005 |  100.0 |
| |3>>>     |_blts       |   3473 |      3 | wall_clock | sec    | 0.232512 | 0.000067 | 0.000040 | 0.000959 | 0.000000 | 0.000034 |  100.0 |
| |3>>>     |_jacu       |   3473 |      3 | wall_clock | sec    | 0.166229 | 0.000048 | 0.000046 | 0.000148 | 0.000000 | 0.000005 |  100.0 |
| |3>>>     |_buts       |   3473 |      3 | wall_clock | sec    | 0.236484 | 0.000068 | 0.000041 | 0.000391 | 0.000000 | 0.000031 |  100.0 |
| |2>>>   |_start_thread |      1 |      2 | wall_clock | sec    | 2.452309 | 2.452309 | 2.452309 | 2.452309 | 0.000000 | 0.000000 |   58.1 |
| |2>>>     |_erhs       |      1 |      3 | wall_clock | sec    | 0.001895 | 0.001895 | 0.001895 | 0.001895 | 0.000000 | 0.000000 |  100.0 |
| |2>>>     |_rhs        |    153 |      3 | wall_clock | sec    | 0.229776 | 0.001502 | 0.001410 | 0.001893 | 0.000000 | 0.000115 |  100.0 |
| |2>>>     |_jacld      |   3473 |      3 | wall_clock | sec    | 0.204609 | 0.000059 | 0.000057 | 0.000152 | 0.000000 | 0.000006 |  100.0 |
| |2>>>     |_blts       |   3473 |      3 | wall_clock | sec    | 0.192986 | 0.000056 | 0.000047 | 0.000358 | 0.000000 | 0.000026 |  100.0 |
| |2>>>     |_jacu       |   3473 |      3 | wall_clock | sec    | 0.199029 | 0.000057 | 0.000055 | 0.000188 | 0.000000 | 0.000007 |  100.0 |
| |2>>>     |_buts       |   3473 |      3 | wall_clock | sec    | 0.198972 | 0.000057 | 0.000048 | 0.000372 | 0.000000 | 0.000026 |  100.0 |
| |1>>>   |_start_thread |      1 |      2 | wall_clock | sec    | 2.453072 | 2.453072 | 2.453072 | 2.453072 | 0.000000 | 0.000000 |   58.6 |
| |1>>>     |_erhs       |      1 |      3 | wall_clock | sec    | 0.001905 | 0.001905 | 0.001905 | 0.001905 | 0.000000 | 0.000000 |  100.0 |
| |1>>>     |_rhs        |    153 |      3 | wall_clock | sec    | 0.229742 | 0.001502 | 0.001410 | 0.001894 | 0.000000 | 0.000115 |  100.0 |
| |1>>>     |_jacld      |   3473 |      3 | wall_clock | sec    | 0.206418 | 0.000059 | 0.000057 | 0.000934 | 0.000000 | 0.000016 |  100.0 |
| |1>>>     |_blts       |   3473 |      3 | wall_clock | sec    | 0.186097 | 0.000054 | 0.000047 | 0.000344 | 0.000000 | 0.000023 |  100.0 |
| |1>>>     |_jacu       |   3473 |      3 | wall_clock | sec    | 0.198689 | 0.000057 | 0.000055 | 0.000186 | 0.000000 | 0.000006 |  100.0 |
| |1>>>     |_buts       |   3473 |      3 | wall_clock | sec    | 0.192470 | 0.000055 | 0.000048 | 0.000356 | 0.000000 | 0.000022 |  100.0 |
| |0>>> |_erhs           |      1 |      1 | wall_clock | sec    | 0.001961 | 0.001961 | 0.001961 | 0.001961 | 0.000000 | 0.000000 |  100.0 |
| |0>>> |_rhs            |    153 |      1 | wall_clock | sec    | 0.229889 | 0.001503 | 0.001410 | 0.001891 | 0.000000 | 0.000116 |  100.0 |
| |0>>> |_jacld          |   3473 |      1 | wall_clock | sec    | 0.208903 | 0.000060 | 0.000057 | 0.000359 | 0.000000 | 0.000017 |  100.0 |
| |0>>> |_blts           |   3473 |      1 | wall_clock | sec    | 0.172646 | 0.000050 | 0.000047 | 0.000822 | 0.000000 | 0.000020 |  100.0 |
| |0>>> |_jacu           |   3473 |      1 | wall_clock | sec    | 0.202130 | 0.000058 | 0.000055 | 0.000350 | 0.000000 | 0.000016 |  100.0 |
| |0>>> |_buts           |   3473 |      1 | wall_clock | sec    | 0.176975 | 0.000051 | 0.000048 | 0.000377 | 0.000000 | 0.000016 |  100.0 |
| |0>>> |_pintgr         |      1 |      1 | wall_clock | sec    | 0.000054 | 0.000054 | 0.000054 | 0.000054 | 0.000000 | 0.000000 |  100.0 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|
```


AMD
together we advance_

[AMD Official Use Only - General]

OpenMP® Visualization



AMD
together we advance_

[AMD Official Use Only - General]

Python™

The omnitrace Python package is installed in /path/omnitrace_install/lib/pythonX.Y/site-packages/omnitrace

Setup the environment:

```
$ export PYTHONPATH=/path/omnitrace/lib/python/site-packages/:${PYTHONPATH}
```

We use the Fibonacci example in omnitrace/examples/python/source.py

Execute the python program with:

```
$ omnitrace-python ./external.py
```

Profiled data is dumped in output directory:

```
$ cat omnitrace-source-output/timestamp/wall_clock.txt
```

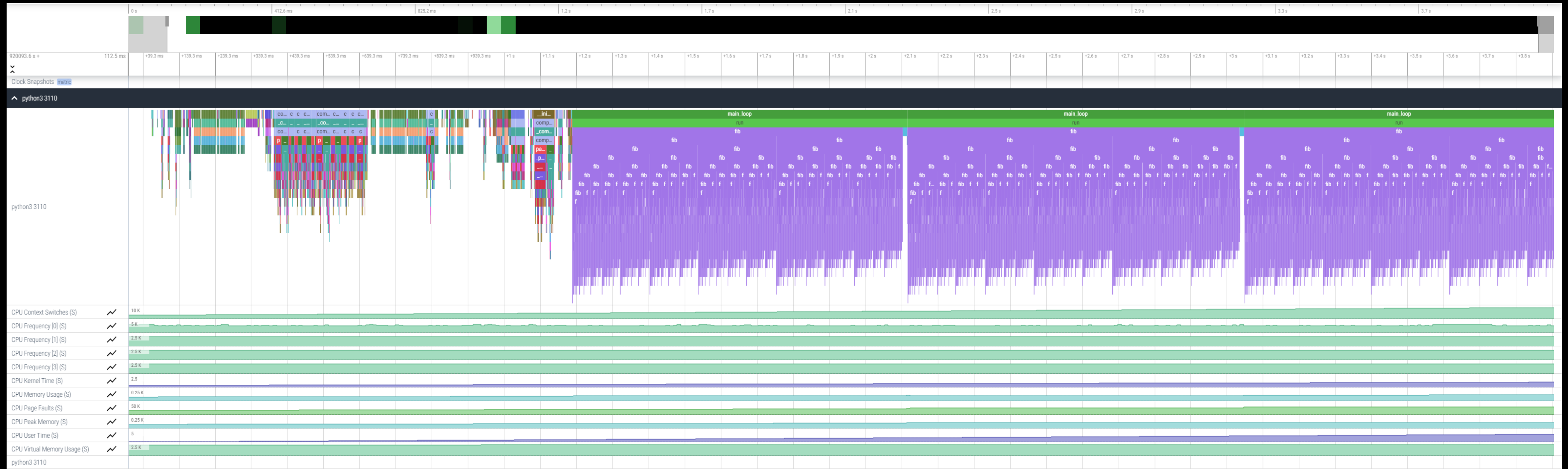
```
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                                     REAL-CLOCK TIMER (I.E. WALL-CLOCK TIMER)                                                                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                         LABEL                          | COUNT  | DEPTH  |   METRIC   | UNITS  |   SUM    |   MEAN   |   MIN    |   MAX    |   VAR    |  STDDEV  | % SELF |
|--------------------------------------------------------|--------|--------|------------|--------|----------|----------|----------|----------|----------|----------|--------|
| |0>>> main_loop                                        |      3 |      0 | wall_clock | sec    | 2.786075 | 0.928692 | 0.926350 | 0.932130 | 0.000009 | 0.003042 |    0.0 |
| |0>>> |_run                                            |      3 |      1 | wall_clock | sec    | 2.785799 | 0.928600 | 0.926250 | 0.932037 | 0.000009 | 0.003043 |    0.0 |
| |0>>>   |_fib                                          |      3 |      2 | wall_clock | sec    | 2.750104 | 0.916701 | 0.914454 | 0.919577 | 0.000007 | 0.002619 |    0.0 |
| |0>>>     |_fib                                        |      6 |      3 | wall_clock | sec    | 2.749901 | 0.458317 | 0.348962 | 0.567074 | 0.013958 | 0.118145 |    0.0 |
| |0>>>       |_fib                                      |     12 |      4 | wall_clock | sec    | 2.749511 | 0.229126 | 0.133382 | 0.350765 | 0.006504 | 0.080650 |    0.0 |
| |0>>>         |_fib                                    |     24 |      5 | wall_clock | sec    | 2.748734 | 0.114531 | 0.050867 | 0.217030 | 0.002399 | 0.048977 |    0.1 |
| |0>>>           |_fib                                  |     48 |      6 | wall_clock | sec    | 2.747118 | 0.057232 | 0.019302 | 0.134596 | 0.000806 | 0.028396 |    0.1 |
| |0>>>             |_fib                                |     96 |      7 | wall_clock | sec    | 2.743922 | 0.028583 | 0.007181 | 0.083350 | 0.000257 | 0.016026 |    0.2 |
| |0>>>               |_fib                              |    192 |      8 | wall_clock | sec    | 2.737564 | 0.014258 | 0.002690 | 0.051524 | 0.000079 | 0.008887 |    0.5 |
| |0>>>                 |_fib                            |    384 |      9 | wall_clock | sec    | 2.724966 | 0.007096 | 0.000973 | 0.031798 | 0.000024 | 0.004865 |    0.9 |
| |0>>>                   |_fib                          |    768 |     10 | wall_clock | sec    | 2.699251 | 0.003515 | 0.000336 | 0.019670 | 0.000007 | 0.002637 |    1.9 |
| |0>>>                     |_fib                        |   1536 |     11 | wall_clock | sec    | 2.648006 | 0.001724 | 0.000096 | 0.012081 | 0.000002 | 0.001417 |    3.9 |
| |0>>>                       |_fib                      |   3072 |     12 | wall_clock | sec    | 2.545260 | 0.000829 | 0.000016 | 0.007461 | 0.000001 | 0.000758 |    8.0 |
| |0>>>                         |_fib                    |   6078 |     13 | wall_clock | sec    | 2.342276 | 0.000385 | 0.000016 | 0.004669 | 0.000000 | 0.000404 |   16.0 |
| |0>>>                           |_fib                  |  10896 |     14 | wall_clock | sec    | 1.967475 | 0.000181 | 0.000015 | 0.002752 | 0.000000 | 0.000218 |   28.6 |
| |0>>>                             |_fib                |  15060 |     15 | wall_clock | sec    | 1.404069 | 0.000093 | 0.000015 | 0.001704 | 0.000000 | 0.000123 |   43.6 |
| |0>>>                               |_fib              |  14280 |     16 | wall_clock | sec    | 0.791873 | 0.000055 | 0.000015 | 0.001044 | 0.000000 | 0.000076 |   58.3 |
| |0>>>                                 |_fib            |   8826 |     17 | wall_clock | sec    | 0.330189 | 0.000037 | 0.000015 | 0.000620 | 0.000000 | 0.000050 |   70.9 |
| |0>>>                                   |_fib          |   3456 |     18 | wall_clock | sec    | 0.096120 | 0.000028 | 0.000015 | 0.000380 | 0.000000 | 0.000034 |   81.0 |
| |0>>>                                     |_fib        |    822 |     19 | wall_clock | sec    | 0.018294 | 0.000022 | 0.000015 | 0.000209 | 0.000000 | 0.000024 |   88.9 |
| |0>>>                                       |_fib      |    108 |     20 | wall_clock | sec    | 0.002037 | 0.000019 | 0.000016 | 0.000107 | 0.000000 | 0.000015 |   94.9 |
| |0>>>                                         |_fib    |      6 |     21 | wall_clock | sec    | 0.000104 | 0.000017 | 0.000016 | 0.000019 | 0.000000 | 0.000001 |  100.0 |
| |0>>>   |_inefficient                                  |      3 |      2 | wall_clock | sec    | 0.035450 | 0.011817 | 0.010096 | 0.012972 | 0.000002 | 0.001519 |   95.8 |
| |0>>>     |__sum                                       |      3 |      3 | wall_clock | sec    | 0.001494 | 0.000498 | 0.000440 | 0.000537 | 0.000000 | 0.000051 |  100.0 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
```

Python documentation: https://amdresearch.github.io/omnitrace/python.html

AMD
together we advance_

[AMD Official Use Only - General]

Visualizing Python™ Perfetto Tracing



AMD
together we advance_

[AMD Official Use Only - General]

Kokkos

Omnitrace can instrument Kokkos applications too.

Edit the $HOME/.omnitrace.cfg file and enable omnitrace:

```
...
OMNITRACE_USE_KOKKOSP = true
...
```

Profiling with omnitrace produces *kokkos*.txt files:

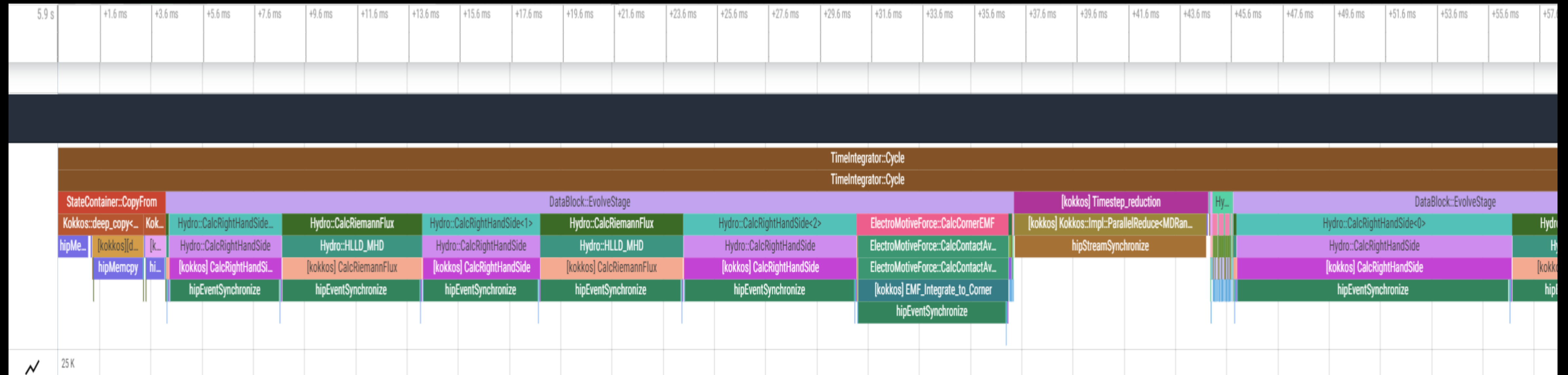
```
$ cat kokkos_memory0.txt
```

```
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence           |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence                 |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>   |_[kokkos][deep_copy] Host=DataBlock_A2_mirror HIP=DataBlock_A2                                   |     1 |     2 | kokkos_memory | MB     |  142 |  142 |  100 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check      |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check            |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence           |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence                 |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>   |_[kokkos][deep_copy] Host=DataBlock_dV_mirror HIP=DataBlock_dV                                   |     1 |     2 | kokkos_memory | MB     |  140 |  140 |  100 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check      |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check            |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence           |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence                 |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>> |_DataBlockHost::SyncToDevice()                                                                     |     1 |     1 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>   |_[kokkos][deep_copy] HIP=Hydro_Vc Host=Hydro_Vc_mirror                                           |     1 |     2 | kokkos_memory | MB     | 1124 | 1124 |  100 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check      |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check            |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence           |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence                 |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>   |_[kokkos][deep_copy] HIP=Hydro_InvDt Host=Hydro_InvDt_mirror                                     |     1 |     2 | kokkos_memory | MB     |  140 |  140 |  100 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check      |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check            |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence           |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence                 |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>   |_[kokkos][deep_copy] HIP=Hydro_Vs Host=Hydro_Vs_mirror                                           |     1 |     2 | kokkos_memory | MB     |  426 |  426 |  100 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, pre view equality check      |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, pre view equality check            |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos][dev0] Kokkos::deep_copy: copy between contiguous views, post deep copy fence           |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
| |0>>>     |_[kokkos] Kokkos::deep_copy: copy between contiguous views, post deep copy fence                 |     1 |     3 | kokkos_memory | MB     |    0 |    0 |    0 |
```


AMD
together we advance_

Visualizing Kokkos with Perfetto Trace

- Visualize `perfetto-trace-0.proto` (with sampling enabled)



AMD
together we advance_

Causal Profiling

[AMD Official Use Only - General]

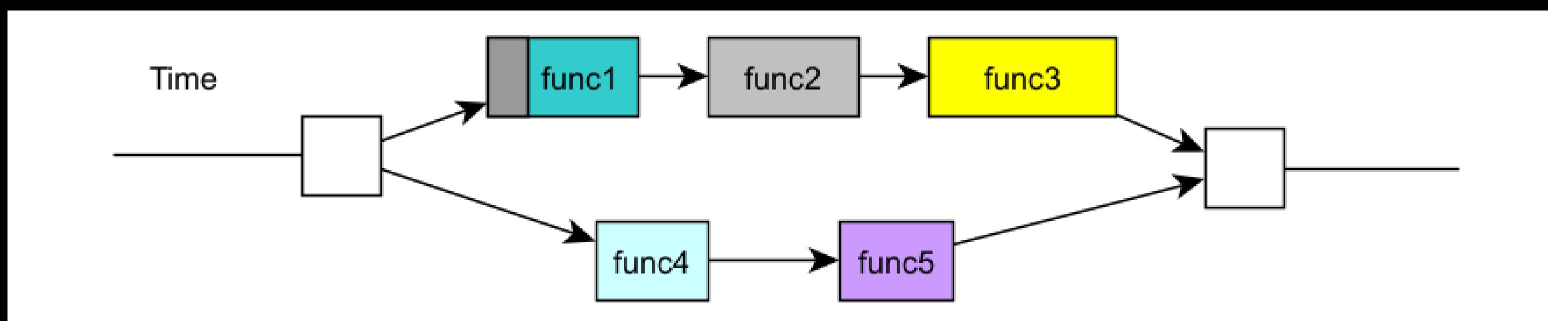
Causal Profiling

- Causal profiling requires multiple "experiments"
  - Each experiment has two independent variables:
    - Function/LOC selected for experiment
    - Virtual speed amount
  - Requires multiple runs of the application
    - For each function/LOC:
      - Baseline generation (0% virtual speedup)
      - 1+ virtual speedups > 0%
    - Speedup prediction is highly dependent on baseline
  - Progress points are required
    - Exception end to end runs
- Supports: sample space of fixed speedups, binary scope pattern, function scope pattern, source scope pattern, line scope pattern
- For now, it is for CPU threads and workload, work in progress for GPU kernels
- GUI available via PyPI: omnitrace-causal-viewer
- We use the example: https://github.com/AMDResearch/omnitrace/tree/main/examples/causal
- Documentation: https://amdresearch.github.io/omnitrace/causal_profiling.html

AMD
together we advance_

[AMD Official Use Only - General]

Causal Profiling – Example



What if, in order to decrease the performance of func4, we delayed the performance of func1?

AMD
together we advance_

[AMD Official Use Only - General]

Causal Profiling - Recommendations

- Generate a flat profile to get familiar with the functions that take most of the time
- Insert throughput progress points in high-traffic areas
- Reduce the virtual speedup sampling space
  - Default: 0-100 in increments of 5
- Use “scoping” to restrict the experiment sampling space
  - E.g. Binary scope, source scope, function scope, line scope
- Use the function mode initially because it reduces experiment sampling space
- Use the line mode in combination with a strict function scope

AMD
together we advance_

[AMD Official Use Only - General]

Progress points

- Progress points could be MPI, Kokkos, roctracer and other calls.
- You can use the USER API, declare in your code:

```
#include <omnitrace/causal.h>
#define CAUSAL_PROGRESS              OMNITRACE_CAUSAL_PROGRESS
#define CAUSAL_PROGRESS_NAMED(LABEL) OMNITRACE_CAUSAL_PROGRESS_NAMED(LABEL)
#define CAUSAL_BEGIN(LABEL)          OMNITRACE_CAUSAL_BEGIN(LABEL)
#define CAUSAL_END(LABEL)            OMNITRACE_CAUSAL_END(LABEL)
```

- Link also with the library libomnitrace-user.

AMD
together we advance_

Advanced options

- Source scope restricted to lines 100 and 110 of causal.cpp
  - -m line
  - -S "causal\\.cpp:(100|110)“
- Function scope, exclude functions which start with “kokkos::” or “std:enable_if”
  - -m func
  - -FE "^(Kokkos::|std::enable_if)"

AMD
together we advance_

[AMD Official Use Only - General]

Example – Causal-cpu-omni

We have two functions one fast and one slow that we can control their ratio

srun -n 1 -c2 ./causal-cpu-omni

Fraction: 70.00, iterations: 50, random seed: 4093769362 :: slow = 200000000, fast = 140000000, expected ratio = 70.00, sync every 1 iterations
executing iteration: 0
executing iteration: 10
executing iteration: 20
executing iteration: 30
executing iteration: 40
executing iteration: 49
slow_func() took 10000.891 ms
fast_func() took   7000.705 ms
total is   10001.183 ms
ratio is   70.001 %
rdiff is   0.001 %
Source code: https://github.com/AMDResearch/omnitrace/tree/main/examples/causal

AMD
together we advance_

[AMD Official Use Only - General]

Script to run various cases

```
#!/bin/bash -e

#create config file
cat << EOF > $PWD/causal.cfg
OMNITRACE_VERBOSE        = 0
OMNITRACE_OUTPUT_PREFIX = %argt%/
OMNITRACE_OUTPUT_PATH    = omnitrace-output
OMNITRACE_CAUSAL_BACKEND = perf
EOF

export OMNITRACE_CONFIG_FILE=${PWD}/causal.cfg
export SPEEDUPS="0,0,10,20-40:5,50,60-90:15"

#RESET=--reset
export RESET=""

omnitrace-causal        \
        ${RESET}        \
        -n 5            \
        -s ${SPEEDUPS}  \
        -m func         \
        --              \
        ./causal-cpu-omni "${@}"


omnitrace-causal        \
        ${RESET}        \
        -n 10           \
        -s ${SPEEDUPS}  \
        -m func         \
        -S "causal.cpp" \
        -o experiment.func \
         --              \
        ./causal-cpu-omni "${@}"
```

```
omnitrace-causal        \
        ${RESET}        \
        -n 10           \
        -s ${SPEEDUPS}  \
        -m line         \
        -S "causal.cpp" \
        -F "cpu_(slow|fast)_func" \
        -o experiment.line \
        --                 \
        ./causal-cpu-omni "${@}"


omnitrace-causal        \
        ${RESET}        \
        -n 2           \
        -s ${SPEEDUPS}  \
        -m line         \
        -S "causal.cpp" \
        -F "cpu_slow_func" "cpu_fast_func" \
        -o experiment.line.e2e \
        -e                 \
        --                 \
        ./causal-cpu-omni "${@}"
```

```
$ ./run-causal-demo.sh
$ omnitrace-causal-plot -w omnitrace-output/causal-cpu-omni/causal/
```

Open web browser with he provided link

AMD
together we advance_

[AMD Official Use Only - General]

Plots

Selected Causal Profiles

bool cpu_impl_func<false>(long, int)

iteration
cpu_impl

Program Speedup

Function Speedup

cpu_slow_func(long, int)

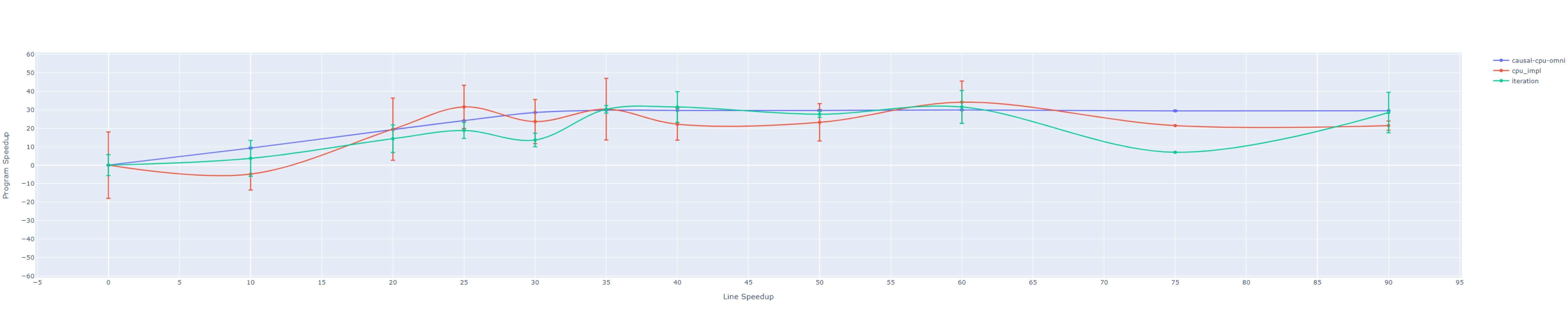
cpu_impl
iteration

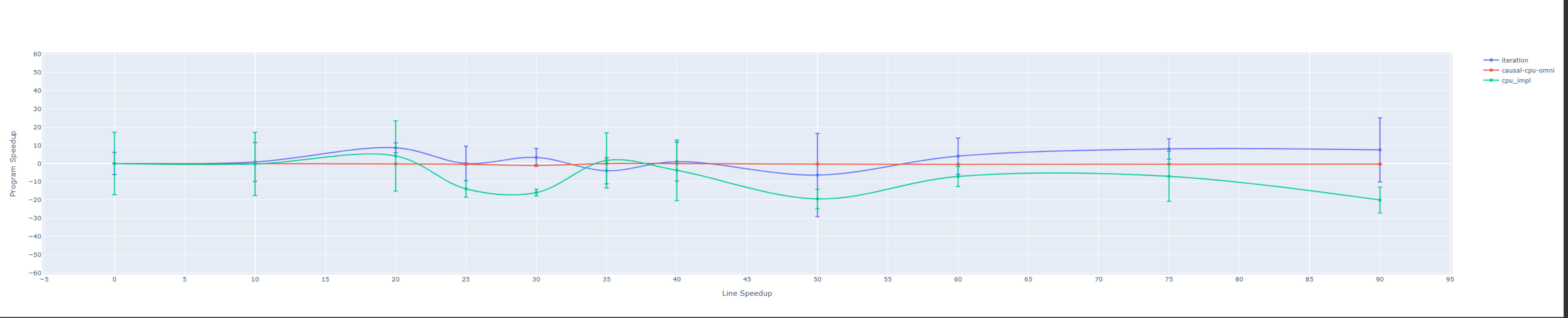
Program Speedup

Function Speedup

[AMD Official Use Only - General]

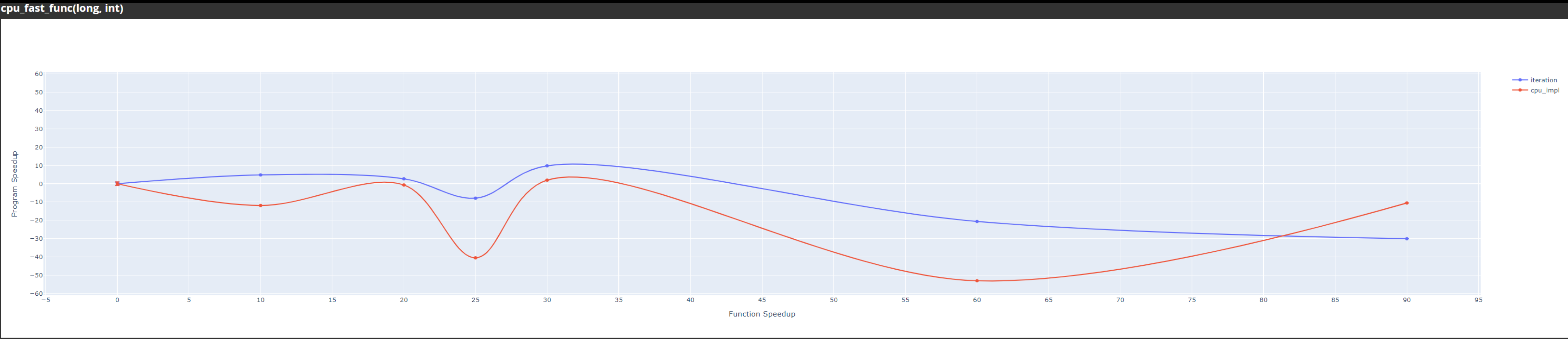
Plots





[AMD Official Use Only - General]

Plots



AMD
together we advance_

[AMD Official Use Only - General]

Other Executables

- `omnitrace-sample`
  - For sampling with low overhead, use `omnitrace-sample`
  - Use `omnitrace-sample --help` to get relevant options
  - Settings in the OmniTrace config file will be used by `omnitrace-sample`
  - Example invocation to get a flat tracing profile on Host and Device (`-PTHD`), excluding all components (`-E all`) and including only `rocm-smi, roctracer, rocprofiler` and `roctx` components (`-I ...`)

```
mpirun -np 1 omnitrace-sample -PTHD -E all -I rocm-smi -I roctracer -I rocprofiler -I roctx -- ./Jacobi_hip -g 1 1
```

- `omnitrace-causal`
  - Invokes causal profiling
- `omnitrace-critical-trace`
  - Post-processing tool for critical-trace data output by omnitrace

Current documentation: https://amdresearch.github.io/omnitrace/development.html#executables

AMD
together we advance_

[AMD Official Use Only - General]

Tips & Tricks

- My Perfetto timeline seems weird how can I check the clock skew?
  - Set `OMNITRACE_VERBOSE=1` or higher for verbose mode and it will print the timestamp skew
- It takes too long to map rocm-smi samples to kernels.
  - Temporarily set OMNITRACE_USE_ROCM_SMI=OFF
- What is the best way to profile multi-process runs?
  - Use OmniTrace's binary rewrite (-o) option to instrument the binary first, run the instrumented binary with mpirun/srun
- If you are doing binary rewrite and you do not get information about kernels, set:
  - HSA_TOOLS_LIB=libomnitrace.so in the env. and set OMNITRACE_USE_ROCTRACER=ON in the cfg file
- My HIP application hangs in different points, what do I do?
  - Try to set HSA_ENABLE_INTERRUPT=0 in the environment, this changes how HIP runtime is notified when GPU kernels complete
- My Perfetto trace is too big, can I decrease it?
  - Yes, with v1.7.3 and later declare OMNITRACE_PERFETTO_ANNOTATIONS to false
- I want to remove the many rows of CPU frequency lines from the Perfetto trace
  - Declare the OMNITRACE_USE_PROCESS_SAMPLING = false

AMD
together we advance_

[AMD Official Use Only - General]

Summary

- Omnitrace is a powerful tool to understand CPU + GPU activity
  - Ideal for an initial look at how an application runs
- Leverages several other tools and combines their data into a comprehensive output file
  - Some tools used are AMD uProf, rocprof, rocm-smi, roctracer, perf, etc.
- Easy to visualize traces in Perfetto
- Includes several features:
  - Dynamic Instrumentation either at Runtime or using Binary Rewrite
  - Statistical Sampling for call-stack info
  - Process sampling, monitoring of system metrics during application run
  - Causal Profiling
  - Critical Path Tracing

AMD
together we advance_

Introduction to Omniperf

and Hierarchical Roofline on AMD Instinct™ MI200 GPUs

AMD
together we advance_

Background – AMD Profilers

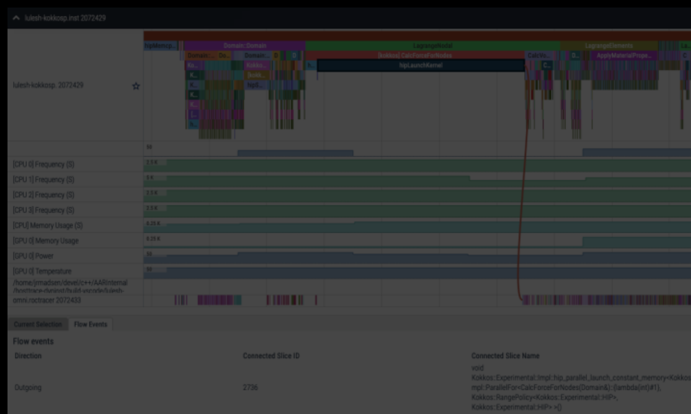
ROC-profiler (rocprof)

- **Hardware Counters**
  - Raw collection of GPU counters and traces
  - Counter collection with user input files
  - Counter results printed to a CSV
- **Traces and timelines**
  - Trace collection support for
  - CPU copy | HIP API | HSA API | GPU Kernels
- **Visualisation**
  - Traces visualized with Perfetto

	A	B	C	D	E
1	Name	Calls	TotalDura	AverageN:	Percentage
2	hipMemcpyAsync	99	3.22E+10	3.25E+08	44.14872
3	hipEventSynchronize	330	2.42E+10	73394557	33.225
4	hipMemsetAsync	87	7.76E+09	89232696	10.64953
5	hipHostMalloc	9	5.41E+09	6.01E+08	7.415198
6	hipDeviceSynchronize	28	1.32E+09	47006288	1.805515
7	hipHostFree	17	1.05E+09	61534688	1.435014
8	hipMemcpy	41	8.11E+08	19791876	1.113161
9	hipLaunchKernel	1856	58082083	31294	0.079676
10	hipStreamCreate	2	46380834	23190417	0.063625
11	hipMemset	2	18847246	9423623	0.025854
12	hipStreamDestroy	2	15183338	7591669	0.020828
13	hipFree	38	8269713	217624	0.011344
14	hipEventRecord	330	2520035	7636	0.003457
15	hipMalloc	30	1484804	49493	0.002037
16	__hipPopCallConfigura	1856	229159	123	0.000314
17	__hipPushCallConfigur	1856	224177	120	0.000308
18	hipGetLastError	1494	100458	67	0.000138
19	hipEventCreate	330	76675	232	0.000105
20	hipEventDestroy	330	64671	195	8.87E-05
21	hipGetDevicePropertie	47	51808	1102	7.11E-05
22	hipGetDevice	64	11611	181	1.59E-05
23	hipSetDevice	1	401	401	5.50E-07
24	hipGetDeviceCount	1	220	220	3.02E-07

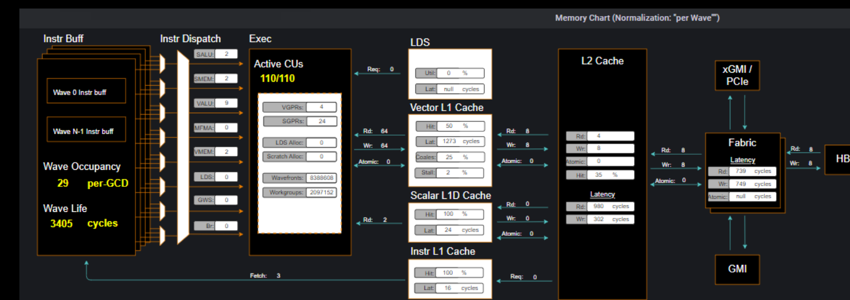
Omnitrace

- **Trace collection**
  - Comprehensive trace collection
  - CPU | GPU
- **Supports**
  - CPU copy | HIP API | HSA API | GPU Kernels
  - OpenMP® | MPI | Kokkos | p-threads | multi-GPU
- **Visualisation**
  - Traces visualized with Perfetto



Omniperf

- **Performance Analysis**
  - Automated collection of hardware counters
  - Analysis | Visualisation
- **Supports**
  - Speed of Light | Memory chart | Rooflines | Kernel comparison
- **Visualisation**
  - With Grafana or standalone GUI

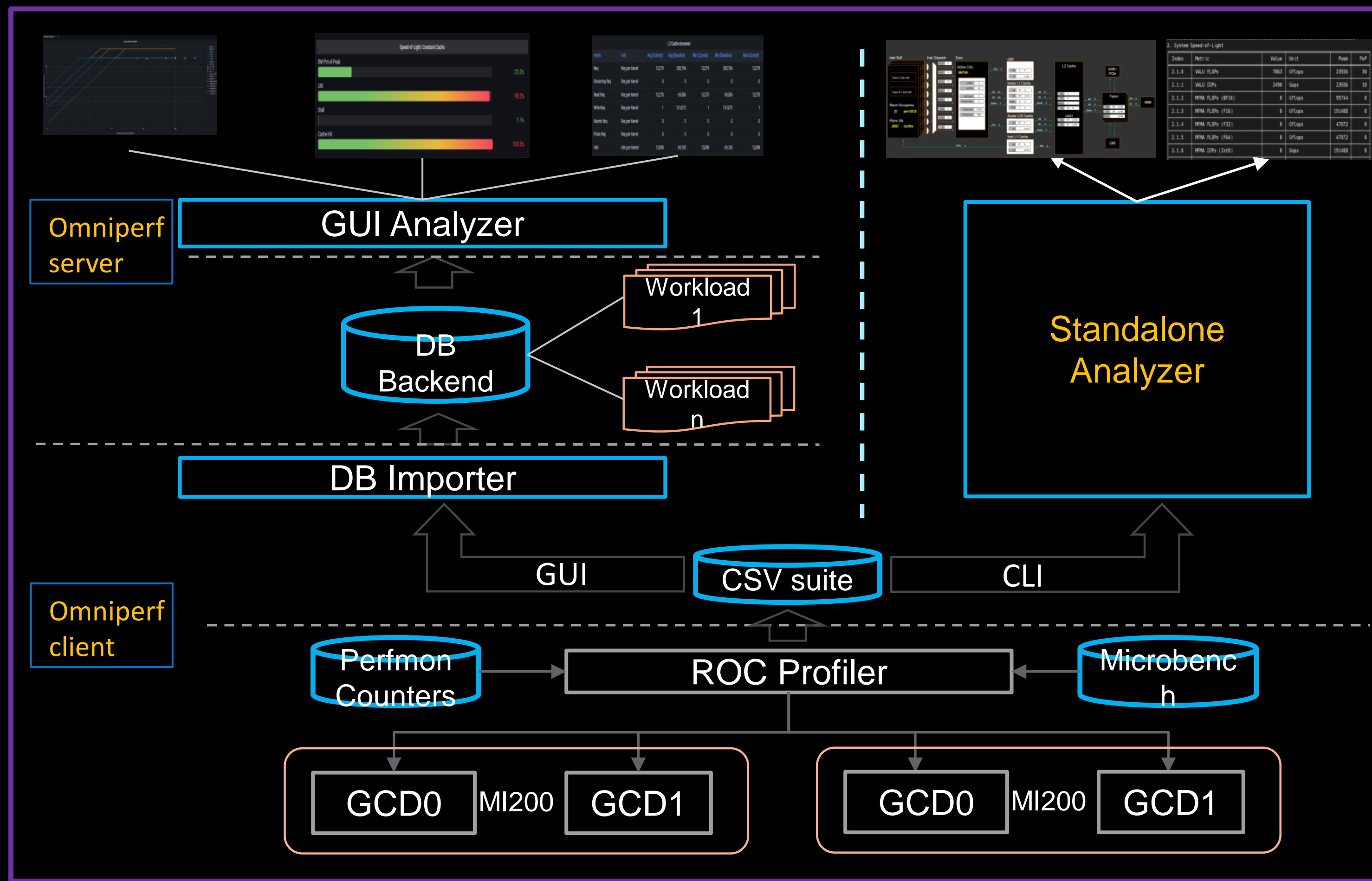


AMD
together we advance_

[AMD Official Use Only - General]

Omniperf: Automated Collection of Hardware Counters and Analysis

AMD Research Tool	Repository: https://github.com/AMDResearch/omniperf			
	Not part of ROCm stack		Built on top of ROC-profiler	
Integrated Performance Analyzer for AMD GPUs	Speed-of-Light	Roofline	Memory chart	Baseline comparison
	Sub-system performance analysis			
	LDS	vL1D	L2 Cache	HBM
	Shader Compute	Wavefront	Instruction mix	Latencies
INSTINCT™ Support	MI200		MI100	
User Interfaces	Grafana™ GUI	Standalone GUI	Command Line (CLI)	



Refer to current documentation for recent updates

AMD together we advance_

[AMD Official Use Only - General]

Omniperf features

Omniperf Features	
MI200 support	Roofline Analysis Panel (*Supported on MI200 only, SLES 15 SP3 or RHEL8*)
MI100 support	Command Processor (CP) Panel
Standalone GUI Analyzer	Shader Processing Input (SPI) Panel
Grafana/MongoDB GUI Analyzer	Wavefront Launch Panel
Dispatch Filtering	Compute Unit - Instruction Mix Panel
Kernel Filtering	Compute Unit - Pipeline Panel
GPU ID Filtering	Local Data Share (LDS) Panel
Baseline Comparison	Instruction Cache Panel
Multi-Normalizations	Scalar L1D Cache Panel
System Info Panel	Texture Addresser and Data Panel
System Speed-of-Light Panel	Vector L1D Cache Panel
Kernel Statistic Panel	L2 Cache Panel
Memory Chart Analysis Panel	L2 Cache (per-Channel) Panel

AMD
together we advance_

[AMD Official Use Only - General]

Omniperf

- Omniperf is an integrated performance analyzer for AMD GPUs built on ROCprofiler
- Omniperf executes the code many times to collect various hardware counters (over 100 counters default behavior)
- Using specific filtering options (kernel, dispatch ID, metric group), the overhead of profiling can be reduced
- Roofline analysis is supported on MI200 GPUs
- Omniperf shows many panels of metrics based on hardware counters, we will show a few here
- Typical Omniperf workflows:
  - Profile + Analyze with CLI or visualize with standalone GUI
  - Profile + Import to database and visualize with Grafana
- Omniperf targets MI100 and MI200 and future generation AMD GPUs
- Omniperf requires to use just 1 MPI process

- For problems, create an issue here: https://github.com/AMDResearch/omniperf/issues

AMD
together we advance_

Client-side installation (if required)

Download the latest version from here: https://github.com/AMDResearch/omniperf/releases

Full documentation: https://amdresearch.github.io/omniperf/

```
wget https://github.com/AMDResearch/omniperf/releases/download/v1.0.8-PR2/omniperf-v1.0.8-PR2.tar.gz

tar zxvf omniperf-v1.0.8-PR2.tar.gz

cd omniperf-v1.0.8-PR2/
python3 -m pip install -t ${INSTALL_DIR}/python-libs -r requirements.txt
mkdir build
cd build
export PYTHONPATH=$INSTALL_DIR/python-libs:$PYTHONPATH
cmake -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR}/1.0.8 \
        -DPYTHON_DEPS=${INSTALL_DIR}/python-libs \
        -DMOD_INSTALL_PATH=${INSTALL_DIR}/modulefiles ..
make install
export PATH=$INSTALL_DIR/1.0.8/bin:$PATH
```


AMD
together we advance_

[AMD Official Use Only - General]

Omniperf modes

Mode			
Profile	Target application is launched using AMD ROC-profiler		
	Kernels	Dispatches	IP Blocks
Analyze	Profiled data is loaded to omniperf CLI		
	Immediate access to metrics	Lightweight standalone GUI	
Database	Profiled data is imported to Grafana™ database		
	Grafana™ GUI is based on MongoDB	Interact with saved workload database	

Basic command-line syntax:

Profile:

```
$ omniperf profile -n workload_name [profile options]
                    [roofline options] -- <CMD> <ARGS>
```

Analyze:

```
$ omniperf analyze -p
<path/to/workloads/workload_name/mi200/>
```

To use a lightweight standalone GUI with CLI analyzer:

```
$ omniperf analyze -p
<path/to/workloads/workload_name/mi200/> --gui
```

Database:

```
$ omniperf database <interaction type> [connection options]
```

For more information or help use -h/--help/? flags:

```
$ omniperf profile --help
```

For problems, create an issue here: https://github.com/AMDResearch/omniperf/issues
Documentation: https://amdresearch.github.io/omniperf

AMD
together we advance_

[AMD Official Use Only - General]

Omniperf profiling

We use the example sample/vcopy.cpp from the Omniperf installation folder:

```
$ wget https://github.com/AMDResearch/omniperf/raw/main/sample/vcopy.cpp
```

Compile with hipcc:

```
$ hipcc -o vcopy vcopy.cpp
```

Profile with Omniperf:

```
 $ omniperf profile -n vcopy_all -- ./vcopy 1048576 256
…
-------------
Profile only
-------------

omniperf ver:  1.0.4
Path:  /pfs/lustrep4/scratch/project_462000075/markoman/omniperf-
1.0.4/build/workloads
Target:  mi200
Command:  ./vcopy 1048576 256
Kernel Selection:  None
Dispatch Selection:  None
IP Blocks: All
```

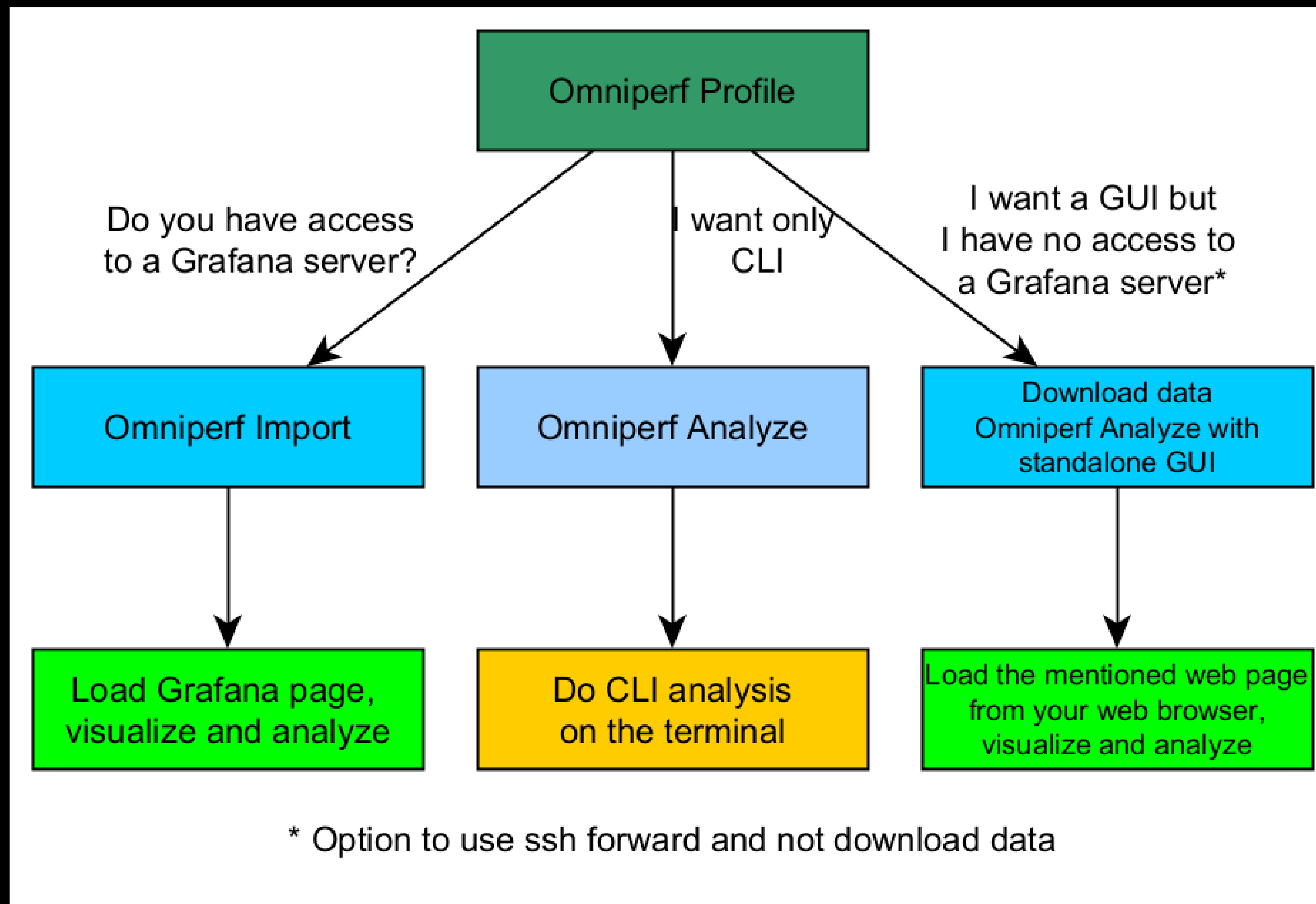
A new directory will be created called `workloads/vcopy_all`

Note: Omniperf executes the code as many times as required to collect all HW metrics. Use kernel/dispatch filters especially when trying to collect roofline analysis.

AMD
together we advance_

[AMD Official Use Only - General]

Omniperf workflows



AMD
together we advance_

[AMD Official Use Only - General]

Omniperf analyze

We use the example sample/vcopy.cpp from the Omniperf installation folder:

```
$ wget https://github.com/AMDResearch/omniperf/raw/main/sample/vcopy.cpp
```

Compile with hipcc:

```
$ hipcc --offload-arch=gfx90a -o vcopy vcopy.cpp
```

Profile with Omniperf:

```
$ omniperf profile -n vcopy_all -- ./vcopy 1048576 256
```

A new directory will be created called `workloads/vcopy_all`

Analyze the profiled workload:

```
$ omniperf analyze -p workloads/vcopy_all/mi200/ &> vcopy_analyze.txt
```

0. Top Stat

	KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pc
0	vecCopy(double*, double*, double*, int, int) [clone .kd]	1	341123.00	341123.00	341123.00	100.0

2. System Speed-of-Light

Index	Metric	Value	Unit	Peak	PoP
2.1.0	VALU FLOPs	0.00	Gflop	23936.0	0.0
2.1.1	VALU IOPs	89.14	Giop	23936.0	0.37242200388114116
2.1.2	MFMA FLOPs (BF16)	0.00	Gflop	95744.0	0.0
2.1.3	MFMA FLOPs (F16)	0.00	Gflop	191488.0	0.0
2.1.4	MFMA FLOPs (F32)	0.00	Gflop	47872.0	0.0
2.1.5	MFMA FLOPs (F64)	0.00	Gflop	47872.0	0.0
2.1.6	MFMA IOPs (Int8)	0.00	Giop	191488.0	0.0
2.1.7	Active CUs	58.00	Cus	110	52.72727272727273
2.1.8	SALU Util	3.69	Pct	100	3.6862586934167525
2.1.9	VALU Util	5.90	Pct	100	5.895531580380328
2.1.10	MFMA Util	0.00	Pct	100	0.0
2.1.11	VALU Active Threads/Wave	32.71	Threads	64	51.10526315789473
2.1.12	IPC - Issue	0.98	Instr/cycle	5	19.576640831930312

7.1 Wavefront Launch Stats

Index	Metric	Avg	Min	Max	Unit
7.1.0	Grid Size	1048576.00	1048576.00	1048576.00	Work items
7.1.1	Workgroup Size	256.00	256.00	256.00	Work items
7.1.2	Total Wavefronts	16384.00	16384.00	16384.00	Wavefronts
7.1.3	Saved Wavefronts	0.00	0.00	0.00	Wavefronts
7.1.4	Restored Wavefronts	0.00	0.00	0.00	Wavefronts
7.1.5	VGPRs	44.00	44.00	44.00	Registers
7.1.6	SGPRs	48.00	48.00	48.00	Registers
7.1.7	LDS Allocation	0.00	0.00	0.00	Bytes
7.1.8	Scratch Allocation	16496.00	16496.00	16496.00	Bytes

AMD
together we advance_

[AMD Official Use Only - General]

Omniperf Analyze

- Execute omniperf analyze –h to see various options
- Use specific IP block (-b)

Top kernels:

```
$ srun -n 1 --gpus 1 omniperf analyze -p workloads/vcopy_all/mi200/ -b 0
```

IP Block of wavefronts

```
$ srun -n 1 --gpus 1 omniperf analyze -p workloads/vcopy_all/mi200/ -b 7.1.2
```

```
--------------------------------------------------------------------------------
0. Top Stat
```

	KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
0	vecCopy(double*, double*, double*, int, int) [clone .kd]	1	20960.00	20960.00	20960.00	100.00

```
--------------------------------------------------------------------------------
7. Wavefront
7.1 Wavefront Launch Stats
```

Index	Metric	Avg	Min	Max	Unit
7.1.2	Total Wavefronts	16384.00	16384.00	16384.00	Wavefronts

AMD
together we advance_

[AMD Official Use Only - General]

Omniperf analyze

To see available options and usage instructions:

```
$ omniperf analyze -h
...

Help:
  -h, --help                    show this help message and exit

General Options:
  -v, --version                 show program's version number and exit
  -V, --verbose                 Increase output verbosity

Analyze Options:
  -p  [ ...], --path  [ ...]                    Specify the raw data root dirs or desired results directory.
  -o , --output                                 Specify the output file.
  --list-kernels                                List kernels. Top 10 kernels sorted by duration (descending order).
  --list-metrics                                List metrics can be customized to analyze on specific arch:
                                                   gfx906
                                                   gfx908
                                                   gfx90a
  -b  [ ...], --metric  [ ...]                  Specify IP block/metric id(s) from --list-metrics for filtering.
  -k  [ ...], --kernel  [ ...]                  Specify kernel id(s) from --list-kernels for filtering.
  --dispatch  [ ...]                            Specify dispatch id(s) for filtering.
  --gpu-id  [ ...]                              Specify GPU id(s) for filtering.
  -n , --normal-unit                            Specify the normalization unit: (DEFAULT: per_wave)
                                                   per_wave
                                                   per_cycle
                                                   per_second
                                                   per_kernel
  --config-dir                                  Specify the directory of customized configs.
  -t , --time-unit                              Specify display time unit in kernel top stats: (DEFAULT: ns)
                                                   s
                                                   ms
                                                   us
                                                   ns
  --decimal                                     Specify the decimal to display. (DEFAULT: 2)
  --cols  [ ...]                                Specify column indices to display.
  -g                                            Debug single metric.
  --dependency                                  List the installation dependency.
  --gui [GUI]                                   Activate a GUI to interate with Omniperf metrics.
                                                Optionally, specify port to launch application (DEFAULT: 8050)
```


AMD
together we advance_

[AMD Official Use Only - General]

Easy things you can check

- Are all the CUs being used?
  - If not, more parallelism is required (for most of the cases)
- Are all the VGPRs being spilled?
  - Try smaller workgroup sizes
- Is the code Integer limited?
  - Try reducing the integer ops, usually in the index calculation

AMD
together we advance_

Omniperf analyze with standalone GUI

We use the example sample/vcopy.cpp from the Omniperf installation folder:

```
$ wget https://github.com/AMDResearch/omniperf/raw/main/sample/vcopy.cpp
```

Compile with hipcc:

```
$ hipcc --offload-arch=gfx90a -o vcopy vcopy.cpp
```

Profile with Omniperf:

```
$ omniperf profile -n vcopy_all -- ./vcopy 1048576 256
```

A new directory will be created called `workloads/vcopy_all`

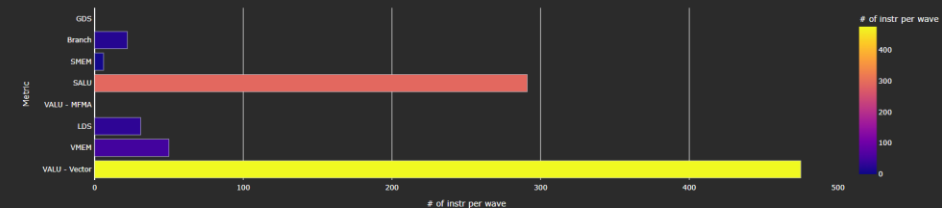
Analyze the profiled workload:

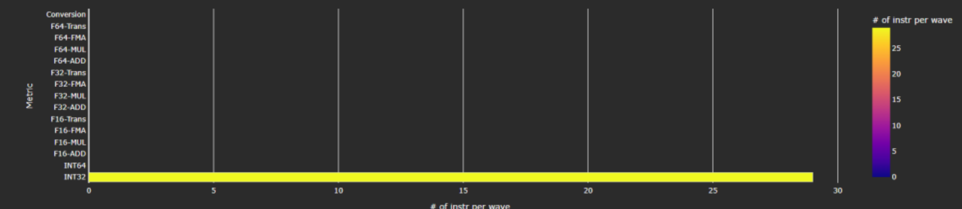
```
$ omniperf analyze -p workloads/vcopy_all/mi200/ --gui
```

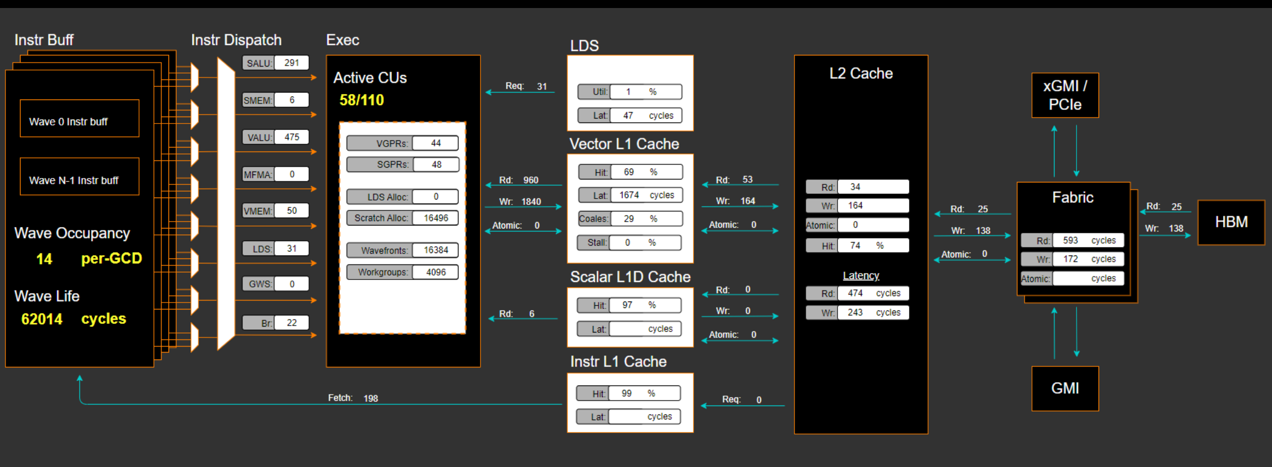
Open web page http://IP:8050/

2. System Speed-of-Light

Metric	Value	Unit	Peak	PoP
VALU FLOPs	0.00	Gflop	23936.00	0.00
VALU IOPs	89.14	Giop	23936.00	0.37
MFMA FLOPs (BF16)	0.00	Gflop	95744.00	0.00
MFMA FLOPs (F16)	0.00	Gflop	191488.00	0.00
MFMA FLOPs (F32)	0.00	Gflop	47872.00	0.00
MFMA FLOPs (F64)	0.00	Gflop	47872.00	0.00
MFMA IOPs (Int8)	0.00	Giop	191488.00	0.00
Active CUs	58.00	Cus	110.00	52.73







AMD
together we advance_

[AMD Official Use Only - General]

Omniperf analyze with Grafana™ GUI

We use the example sample/vcopy.cpp from the Omniperf installation folder:

```
$ wget https://github.com/AMDResearch/omniperf/raw/main/sample/vcopy.cpp
```

Compile with hipcc:

```
$ hipcc --offload-arch=gfx90a -o vcopy vcopy.cpp
```

Profile with Omniperf:

```
$ omniperf profile -n vcopy_all -- ./vcopy 1048576 256
```

A new directory will be created called `workloads/vcopy_all`

Import the database to analyze in Grafana™ GUI:

```
 $ omniperf database --import [connection options] -w workloads/vcopy_demo/mi200/
ROC Profiler:  /usr/bin/rocprof

--------
Import Profiling Results
--------

Pulling data from  /root/test/workloads/vcopy_demo/mi200
The directory exists
Found sysinfo file
KernelName shortening enabled
Kernel name verbose level: 2
Password:
Password recieved
-- Conversion & Upload in Progress –
… …
9 collections added.
Workload name uploaded
-- Complete! --
```

General / MIPerf_v1.0

2021-11-04 09:21:39 to 2

Normalization "per Wave" | Workload miperf_asw_vcopy_demo_mi200 | Dispatch Filter Enter variable value | GCD 0 | Kernels

Baseline Workload miperf_asw_vcopy_mi200 | Baseline Dispatch Filter Enter variable value | Baseline GCD 0 | Baseline Kernels All

TopN 5

› System Info (1 panel)

⌄ System Speed-of-Light

Speed of Light

Metric	Avg	Unit	Theoretical Max	Pct-of-Peak
VALU FLOPs	0	GFLOP	23,936	0%
VALU IOPs	379	GIOP	23,936	2%
MFMA FLOPs (BF16)	0	GFLOP	95,744	0%
MFMA FLOPs (F16)	0	GFLOP	191,488	0%
MFMA FLOPs (F32)	0	GFLOP	47,872	0%
MFMA FLOPs (F64)	0	GFLOP	47,872	0%
MFMA IOPs (Int8)	0	GIOP	191,488	0%
Active CUs	75	CUs	110	68%
SALU Util	4	pct	100	4%
VALU Util	6	pct	100	6%
MFMA Util	0	pct	100	0%
VALU Active Threads/Wave	64	Threads	64	100%
IPC - Issue	1	Instr/cycle	5	20%
LDS BW	0	GB/sec	23,936	0%
LDS Bank Conflict		Conflicts/access	32	
Instr Cache Hit Rate	100	pct	100	100%
Instr Cache BW	217	GB/s	6,093	4%
Scalar L1D Cache Hit Rate	100	pct	100	100%
Scalar L1D Cache BW	217	GB/s	6,093	4%
Vector L1D Cache Hit Rate	50	pct	100	50%
Vector L1D Cache BW	1,733	GB/s	11,968	14%
L2 Cache Hit Rate	36	pct	100	36%
L2-Fabric Read BW	434	GB/s	1,638	26%
L2-Fabric Write BW	301	GB/s	1,638	18%

Dispatch

Dispatch

AMD
together we advance_

Key Insights from Omniperf Analyzer

[AMD Official Use Only - General]

Grafana – System Info

General / Omniperf_v1.0.3_pub

Normalization "per Wave" | Workload miperf_aaa_vcopy_mi200 | Dispatch Filter Enter variable value | GCD 0 | Kernels All | Baseline Workload miperf_asw_vcopy_mi200 | Baseline Dispatch Filter Enter variable value | Baseline GCD 0 | Baseline Kernels All | Comparison Panels System Info | TopN 5

System Info

System Info

Metric	Current	Baseline
Date	Tue Jul 5 20:50:45 2022 (UTC)	Tue Jun 21 18:31:40 2022 (CDT)
Host Name	6fb5ce5e50da	node-bp126-014a
Host CPU	AMD Eng Sample: 100-000000248-08_35/21_N	AMD Eng Sample: 100-000000248-08_35/21_N
Host Distro	Ubuntu 20.04.4 LTS	Ubuntu 20.04.4 LTS
Host Kernel	5.9.1-amdsos-build32-1+	5.9.1-amdsos-build32-1+
ROCm Version	5.1.3-66	5.2.0-9768
GFX SoC	mi200	mi200
GFX ID	gfx90a	gfx90a
Total SEs	8	8
Total SQCs	56	56
Total CUs	110	110
SIMDs/CU	4	4
Max Wavefronts Occupancy Per CU	32	32
Max Workgroup Size	1,024	1,024
L1Cache per CU (KB)	16	16
L2Cache (KB)	8,192	8,192
L2Cache Channels	32	32
Sys Clock (Max) - MHz	1,700	1,700
Memory Clock (Max) - MHz	1,600	1,600
Sys Clock (Cur) - MHz	800	800
Memory Clock (Cur) - MHz	1,600	1,600
HBM Bandwidth - GB/s	1,638.4	1,638.4

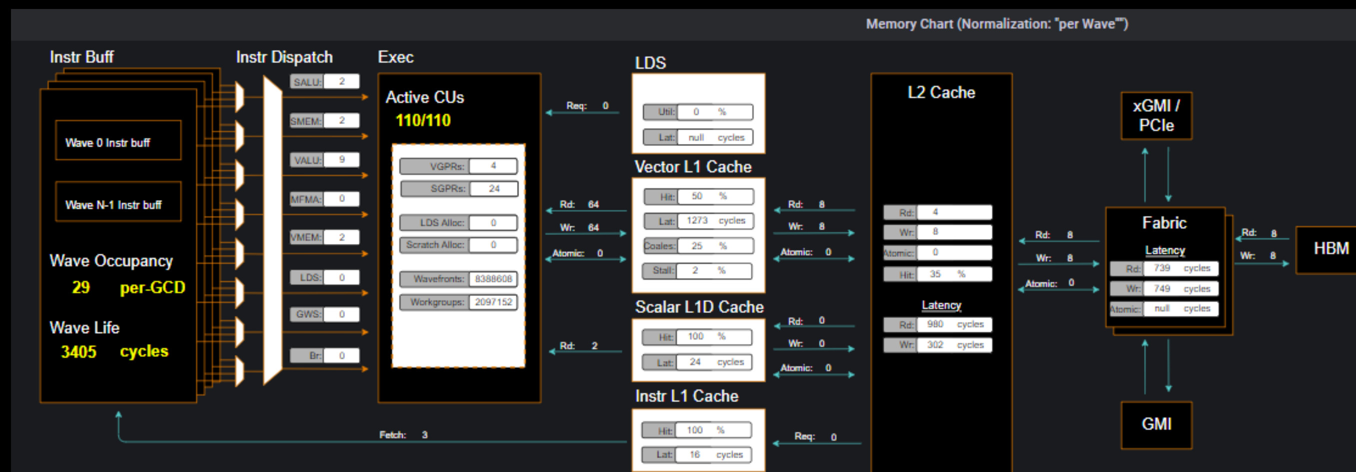
When assessing performance difference between two workloads (current and baseline), it's good to know the differences between underlying systems.

AMD
together we advance_

[AMD Official Use Only - General]

Initial assessment with kernel statistics

- Initial Assessment
  - Instruction/data flow
  - Speed-of-Light (SOL)
- Omniperf tooling support
  - System SOL
  - Memory Chart
  - Kernel statistics



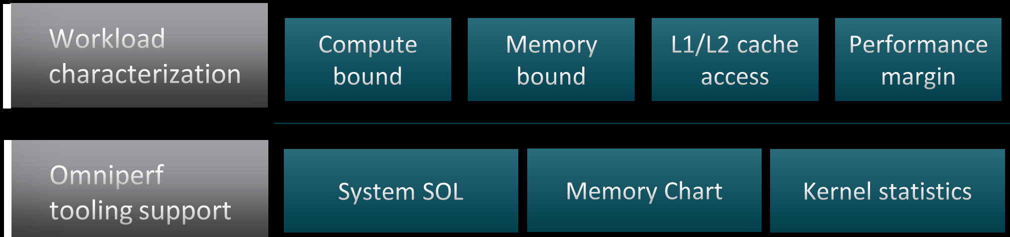
Metric	Avg	Unit	Theoretical Max	Pct-of-Peak
VALU FLOPs	0	GFLOP	23,936	0%
VALU IOPs	433	GIOP	23,936	2%
MFMA FLOPs (BF16)	0	GFLOP	95,744	0%
MFMA FLOPs (F16)	0	GFLOP	191,488	0%
MFMA FLOPs (F32)	0	GFLOP	47,872	0%
MFMA FLOPs (F64)	0	GFLOP	47,872	0%
MFMA IOPs (Int8)	0	GIOP	191,488	0%
Active CUs	110	CUs	110	100%
SALU Util	3	pct	100	3%
VALU Util	8	pct	100	8%
MFMA Util	0	pct	100	0%
VALU Active Threads/Wave	64	Threads	64	100%
IPC - Issue	1	Instr/cycle	5	20%
LDS BW	0	GB/sec	23,936	0%
LDS Bank Conflict		Conflicts/access	32	
Instr Cache Hit Rate	100	pct	100	100%

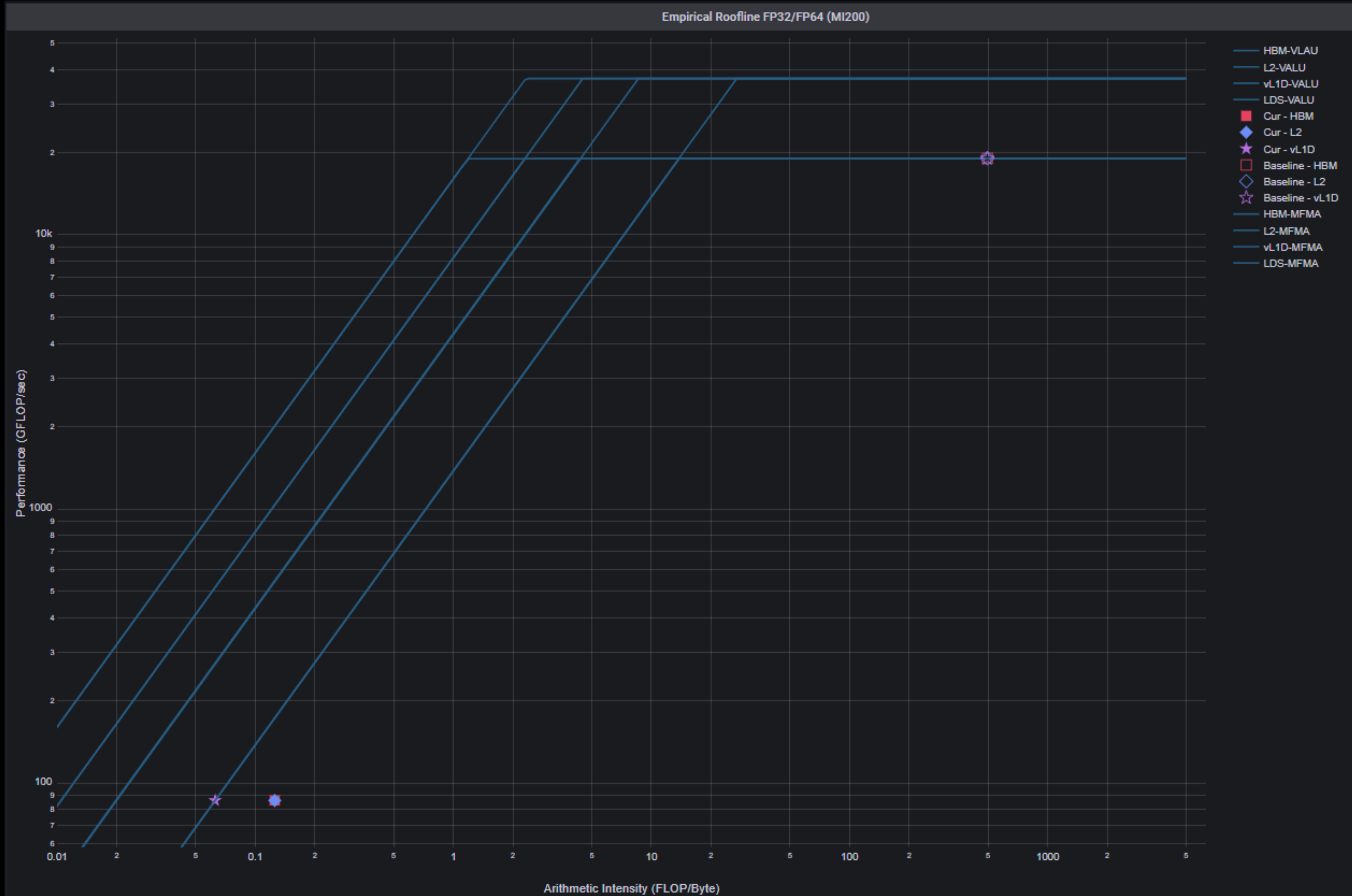
Speed of Light

AMD together we advance_

[AMD Official Use Only - General]

Roofline: the first-step characterization of workload performance





Top Kernels

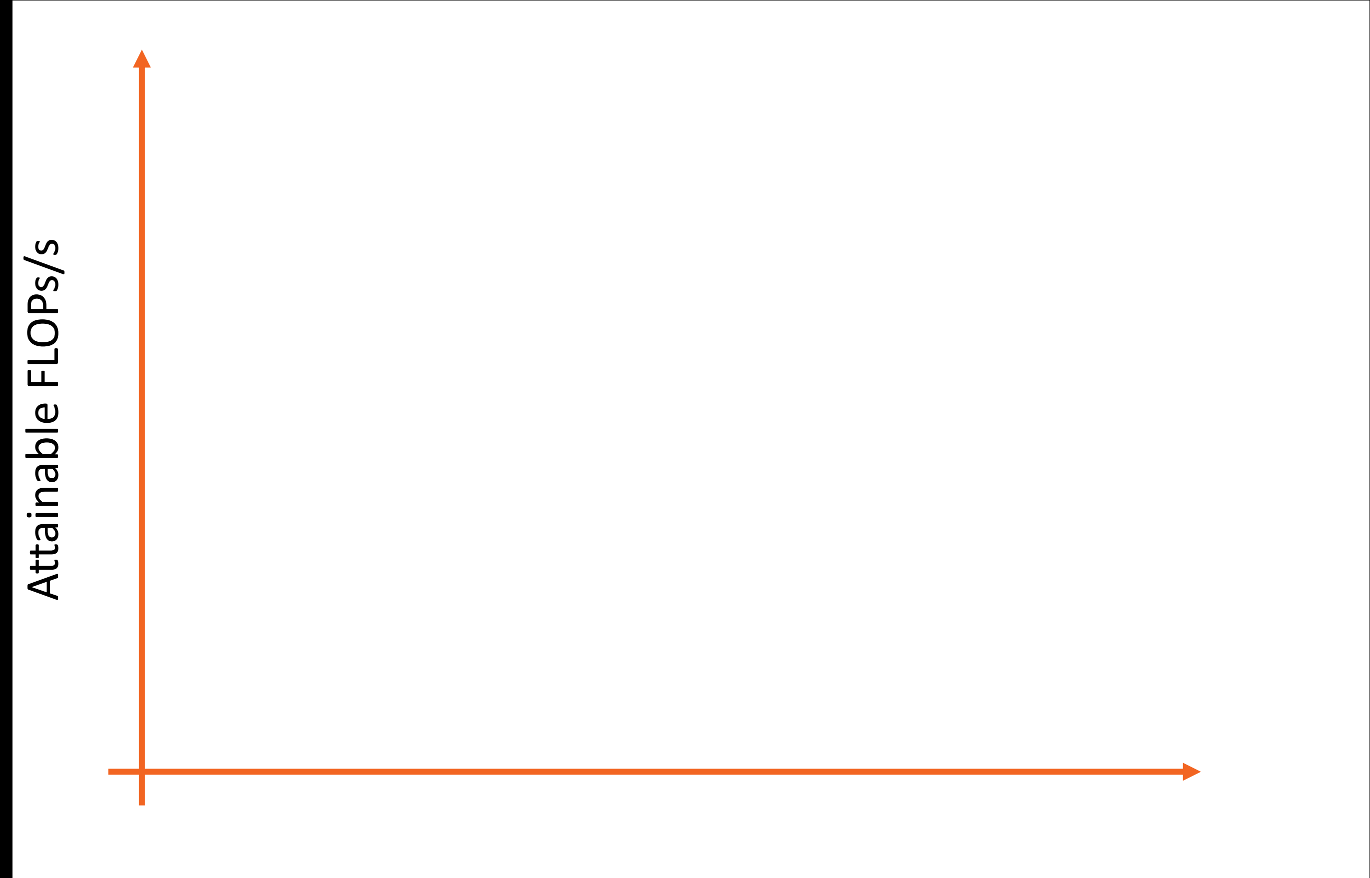
Name	Calls	Performance	HBM BW	Total Duration	Avg Duration	AI (Vector L1D Cache	AI (L2 Cache)	AI (HBM)	Total FLOPs	VALU FLOPs	MFMA FLOPs (F16)	MFMA FLOPs (BF16)
void dot_kernel<doubl...	100	86.5 GFLOPS	689 GB/s	244 ms	2.44 ms	0.063	0.126	0.126	210,583,552	210,583,552	0	0
void triad_kernel<dou...	100	111 GFLOPS	1.33 TB/s	189 ms	1.89 ms	0.042	0.083	0.083	209,715,200	209,715,200	0	0
void add_kernel<doubl...	100	55.7 GFLOPS	1.34 TB/s	188 ms	1.88 ms	0.021	0.042	0.042	104,857,600	104,857,600	0	0
void copy_kernel<dou...	100	0 GFLOPS	1.37 TB/s	122 ms	1.22 ms	0	0	0	0	0	0	0
void mul_kernel<doubl...	100	86.1 GFLOPS	1.38 TB/s	122 ms	1.22 ms	0.031	0.063	0.063	104,857,600	104,857,600	0	0

AMD
together we advance_

Background - What is a roofline?

Background – What is Roofline

- Attainable FLOPs/s
  - FLOPs/s rate as measured empirically on a given device
  - FLOP = floating point operation
  - FLOP counts for common operations
    - Add: 1 FLOP
    - Mul: 1 FLOP
    - FMA: 2 FLOP
  - FLOPs/s = Number of floating-point operations performed per second



AMD
together we advance_

[AMD Official Use Only - General]

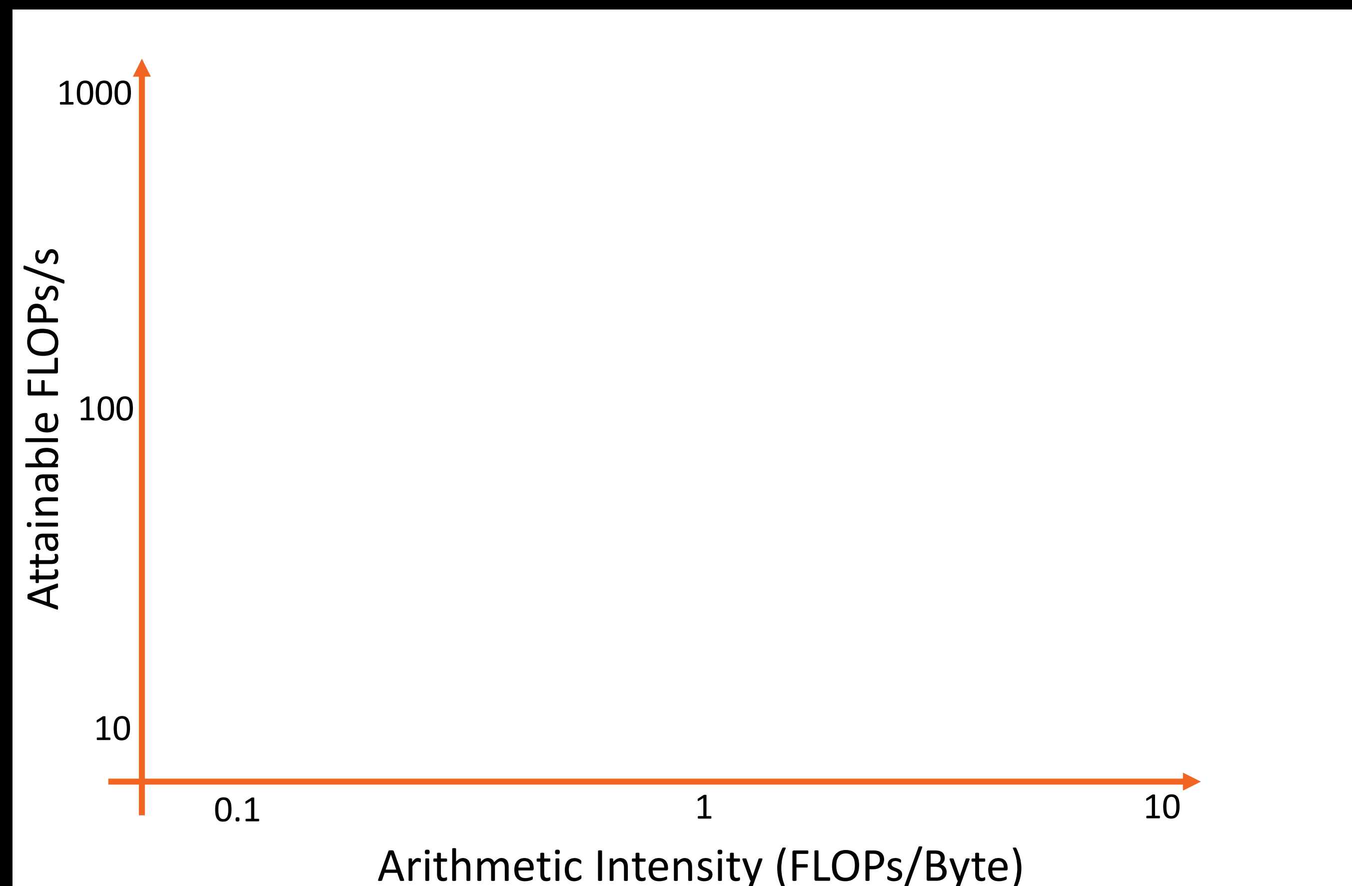
Background – What is Roofline

- Arithmetic Intensity (AI)
  - characteristic of the workload indicating how much compute (FLOPs) is performed per unit of data movement (Byte)
  - Ex: x[i] = y[i] + c
    - FLOPs = 1
    - Bytes = 1xRD + 1xWR = 4 + 4 = 8
    - AI = 1 / 8

AMD
together we advance_

Background – What is Roofline

- Log-Log plot
  - makes it easy to doodle, extrapolate performance along Moore's Law, etc...

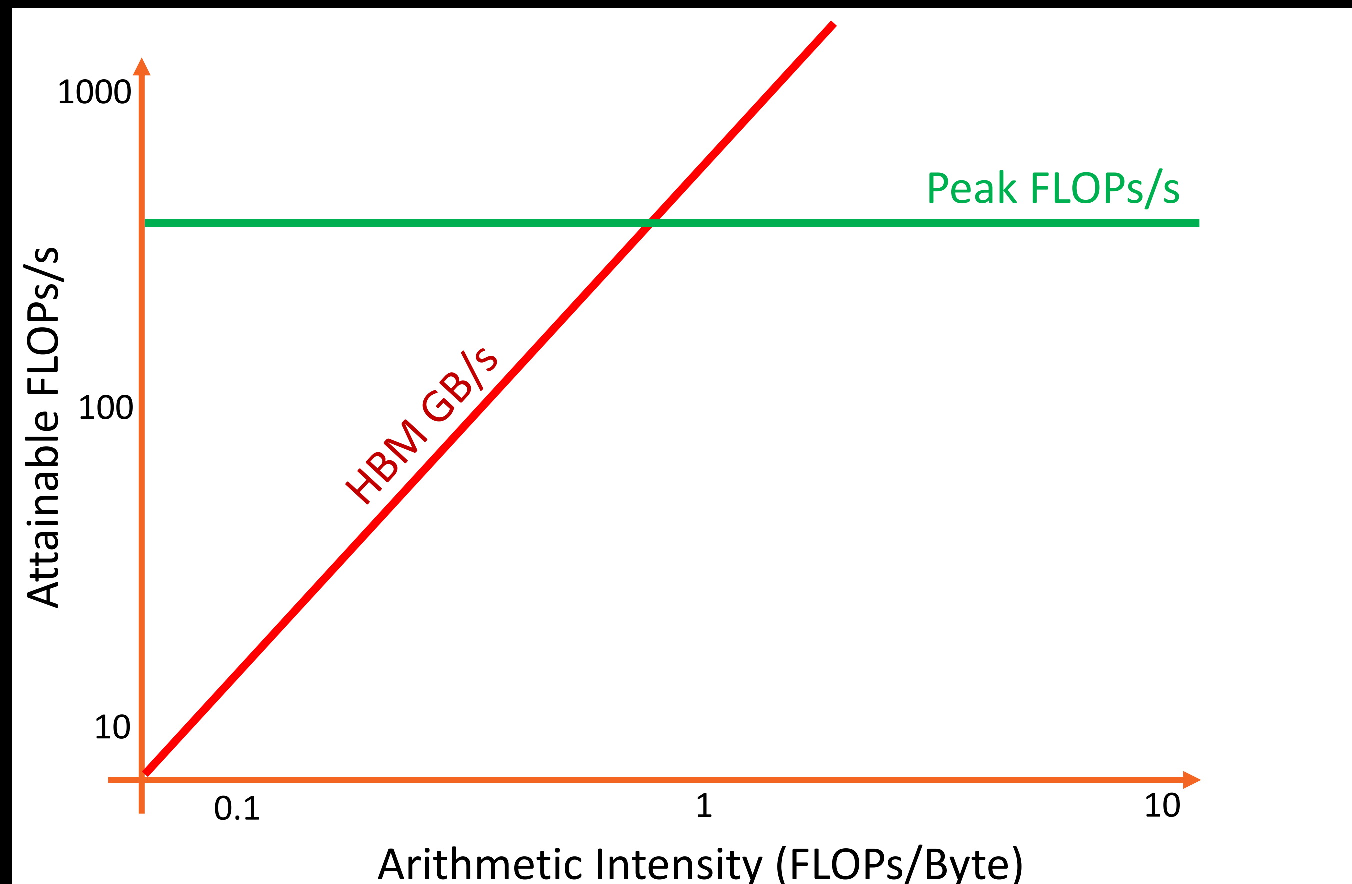


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is Roofline

- Roofline Limiters
  - Compute
    - Peak FLOPs/s
  - Memory BW
    - AI * Peak GB/s
- Note:
  - These are empirically measured values
  - Different SKUs will have unique plots
  - Individual devices within a SKU will have slightly different plots based on thermal solution, system power, etc.
  - Omniperf uses suite of simple kernels to empirically derive these values
  - These are NOT theoretical values indicating peak performance under "unicorn" conditions

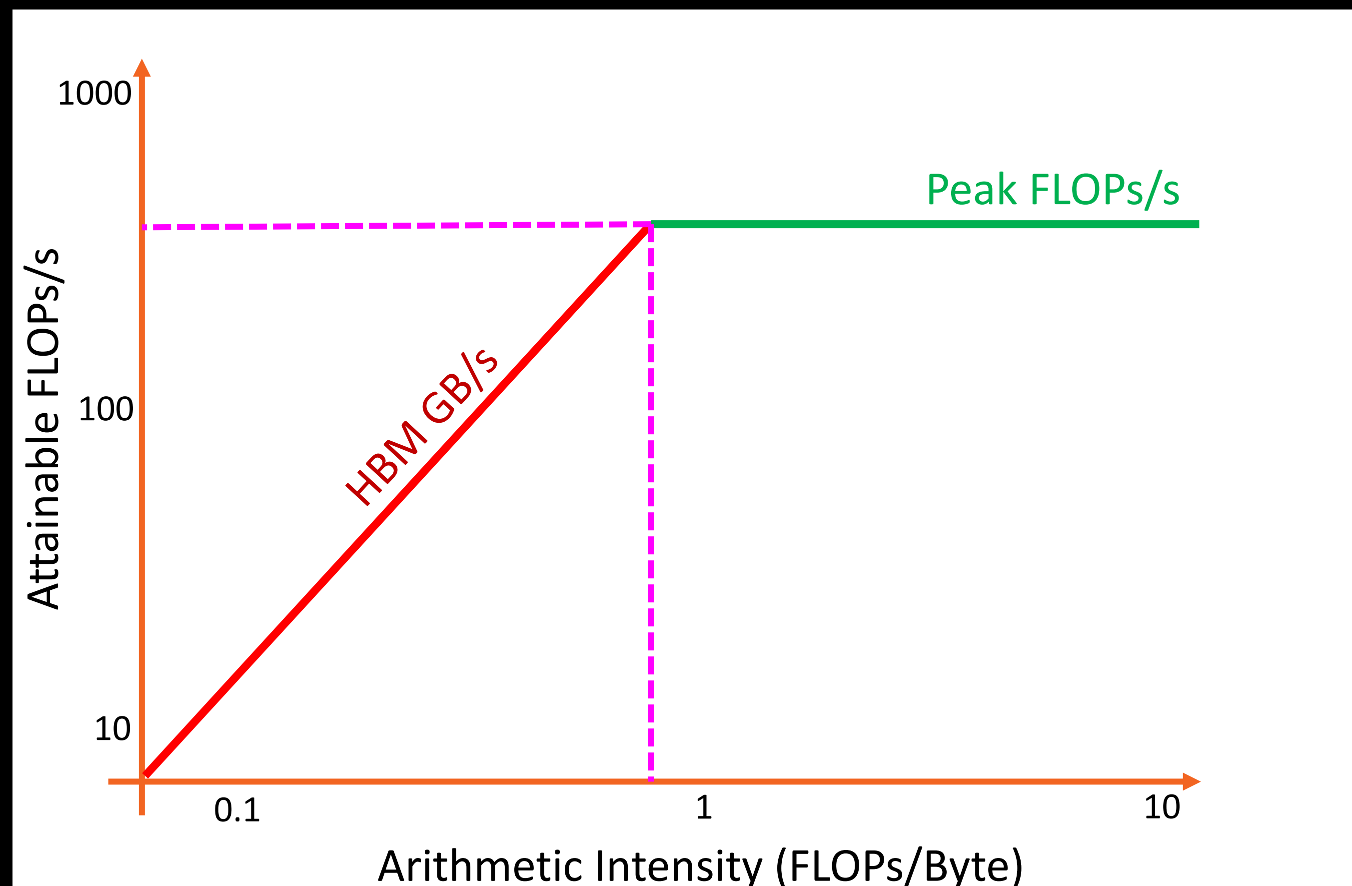


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is Roofline

- Attainable FLOPs/s =
  - $min \begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
  - Typical machine balance: 5-10 FLOPs/B
    - **40-80** FLOPs per double to exploit compute capability
  - MI250x machine balance: ~16 FLOPs/B
    - **128** FLOPs per double to exploit compute capability

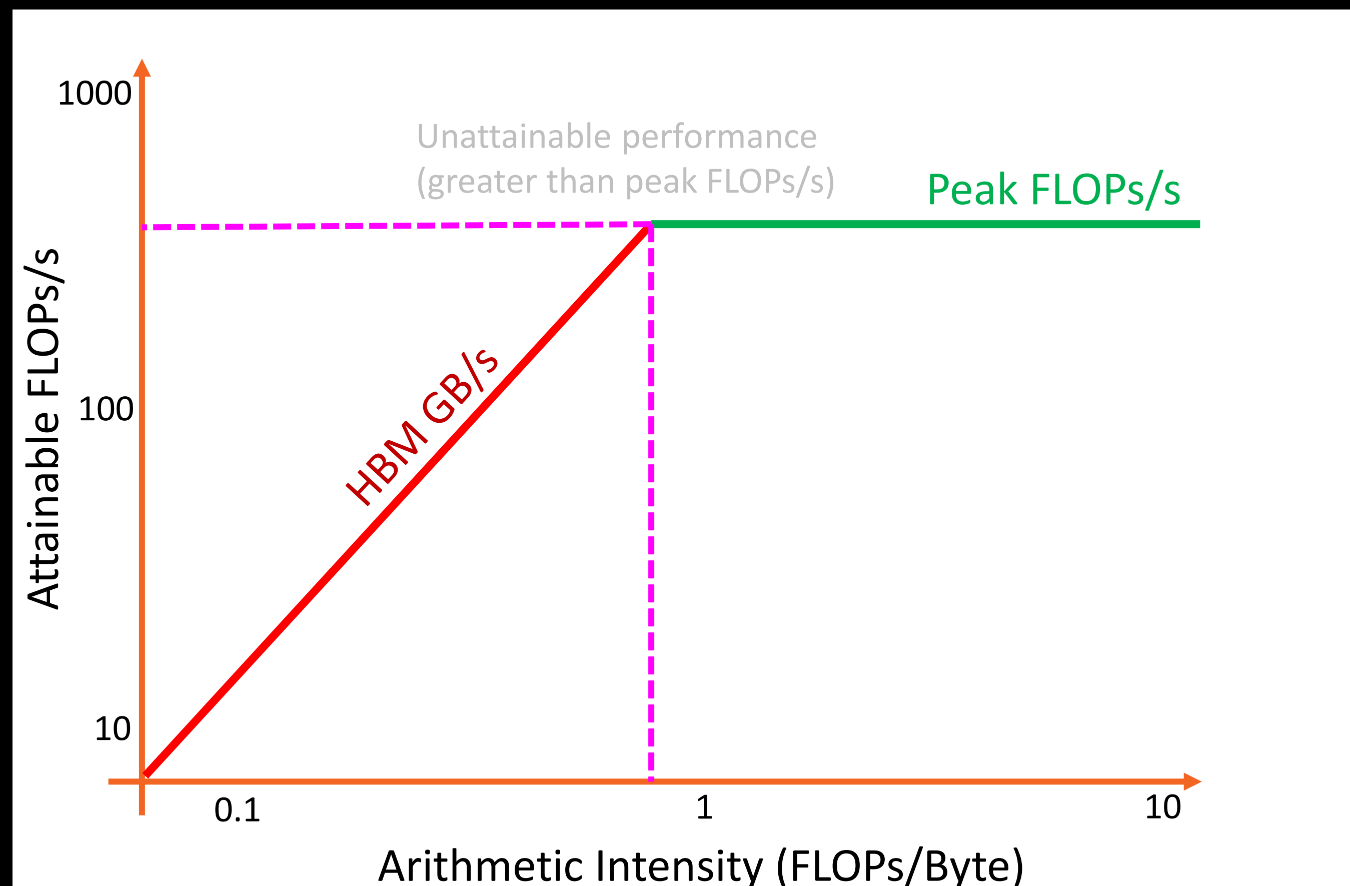


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is Roofline

- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute

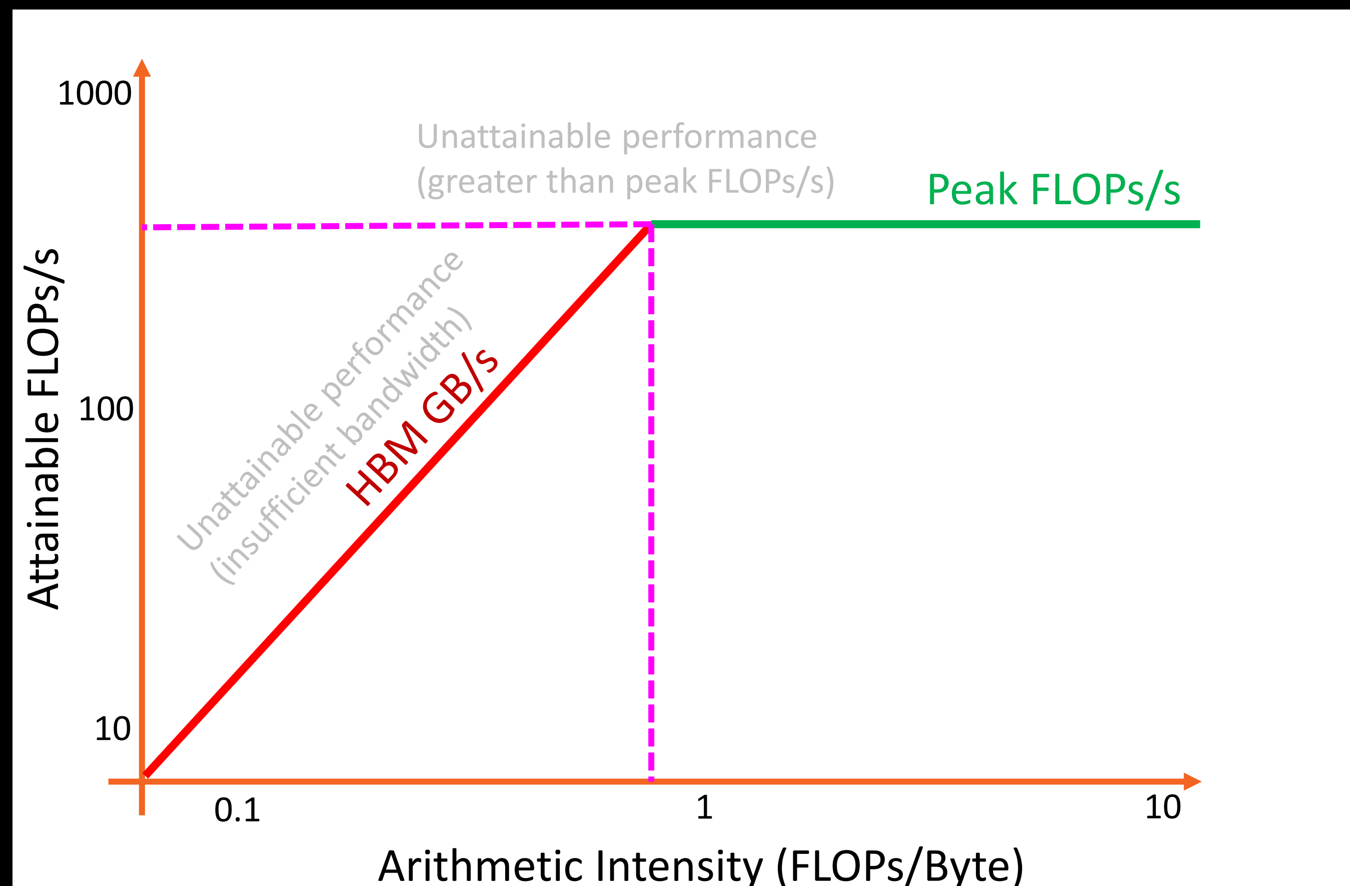


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is Roofline

- Attainable FLOPs/s =
  - $min \begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth

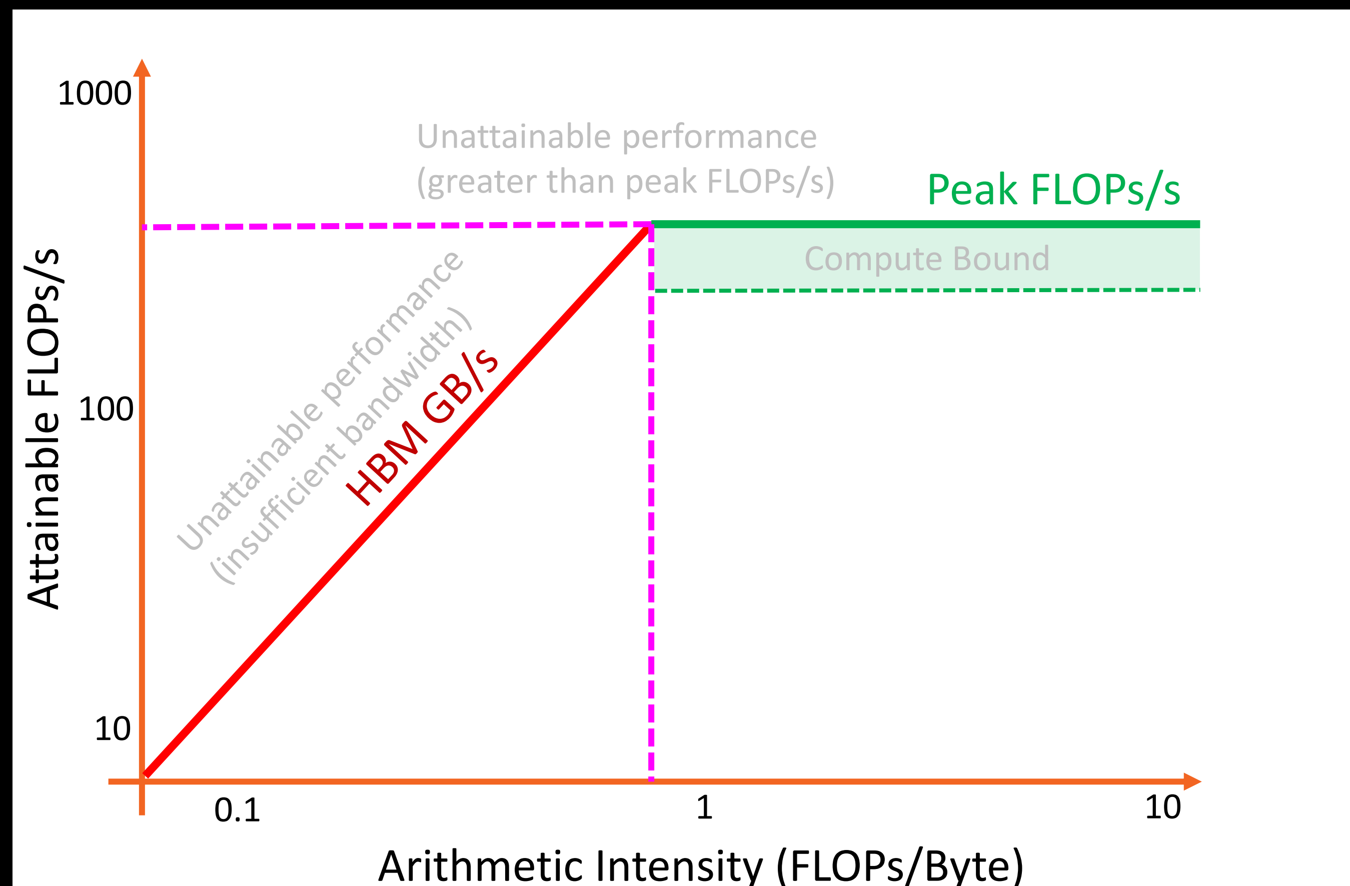


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is Roofline

- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound

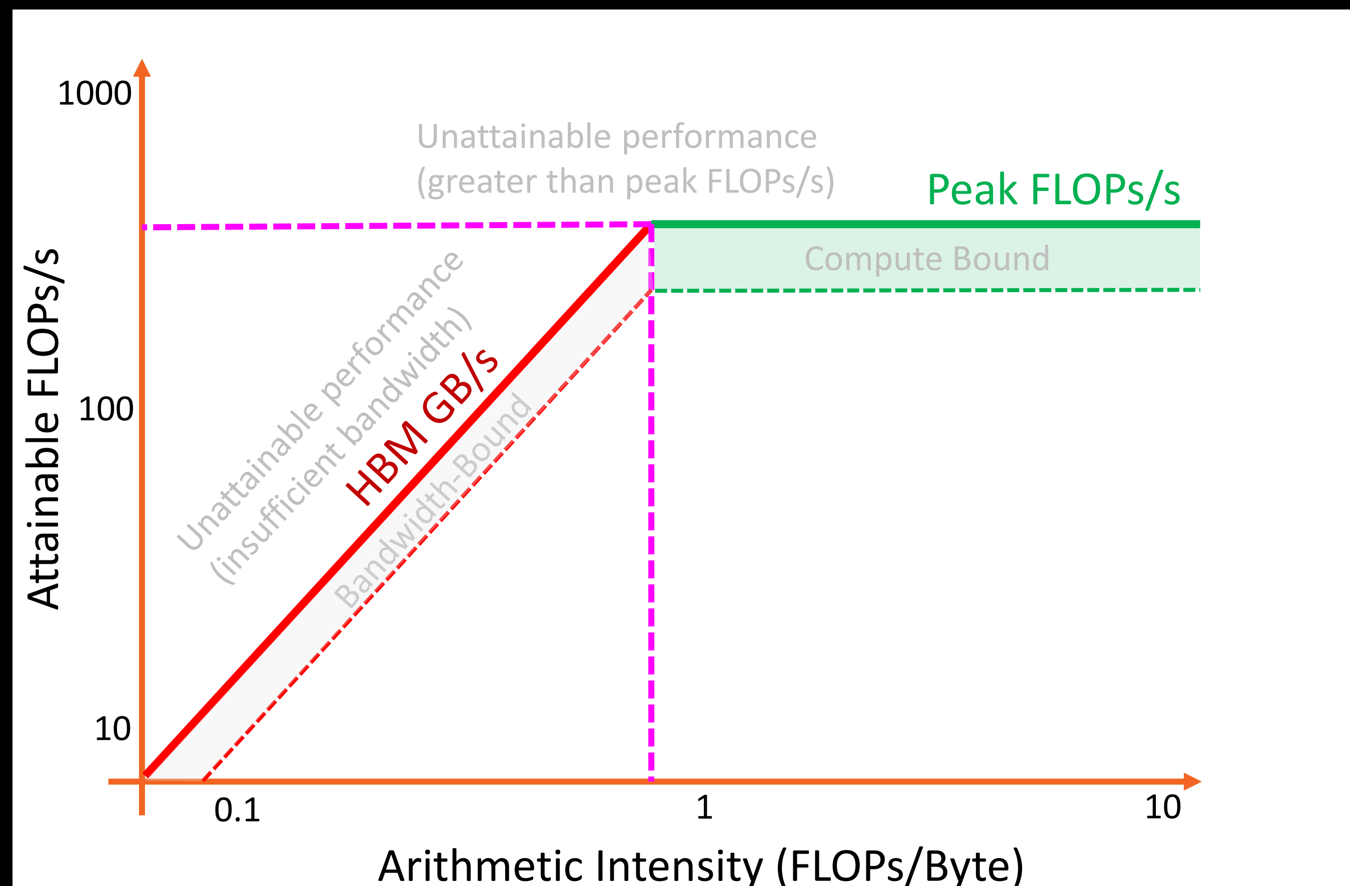


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is Roofline

- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound
  - Bandwidth Bound

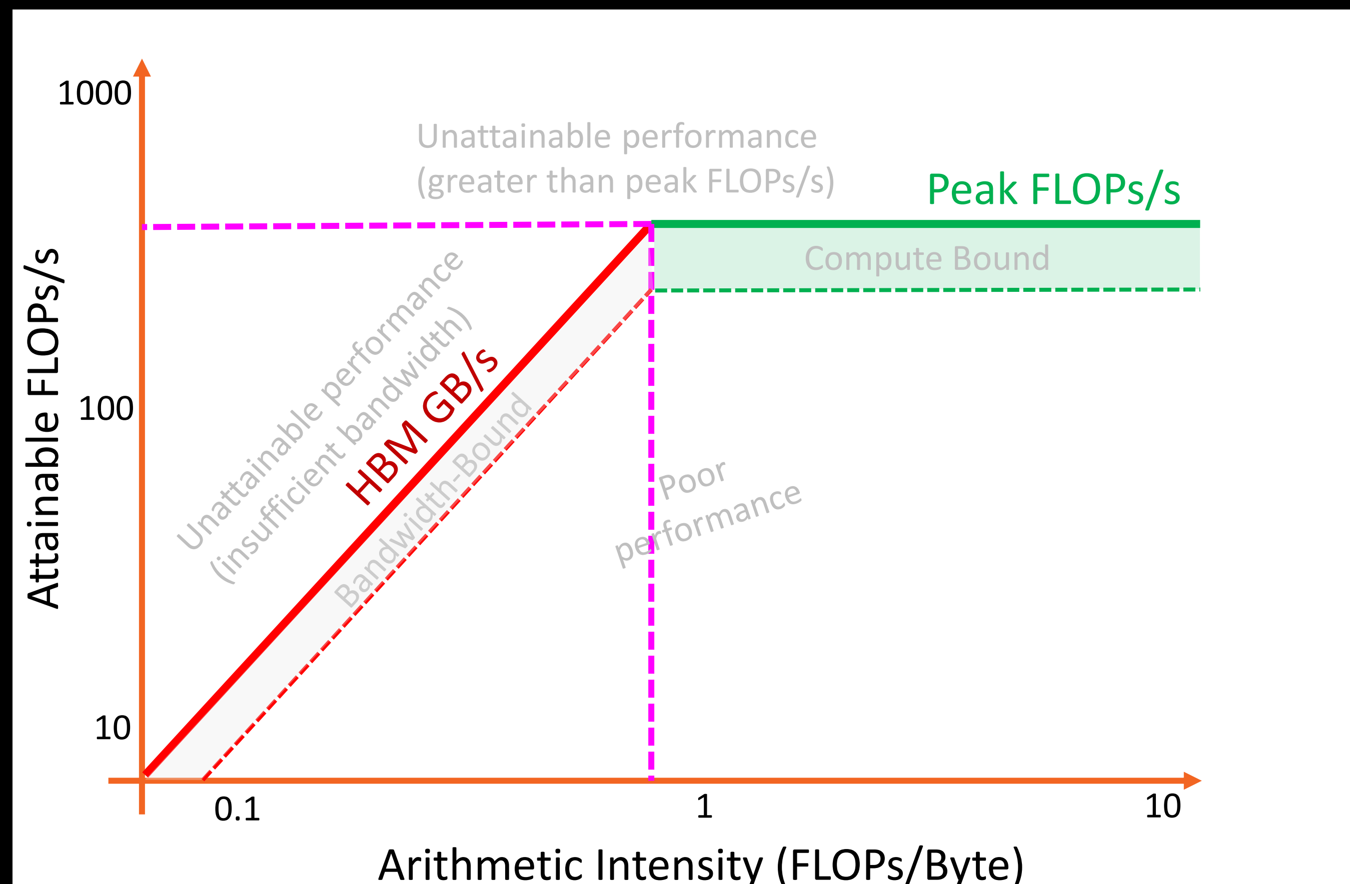


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is Roofline

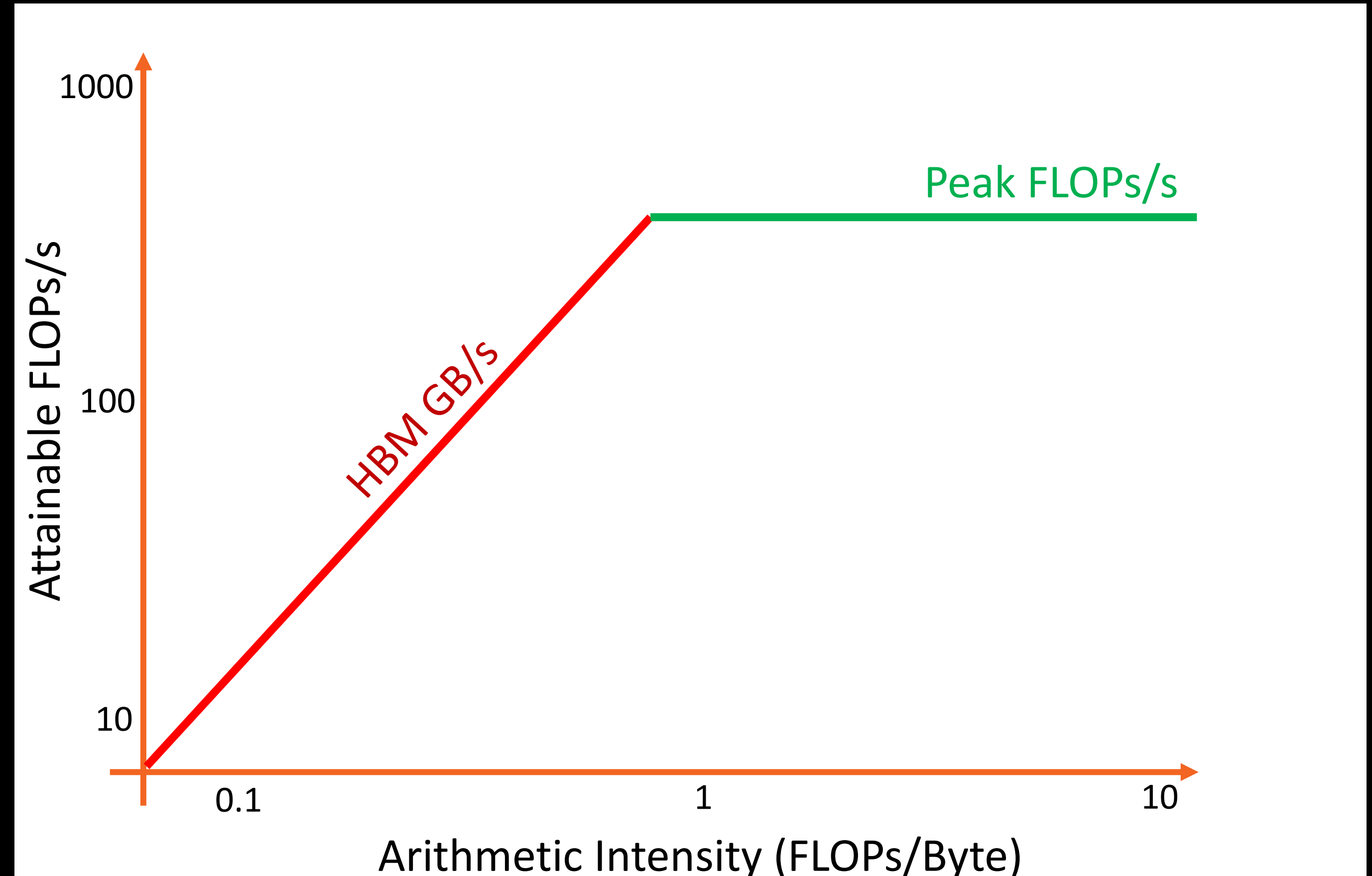
- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Machine Balance:
  - Where $AI = \frac{Peak\ FLOPs/s}{Peak\ GB/s}$
- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound
  - Bandwidth Bound
  - Poor Performance



AMD
together we advance_

Background – What is Roofline

- Attainable FLOPs/s =
  - $min\begin{cases} Peak\ FLOPs/s \\ AI * Peak\ GB/s \end{cases}$
- Final result is a single roofline plot presenting the peak attainable performance (in terms of FLOPs/s) on a given device based on the arithmetic intensity of any potential workload
- We have an application independent way of measuring and comparing performance on any platform

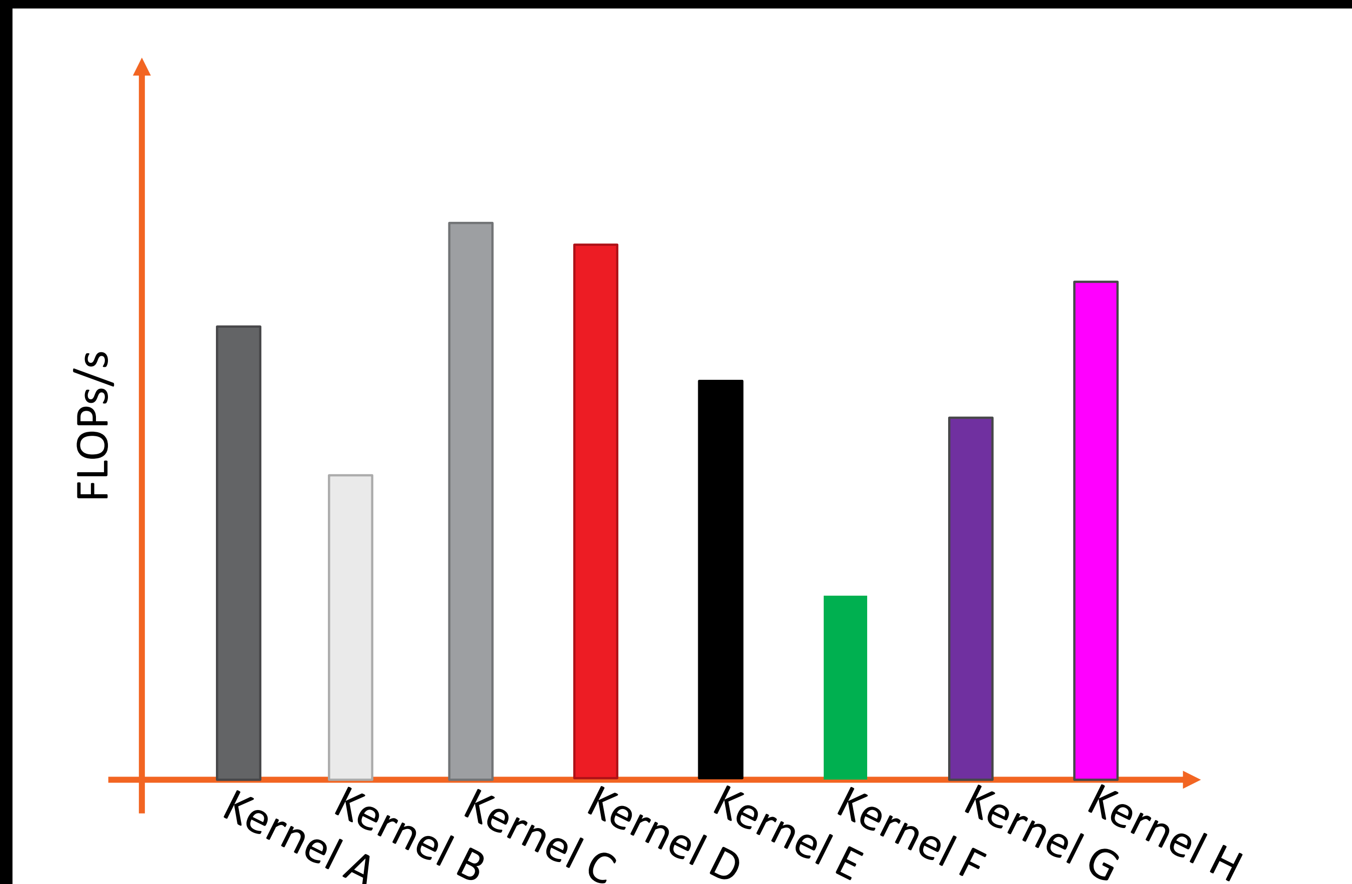


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is "Good" Performance?

- Example:
  - We run a number of kernels and measure FLOPs/s

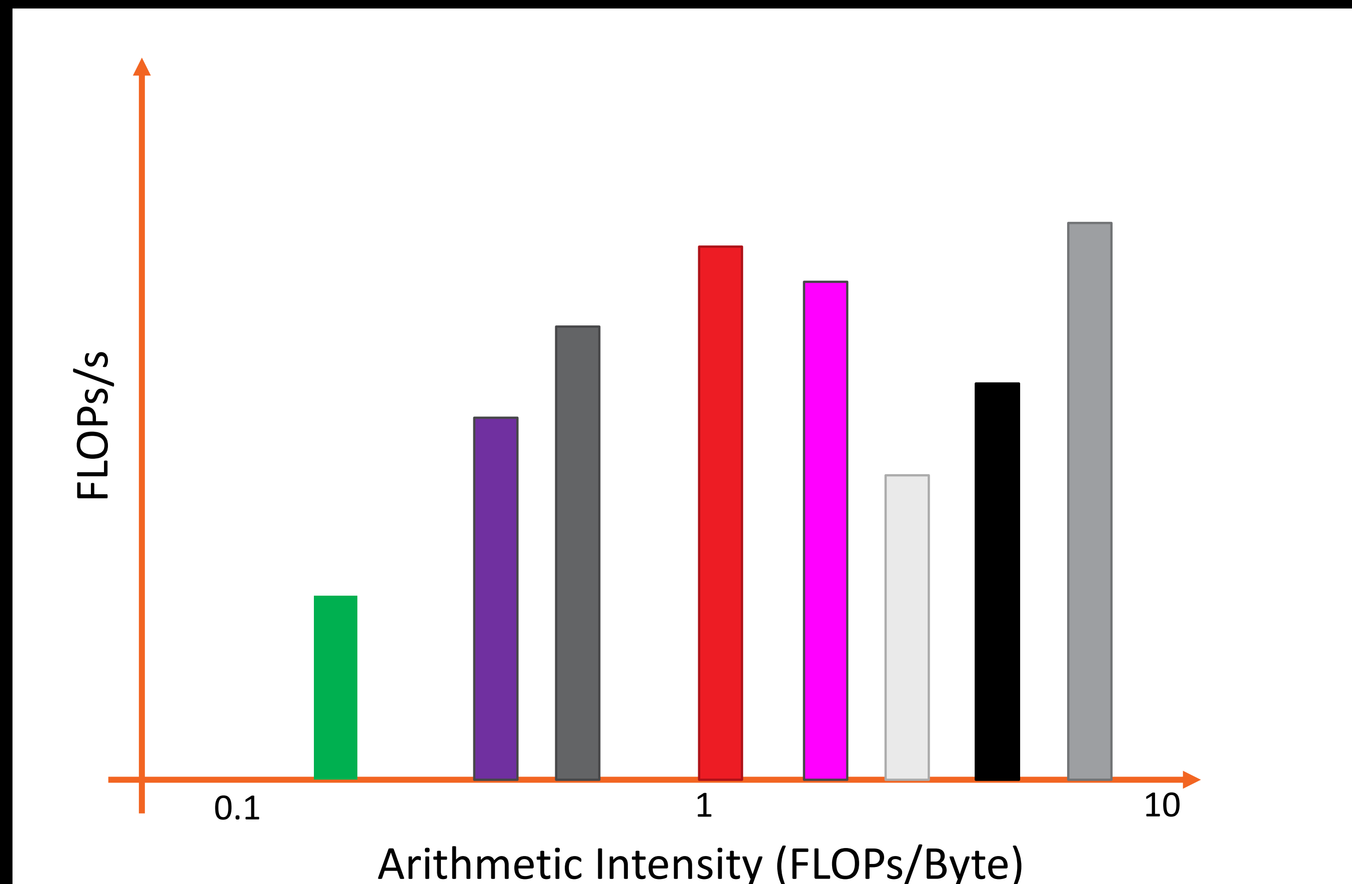


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is “Good” Performance?

- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity

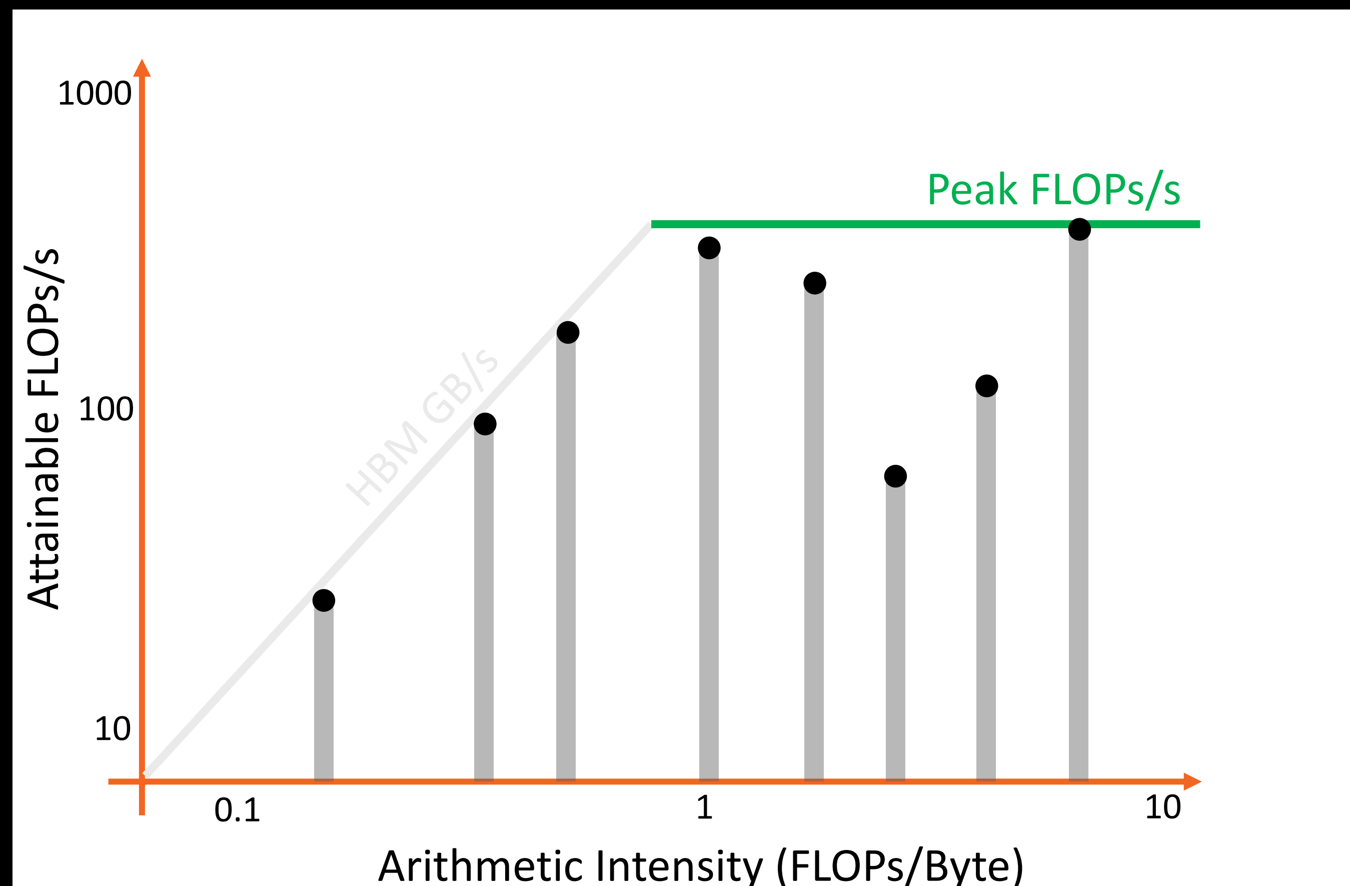


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is “Good” Performance?

- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity
  - Compare performance relative to hardware capabilities

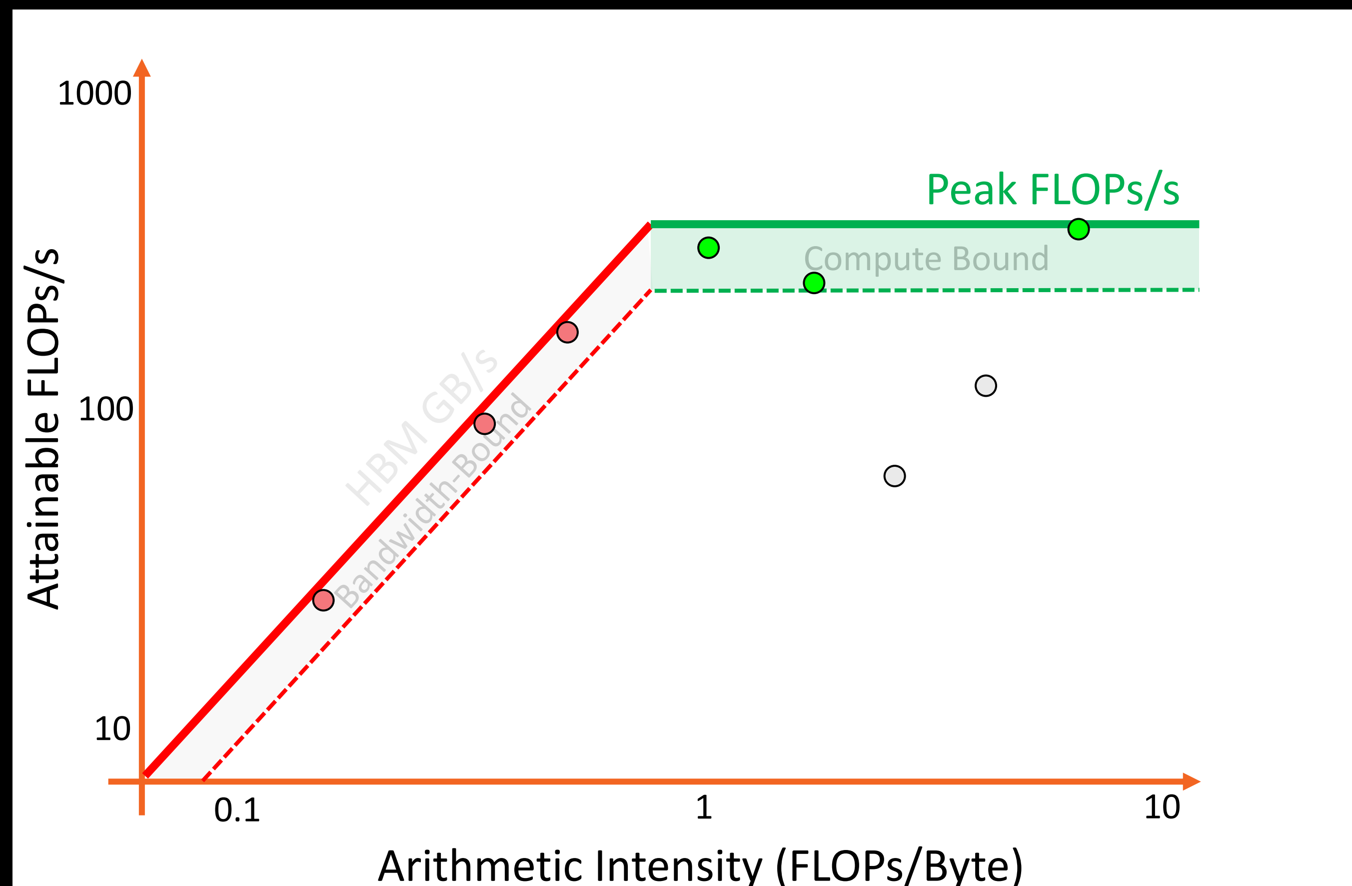


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is "Good" Performance?

- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity
  - Compare performance relative to hardware capabilities
  - Kernels near the roofline are making good use of computational resources
    - Kernels can have low performance (FLOPS/s), but make good use of BW

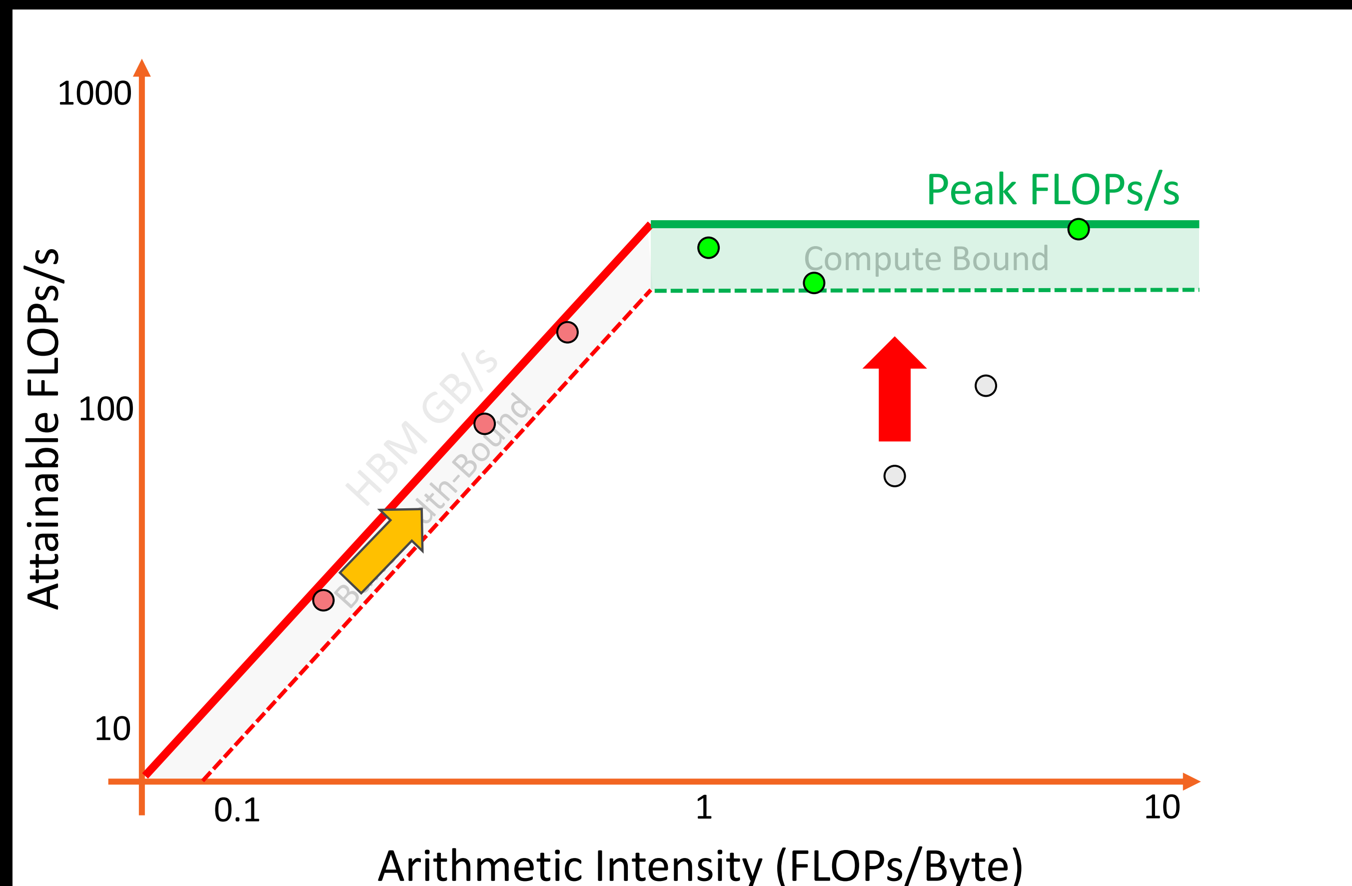


AMD
together we advance_

[AMD Official Use Only - General]

Background – What is "Good" Performance?

- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity
  - Compare performance relative to hardware capabilities
  - Kernels near the roofline are making good use of computational resources
    - Kernels can have low performance (FLOPS/s), but make good use of BW
  - Increase arithmetic intensity when bandwidth limited
    - Reducing data movement increases AI
  - Kernels not near the roofline *should** have optimizations that can be made to get closer to the roofline

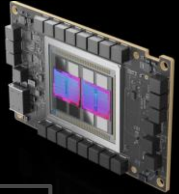


AMD
together we advance_

Roofline Calculations on AMD Instinct™ MI200 GPUs

[AMD Official Use Only - General]

Overview - AMD Instinct™ MI200 Architecture



Graphics Compute Die (GCD)

Compute Unit
Local Data Share (LDS)
Scalar Unit | SIMD0 | SIMD1 | SIMD2 | SIMD3
SGPR | VGPR | VGPR | VGPR | VGPR
Vector L1 Data Cache (vL1D)

Compute Unit
Local Data Share (LDS)
Scalar Unit | SIMD0 | SIMD1 | SIMD2 | SIMD3
SGPR | VGPR | VGPR | VGPR | VGPR
Vector L1 Data Cache (vL1D)

… …

Compute Unit
Local Data Share (LDS)
Scalar Unit | SIMD0 | SIMD1 | SIMD2 | SIMD3
SGPR | VGPR | VGPR | VGPR | VGPR
Vector L1 Data Cache (vL1D)

L2 Cache (L2)

(Peer GCD) ⟷ Data Fabric ⟷ Remote Socket (CPU, GPU)

Memory Controller

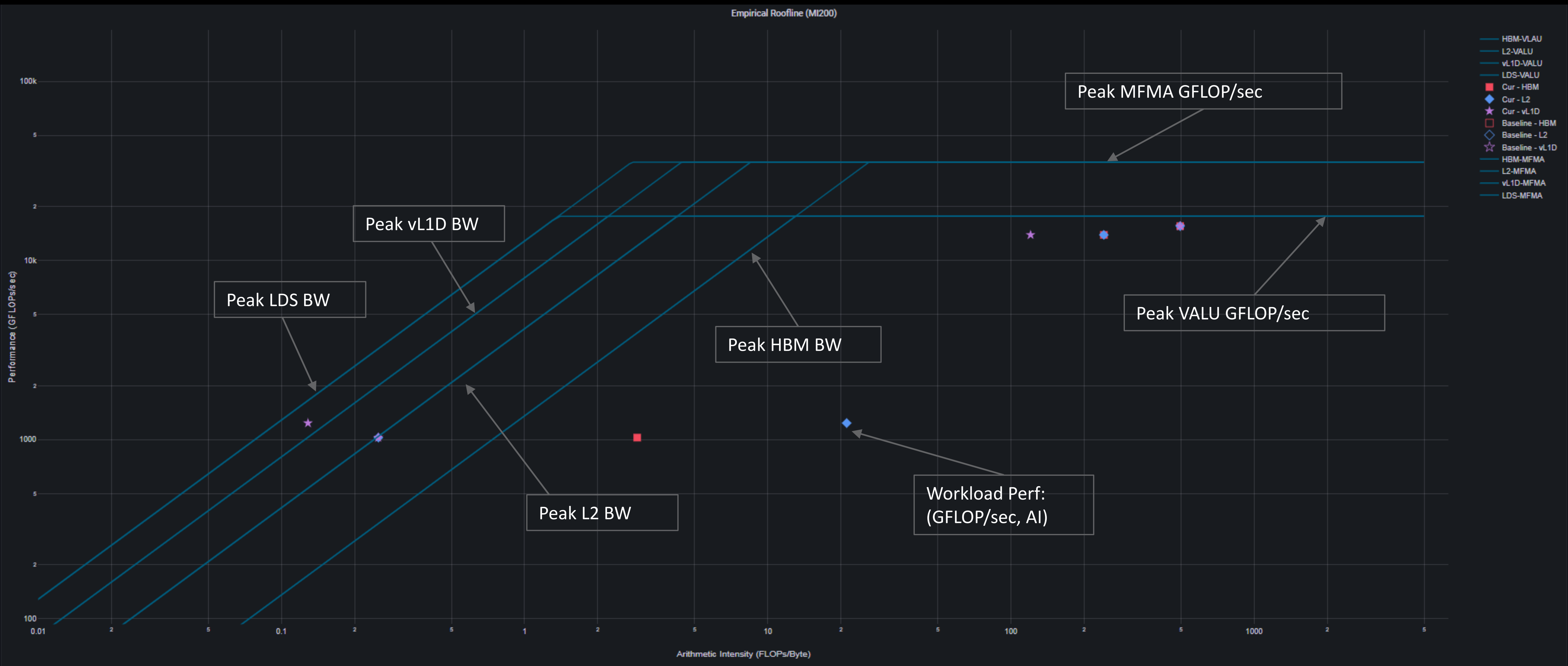
(GCD1) HBM Memory

(GCD2) HBM Memory

AMD
together we advance_

[AMD Official Use Only - General]

Empirical Hierarchical Roofline on MI200 - Overview



AMD
together we advance_

Empirical Hierarchical Roofline on MI200 – Roofline Benchmarking

- Empirical Roofline Benchmarking
  - Measure achievable Peak FLOPS
    - VALU: F32, F64
    - MFMA: F16, BF16, F32, F64
  - Measure achievable Peak BW
    - LDS
    - Vector L1D Cache
    - L2 Cache
    - HBM

```
10:57:35 amd@node-bp126-014a utils ±|master ✗|→ ./roofline
Total detected GPU devices: 2
GPU Device 0: Profiling...
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
HBM BW, GPU ID: 0, workgroupSize:256, workgroups:2097152, experiments:100, Total Bytes=8589934592, Duration=6.2 ms, Mean=1382.7 GB/sec, stdev=2.6 GB/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
L2 BW, GPU ID: 0, workgroupSize:256, workgroups:8192, experiments:100, Total Bytes=687194767360, Duration=157.3 ms, Mean=4321.3 GB/sec, stdev=59.1 GB/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
L1 BW, GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total Bytes=26843545600, Duration=3.3 ms, Mean=8262.6 GB/sec, stdev=5.9 GB/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
LDS BW, GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total Bytes=33554432000, Duration=1.8 ms, Mean=18780.4 GB/sec, stdev=33.0 GB/s
nSize:134217728, 268435456000
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak FLOPs (FP32), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=274877906944, Duration=14.482 ms, Mean=18977.7 GFLOPs/sec, stdev=3.6 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak FLOPs (FP64), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=137438953472, Duration=7.5 ms, Mean=18336.156250.1 GFLOPs/sec, stdev=5.0 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak MFMA FLOPs (BF16), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=2147483648000, Duration=14.0 ms, Mean=153763.7 GFLOPs/sec, stdev=61.0 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak MFMA FLOPs (F16), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=2147483648000, Duration=14.5 ms, Mean=147890.9 GFLOPs/sec, stdev=32.2 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak MFMA FLOPs (F32), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=536870912000, Duration=14.4 ms, Mean=37200.4 GFLOPs/sec, stdev=9.3 GFLOPs/s
 99% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| ]
Peak MFMA FLOPs (F64), GPU ID: 0, workgroupSize:256, workgroups:16384, experiments:100, Total FLOPS=268435456000, Duration=7.3 ms, Mean=36978.4 GFLOPs/sec, stdev=10.0 GFLOPs/s
```

- Internally developed micro benchmark algorithms
  - Peak VALU FLOP: axpy
  - Peak MFMA FLOP: Matrix multiplication based on MFMA intrinsic
  - Peak LDS/vL1D/L2 BW: Pointer chasing
  - Peak HBM BW: Streaming copy

AMD
together we advance_

[AMD Official Use Only - General]

Empirical Hierarchical Roofline on MI200 – Perfmon counters

- Weight
  - ADD: 1
  - MUL: 1
  - FMA: 2
  - Transcendental: 1
- FLOP Count
  - VALU: derived from VALU math instructions (assuming 64 active threads)
  - MFMA: count FLOP directly, in unit of 512
- Transcendental Instructions (7 in total)
  - e^x, $\log(x)$: F16, F32
  - $\frac{1}{x}$, $\sqrt{x}$, $\frac{1}{\sqrt{x}}$: F16, F32, F64
  - $\sin x$, $\cos x$: F16, F32
- Profiling Overhead
  - Require 3 application replays

```
v_rcp_f64_e32 v[4:5], v[2:3]
v_sin_f32_e32 v2, v2
v_cos_f32_e32 v2, v2
v_rsq_f64_e32 v[6:7], v[2:3]
v_sqrt_f32_e32 v3, v2
v_log_f32_e32 v2, v2
v_exp_f32_e32 v2, v2
```

ID	HW Counter	Category
1	SQ_INSTS_VALU_ADD_F16	FLOP counter
2	SQ_INSTS_VALU_MUL_F16	FLOP counter
3	SQ_INSTS_VALU_FMA_F16	FLOP counter
4	SQ_INSTS_VALU_TRANS_F16	FLOP counter
5	SQ_INSTS_VALU_ADD_F32	FLOP counter
6	SQ_INSTS_VALU_MUL_F32	FLOP counter
7	SQ_INSTS_VALU_FMA_F32	FLOP counter
8	SQ_INSTS_VALU_TRANS_F32	FLOP counter
9	SQ_INSTS_VALU_ADD_F64	FLOP counter
10	SQ_INSTS_VALU_MUL_F64	FLOP counter
11	SQ_INSTS_VALU_FMA_F64	FLOP counter
12	SQ_INSTS_VALU_TRANS_F64	FLOP counter
13	SQ_INSTS_VALU_INT32	IOP counter
14	SQ_INSTS_VALU_INT64	IOP counter
15	SQ_INSTS_VALU_MFMA_MOPS_I8	IOP counter

ID	HW Counter	Category
16	SQ_INSTS_VALU_MFMA_MOPS_F16	FLOP counter
17	SQ_INSTS_VALU_MFMA_MOPS_BF16	FLOP counter
18	SQ_INSTS_VALU_MFMA_MOPS_F32	FLOP counter
19	SQ_INSTS_VALU_MFMA_MOPS_F64	FLOP counter
20	SQ_LDS_IDX_ACTIVE	LDS Bandwidth
21	SQ_LDS_BANK_CONFLICT	LDS Bandwidth
22	TCP_TOTAL_CACHE_ACCESSES_sum	vL1D Bandwidth
23	TCP_TCC_WRITE_REQ_sum	L2 Bandwidth
24	TCP_TCC_ATOMIC_WITH_RET_REQ_sum	L2 Bandwidth
25	TCP_TCC_ATOMIC_WITHOUT_RET_REQ_sum	L2 Bandwidth
26	TCP_TCC_READ_REQ_sum	L2 Bandwidth
27	TCC_EA_RDREQ_sum	HBM Bandwidth
28	TCC_EA_RDREQ_32B_sum	HBM Bandwidth
29	TCC_EA_WRREQ_sum	HBM Bandwidth
3	TCC_EA_WRREQ_64B_sum	HBM

[AMD Official Use Only - General]

Empirical Hierarchical Roofline on MI200 - Arithmetic

$\text{Total_FLOP} =$ 64 $*$ (SQ_INSTS_VALU_ADD_F16 + SQ_INSTS_VALU_MUL_F16 + SQ_INSTS_VALU_TRANS_F16 + 2 $*$ SQ_INSTS_VALU_FMA_F16)
+ 64 $*$ (SQ_INSTS_VALU_ADD_F32 + SQ_INSTS_VALU_MUL_F32 + SQ_INSTS_VALU_TRANS_F32 + 2 $*$ SQ_INSTS_VALU_FMA_F32)
+ 64 $*$ (SQ_INSTS_VALU_ADD_F64 + SQ_INSTS_VALU_MUL_F64 + SQ_INSTS_VALU_TRANS_F64 + 2 $*$ SQ_INSTS_VALU_FMA_F64)
+ 512 * SQ_INSTS_VALU_MFMA_MOPS_F16
+ 512 * SQ_INSTS_VALU_MFMA_MOPS_BF16
+ 512 * SQ_INSTS_VALU_MFMA_MOPS_F32
+ 512 * SQ_INSTS_VALU_MFMA_MOPS_F64

$\text{Total_IOP} =$ 64 $*$ (SQ_INSTS_VALU_INT32 + SQ_INSTS_VALU_INT64)

$LDS_{BW} =$ 32 $*$ 4 $*$ (SQ_LDS_IDX_ACTIVE – SQ_LDS_BANK_CONFLICT)

$vL1D_{BW} =$ 64 $*$ TCP_TOTAL_CACHE_ACCESSES_sum

$L2_{BW} =$ 64 $*$ TCP_TCC_READ_REQ_sum
+ 64 * TCP_TCC_WRITE_REQ_sum
+ 64 * (TCP_TCC_ATOMIC_WITH_RET_REQ_sum + TCP_TCC_ATOMIC_WITHOUT_RET_REQ_sum)

$HBM_{BW} =$ 32 * TCC_EA_RDREQ_32B_sum + 64 * (TCC_EA_RDREQ_sum - TCC_EA_RDREQ_32B_sum)
+ 32 * (TCC_EA_WRREQ_sum – TCC_EA_WRREQ_64B_sum) + 64 * TCC_EA_WRREQ_64B_sum

$$AI_{LDS} \frac{TOTAL_FLOP}{LDS_{BW}}$$

$$AI_{vL1D} \frac{TOTAL_FLOP}{vL1D_{BW}}$$

$$AI_{L2} \frac{TOTAL_FLOP}{L2_{BW}}$$

$$AI_{HBM} = \frac{TOTAL_FLOP}{HBM_{BW}}$$

** All calculations are subject to change*

AMD together we advance_

Empirical Hierarchical Roofline on MI200 - Manual Rocprof

- For those who like getting their hands dirty
- Generate input file
  - See example roof-counters.txt →
- Run rocprof

```
foo@bar:~$ rocprof -i roof-counters.txt --timestamp on ./myCoolApp
```

- Analyze results
  - Load *results.csv* output file in csv viewer of choice
  - Derive final metric values using equations on previous slide
- Profiling Overhead
  - Requires one application replay for each pmc line

```
## roof-counters.txt

# FP32 FLOPs
pmc: SQ_INSTS_VALU_ADD_F32 SQ_INSTS_VALU_MUL_F32 SQ_INSTS_VALU_FMA_F32 SQ_INSTS_VALU_TRANS_F32

# HBM Bandwidth
pmc: TCC_EA_RDREQ_sum TCC_EA_RDREQ_32B_sum TCC_EA_WRREQ_sum TCC_EA_WRREQ_64B_sum

# LDS Bandwidth
pmc: SQ_LDS_IDX_ACTIVE SQ_LDS_BANK_CONFLICT

# L2 Bandwidth
pmc: TCP_TCC_READ_REQ_sum TCP_TCC_WRITE_REQ_sum TCP_TCC_ATOMIC_WITH_RET_REQ_sum
TCP_TCC_ATOMIC_WITHOUT_RET_REQ_sum

# vL1D Bandwidth
pmc: TCP_TOTAL_CACHE_ACCESSES_sum
```


AMD
together we advance_

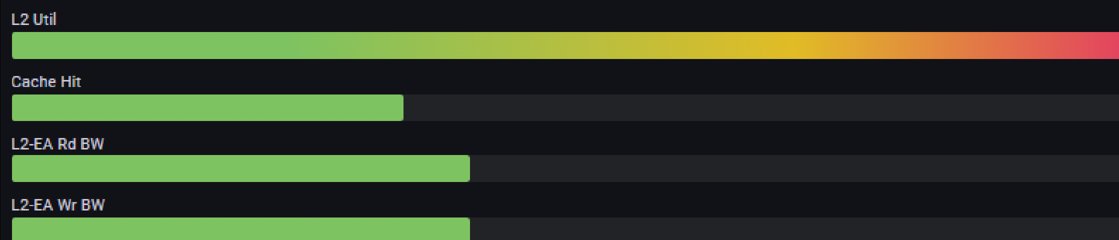
Omniperf Performance Analyzer (cont..)

[AMD Official Use Only - General]

Subsystem performance analysis

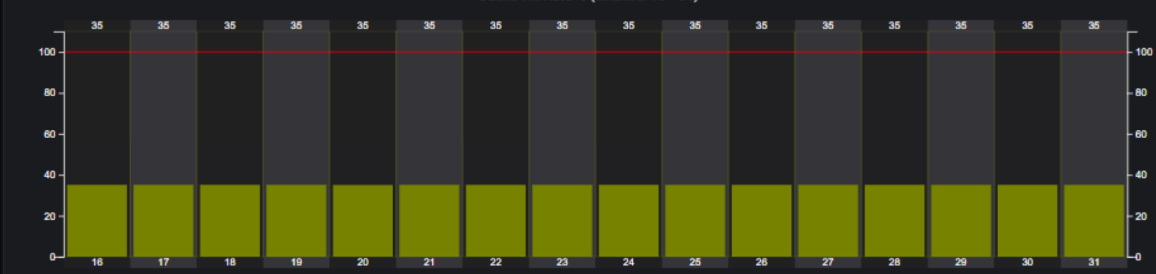
Memory subsystems: L2 Cache | HBM access | LDS | vL1D

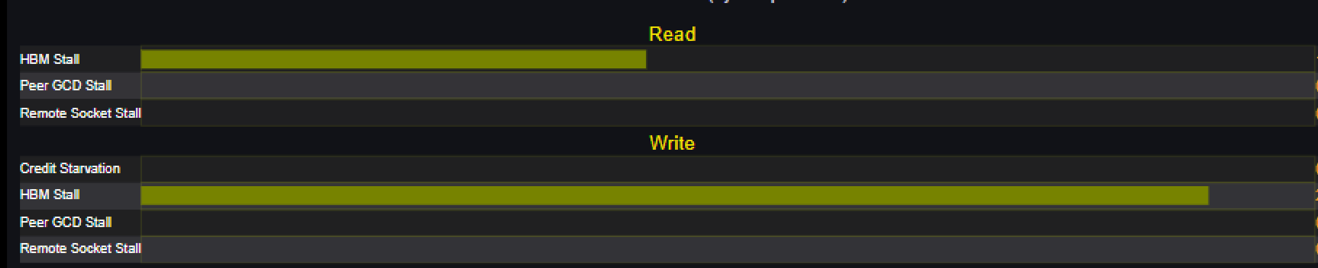
Omniperf tooling support: L2 Cache SOL | L2 fabric metrics | Per-channel statistics



L2 - Fabric Transactions

Metric	Avg	Min	Max	Unit
Read BW	693,148,700,953	664,565,016,054	695,197,543,698	Bytes per Sec
Write BW	692,659,558,092	664,096,634,666	694,705,946,653	Bytes per Sec
Read (32B)	0	0	0	Req per Sec
Read (Uncached 32...	2,304,240	1,434,649	2,370,898	Req per Sec
Read (64B)	10,830,448,452	10,383,828,376	10,862,461,620	Req per Sec
HBM Read	10,830,362,679	10,383,764,324	10,862,381,992	Req per Sec
Write (32B)	0	0	0	Req per Sec
Write (Uncached 32...	0	0	0	Req per Sec
Write (64B)	10,822,805,595	10,376,509,917	10,854,780,416	Req per Sec
HBM Write	10,822,801,389	10,376,488,102	10,854,762,613	Req per Sec
Read Latency	739	732	801	Cycles
Write Latency	749	737	784	Cycles
Atomic Latency				Cycles
Read Stall	3	2	3	pct
Write Stall	6	5	8	pct





AMD
together we advance_

[AMD Official Use Only - General]

Shader compute components

Shader compute | Wavefront life | Instruction mix | Floating/ Integer Ops | Compute pipeline

Instruction Mix

MFMA Arithmetic Instr Mix

MFMA Instr	Count
MFMA-I8	0
MFMA-F16	0
MFMA-BF16	0
MFMA-F32	0
MFMA-F64	995

Wavefront Runtime Stats

Metric	Avg	Min	Max	Unit
Kernel Time (Nanosec)	6,197,098	6,178,719	6,463,519	ns
Kernel Time (Cycles)	9,007,899	8,905,122	9,137,368	Cycle
Instr/wavefront	18	18	18	Instr/wavefro...
Wave Cycles	3,405	3,335	3,455	Cycles/wave
Dependency Wait Cycles	3,209	3,186	3,240	Cycles/wave
Issue Wait Cycles	165	112	193	Cycles/wave
Active Cycles	64	64	64	Cycles/wave
Wavefront Occupancy	3,198	3,166	3,210	Wavefronts

Speed-of-Light: Compute Pipeline

AMD
together we advance_

[AMD Official Use Only - General]

Omniperf profile – Roofline only

Profile with roofline:

```
$ omniperf profile -n roofline_case_app --roof-only -- <CMD> <ARGS>
```

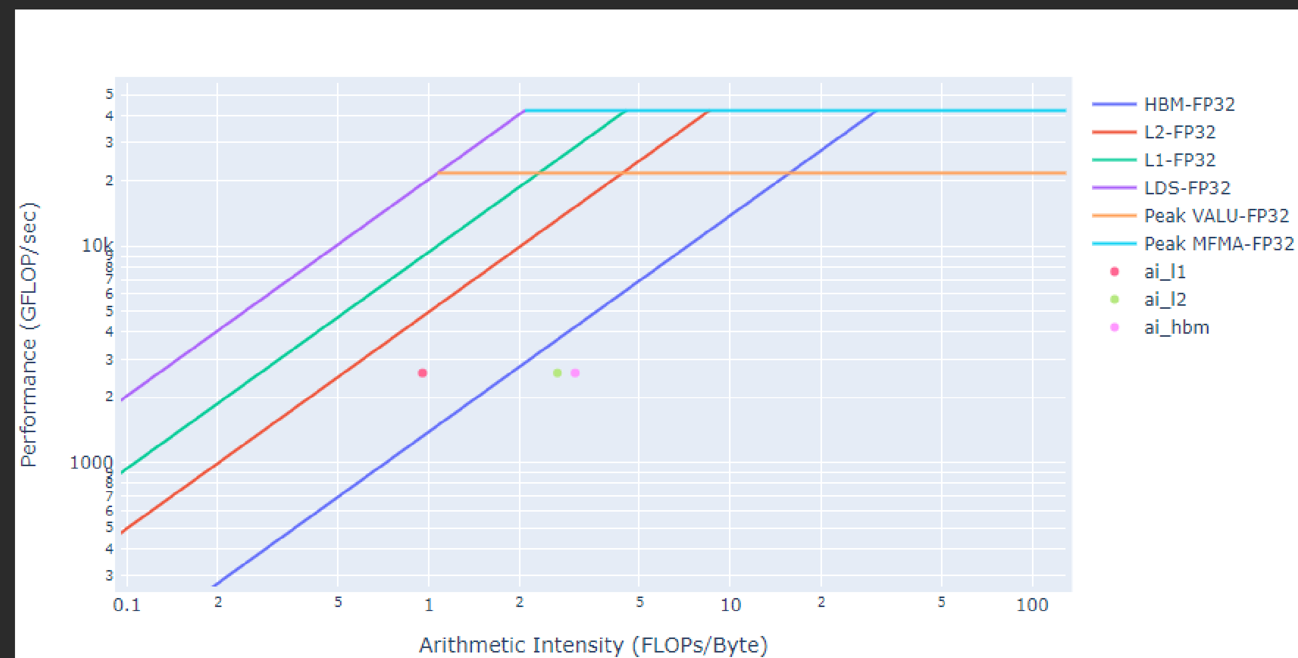
Analyze the profiled workload:

```
$ omniperf analyze -p path/to/workloads/roofline_case_app/mi200 --gui
```

Open web page http://IP:8050/

When profile with --roof-only, a PDF with the roofline will be created. In order to see the name of the kernels, add the --kernel-names and a second PDF will be created with names for the kernel markers:

```
$ omniperf profile -n roofline_case_app --roof-only --kernel-names -- <CMD> <ARGS>
```

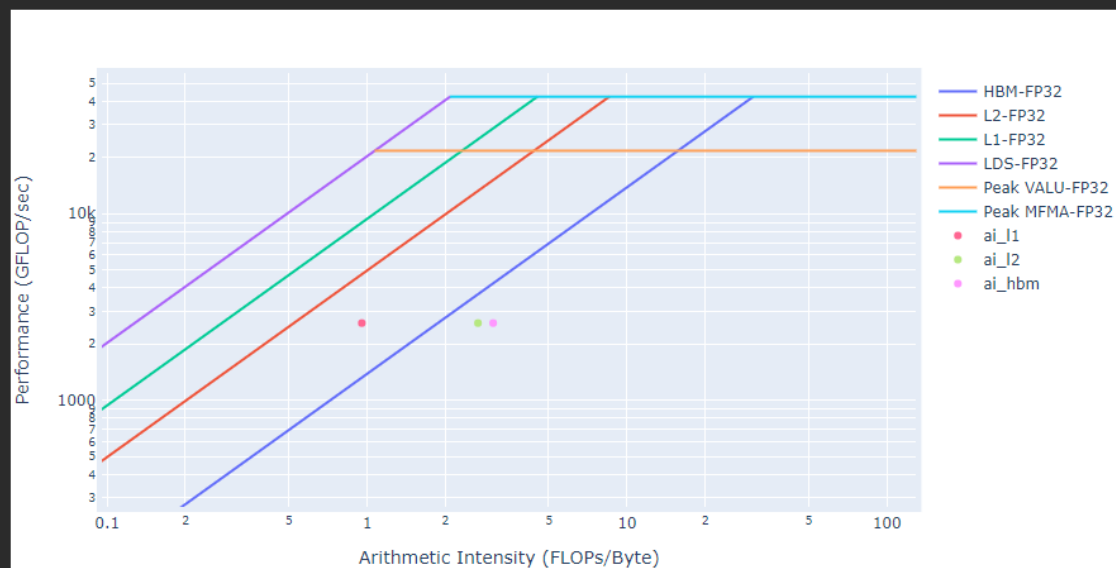


AMD
together we advance_

Roofline Analysis – Kokkos code

Menu NORMALIZATION: per Wave KERNELS: GCD: ALL DISPATCH FILTER: ALL Report Bug

```
void
Kokkos::Experimental::Impl::hip_parallel_launch_constant_memory<Kokkos::Impl::ParallelFor<idefix_for<Hydro::HlldMHD<2>
()::{lambda(int, int, int)#1}>(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, int const&,
int const&, int const&, int const&, int const&, int const&, Hydro::HlldMHD<2>()::{lambda(int, int, int)#1}}::{lambda(int
const&)#1}, Kokkos::RangePolicy<Kokkos::Experimental::HIP>, Kokkos::Experimental::HIP>, 256u, 1u>() [clone .kd]
```



- Roofline: the first-step characterization of workload performance
  - Workload characterization
    - Compute bound
    - Memory bound
    - Performance margin
    - L1/L2 cache accesses
- Thorough SoC perf analysis for each subsystem to identify bottlenecks
  - HBM
  - L1/L2
  - LDS
  - Shader compute
  - Wavefront dispatch
- Omniperf tooling support
  - Roofline plot (float, integer)
  - Baseline roofline comparison
  - Kernel statistics

AMD
together we advance_

[AMD Official Use Only - General]

SPI Resource Allocation

- Dispatch Bound
  - Wavefront dispatching failure due to resources limitation
    - Wavefront slots
    - VGPR
    - SGPR
    - LDS allocation
    - Barriers
    - Etc.
  - Omniperf tooling support
    - Shader Processor Input (SPI) metrics

6.2 SPI Resource Allocation

Metric	Avg	Min	Max	Unit
Wave request Failed (CS)	613303.00	613303.00	613303.00	Cycles
CS Stall	356961.00	356961.00	356961.00	Cycles
CS Stall Rate	62.95	62.95	62.95	Pct
Scratch Stall	0.00	0.00	0.00	Cycles
Insufficient SIMD Waveslots	0.00	0.00	0.00	Simd
Insufficient SIMD VGPRs	16252333.00	16252333.00	16252333.00	Simd
Insufficient SIMD SGPRs	0.00	0.00	0.00	Simd
Insufficient CU LDS	0.00	0.00	0.00	Cu
Insufficient CU Barries	0.00	0.00	0.00	Cu
Insufficient Bulky Resource	0.00	0.00	0.00	Cu
Reach CU Threadgroups Limit	0.00	0.00	0.00	Cycles
Reach CU Wave Limit	0.00	0.00	0.00	Cycles
VGPR Writes	4.00	4.00	4.00	Cycles/wave
SGPR Writes	5.00	5.00	5.00	Cycles/wave

AMD
together we advance_

What if Grafana and web GUI crashes when loading performance data?
(real case)

[AMD Official Use Only - General]

When profiling produces too large data…

- We had an application that the realistic case was dispatching 6.7 million calls to kernels
- Executing Omniperf without any options, it would take up to 36 hours to finish while single non instrumented execution takes less than 1 hour.
- HW counters add overhead
- We had totally around 9 GB of profiling data from 1 MPI process
- Uploading the data to a Grafana server was crashing Grafana server and we had to reboot the service
- Using standalone GUI was never finishing loading the data

- Omniperf profile has an option called –k where you define which specific kernel to profile. You can define the id 0-9 of the top 10 kernels.
- This creates profiling data **only** for the selected kernel
- This way you can split the profiling data to 10 executions, one per kernel:
  - You can use different resources to do the experiments in parallel (remember there can be performance variation between different GPUs)
  - You can visualize each kernel

Profile with roofline for a specific kernel:

```
 $ srun -N 1 -n 1 --ntasks-per-node=1 --gpus=1 --hint=nomultithread omniperf profile -n kernel_roof
-k kernel_name --roof-only  -- ./binary args
```


AMD
together we advance_

Example – DAXPY with a loop in the kernel

[AMD Official Use Only - General]

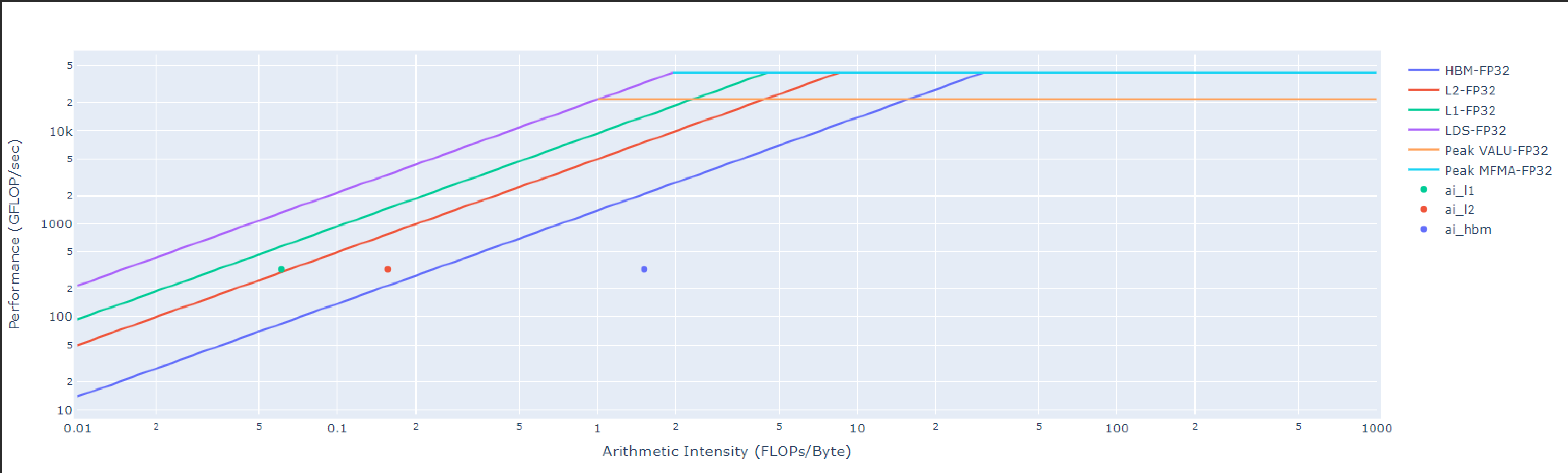
DAXPY – with a loop in the kernel

```
#include <hip/hip_runtime.h>

__constant__ double a = 1.0f;

__global__
void daxpy (int n, double const* x, int incx, double* y, int incy)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n)
       for(int ll=0;ll<20;ll++) {
        y[i] = a*x[i] + y[i];
       }
}

int main()
{
    int n = 1<<24;
    std::size_t size = sizeof(double)*n;

    double* d_x;
    double *d_y;
    hipMalloc(&d_x, size);
    hipMalloc(&d_y, size);

    int num_groups = (n+255)/256;
    int group_size = 256;
    daxpy<<<num_groups, group_size>>>(n, d_x, 1, d_y, 1);
    hipDeviceSynchronize();
```


AMD
together we advance_

[AMD Official Use Only - General]

Roofline



- Performance: almost 330 GFLOPs

AMD
together we advance_

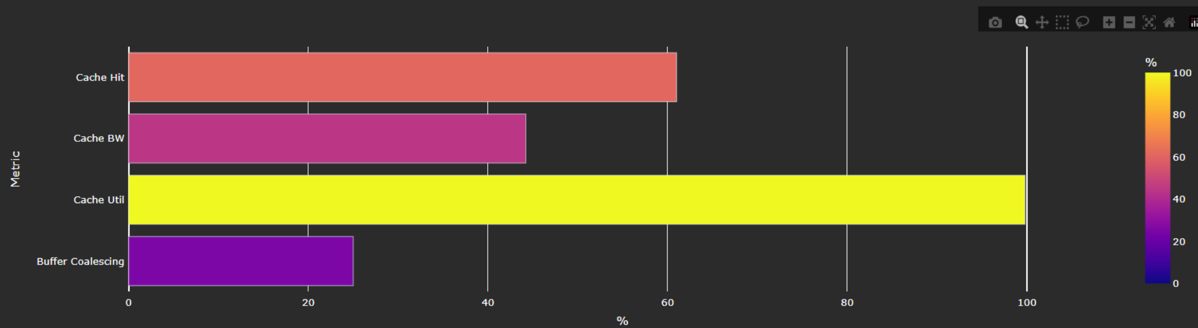
[AMD Official Use Only - General]

Kernel execution time and L1D Cache Accesses

KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
daxpy(int, double const*, int, double*, int) [clone .kd]	1.00	2024491.00	2024491.00	2024491.00	100.00

16. Vector L1 Data Cache

16.1 Speed-of-Light



16.2 L1D Cache Stalls

Metric	Mean	Min	Max	unit
Stalled on L2 Data	73.69	73.69	73.69	Pct
Stalled on L2 Req	19.47	19.47	19.47	Pct
Tag RAM Stall (Read)	0.00	0.00	0.00	Pct
Tag RAM Stall (Write)	0.00	0.00	0.00	Pct
Tag RAM Stall (Atomic)	0.00	0.00	0.00	Pct

16.3 L1D Cache Accesses

Metric	Avg	Min	Max	Unit
Total Req	2624.00	2624.00	2624.00	Req per wave
Read Req	1344.00	1344.00	1344.00	Req per wave
Write Req	1280.00	1280.00	1280.00	Req per wave
Atomic Req	0.00	0.00	0.00	Req per wave
Cache BW	5291.66	5291.66	5291.66	Gb/s
Cache Accesses	656.00	656.00	656.00	Req per wave
Cache Hits	400.16	400.16	400.16	Req per wave
Cache Hit Rate	61.00	61.00	61.00	Pct

AMD
together we advance_

[AMD Official Use Only - General]

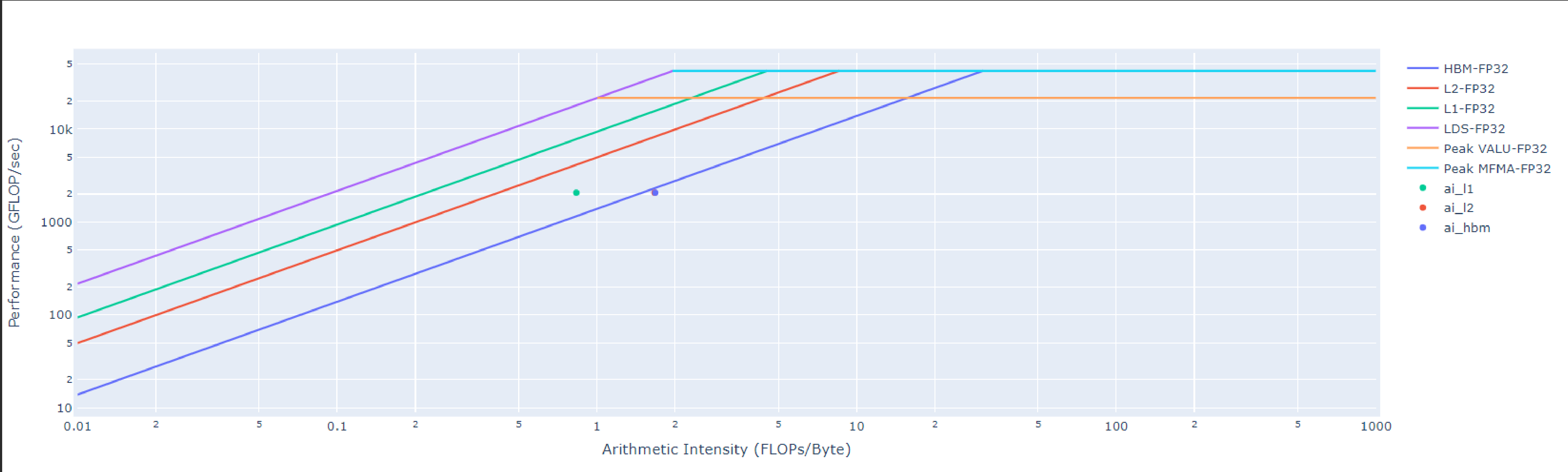
DAXPY – with a loop in the kernel - Optimized

```
#include <hip/hip_runtime.h>

__constant__ double a = 1.0f;

__global__
void daxpy (int n, double const* __restrict__ x, int incx, double* __restrict__ y, int incy)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n)
       for(int ll=0;ll<20;ll++) {
        y[i] = a*x[i] + y[i];
       }
}

int main()
{
    int n = 1<<24;
    std::size_t size = sizeof(double)*n;

    double* d_x;
    double *d_y;
    hipMalloc(&d_x, size);
    hipMalloc(&d_y, size);

    int num_groups = (n+255)/256;
    int group_size = 256;
    daxpy<<<num_groups, group_size>>>(n, d_x, 1, d_y, 1);
    hipDeviceSynchronize();
```


AMD
together we advance_

[AMD Official Use Only - General]

Roofline - Optimized



- Performance: almost 2 TFLOPs

AMD
together we advance_

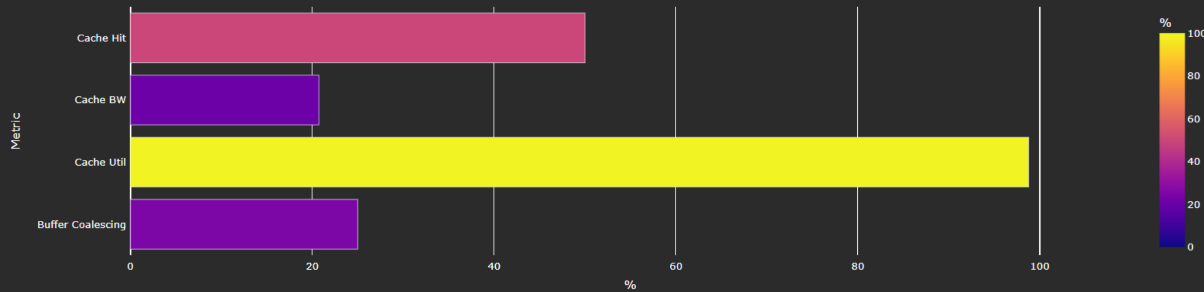
[AMD Official Use Only - General]

Kernel execution time and L1D Cache Accesses - Optimized

KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
daxpy(int, double const*, int, double*, int) [clone .kd]	1.00	323522.00	323522.00	323522.00	100.00

6.2 times faster!

16.1 Speed-of-Light



16.2 L1D Cache Stalls

Metric	Mean	Min	Max	unit
Stalled on L2 Data	79.08	79.08	79.08	Pct
Stalled on L2 Req	15.17	15.17	15.17	Pct
Tag RAM Stall (Read)	0.00	0.00	0.00	Pct
Tag RAM Stall (Write)	0.00	0.00	0.00	Pct
Tag RAM Stall (Atomic)	0.00	0.00	0.00	Pct

16.3 L1D Cache Accesses

Metric	Avg	Min	Max	Unit
Total Req	192.00	192.00	192.00	Req per wave
Read Req	128.00	128.00	128.00	Req per wave
Write Req	64.00	64.00	64.00	Req per wave
Atomic Req	0.00	0.00	0.00	Req per wave
Cache BW	2480.60	2480.60	2480.60	Gb/s
Cache Accesses	48.00	48.00	48.00	Req per wave
Cache Hits	24.00	24.00	24.00	Req per wave
Cache Hit Rate	50.00	50.00	50.00	Pct
Invalidate	0.00	0.00	0.00	Req per wave

AMD
together we advance_

Summary

- Omniperf is a tool that collects many counters automatically
- It can create roofline analysis to understand how efficient are your kernels
- It displays a lot of metrics regarding your kernels, however, it is required to know more about your kernel
- It does not have learning curve to start running it, but requies knowledge for the analysis
- It supports Grafana, standalone GUI, and CLI
- Includes several features such as:
  - System Speed-of-Light Panel
  - Memory Chart Analysis Panel
  - Vector L1D Cache Panel
  - Shader Processing Input (SPI) Panel

AMD
together we advance_

Questions?

DISCLAIMERS AND ATTRIBUTIONS

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

© 2023 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, Radeon™, Instinct™, EPYC, Infinity Fabric, ROCm™, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

AMD
together we advance_

AMD