

Introduction to ROC-Profiler (rocprof)

Presenter: Sam Antao

**LUMI Advanced Training
Mar 6th, 2025**

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	__hipPopCallConfigur	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

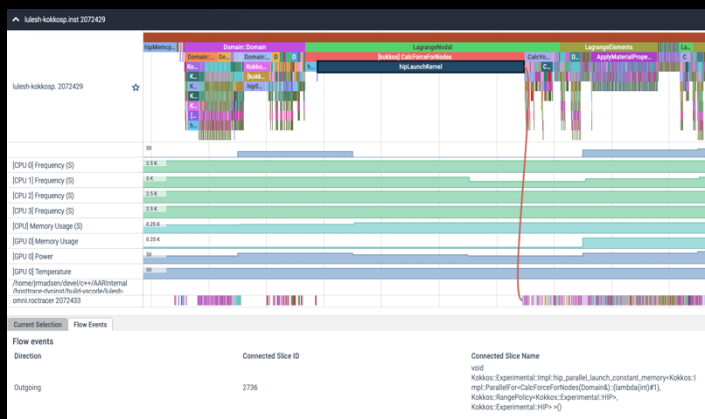
- Comprehensive trace collection
- CPU
- GPU

Supports

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

Visualisation

Traces visualized with Perfetto



Omniperf

Performance Analysis

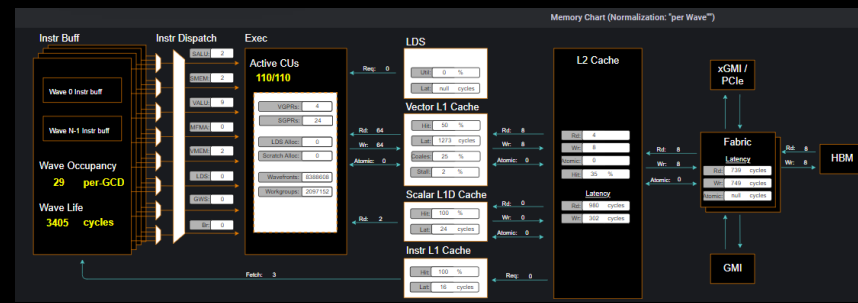
- Automated collection of hardware counters
- Analysis
- Visualisation

Supports

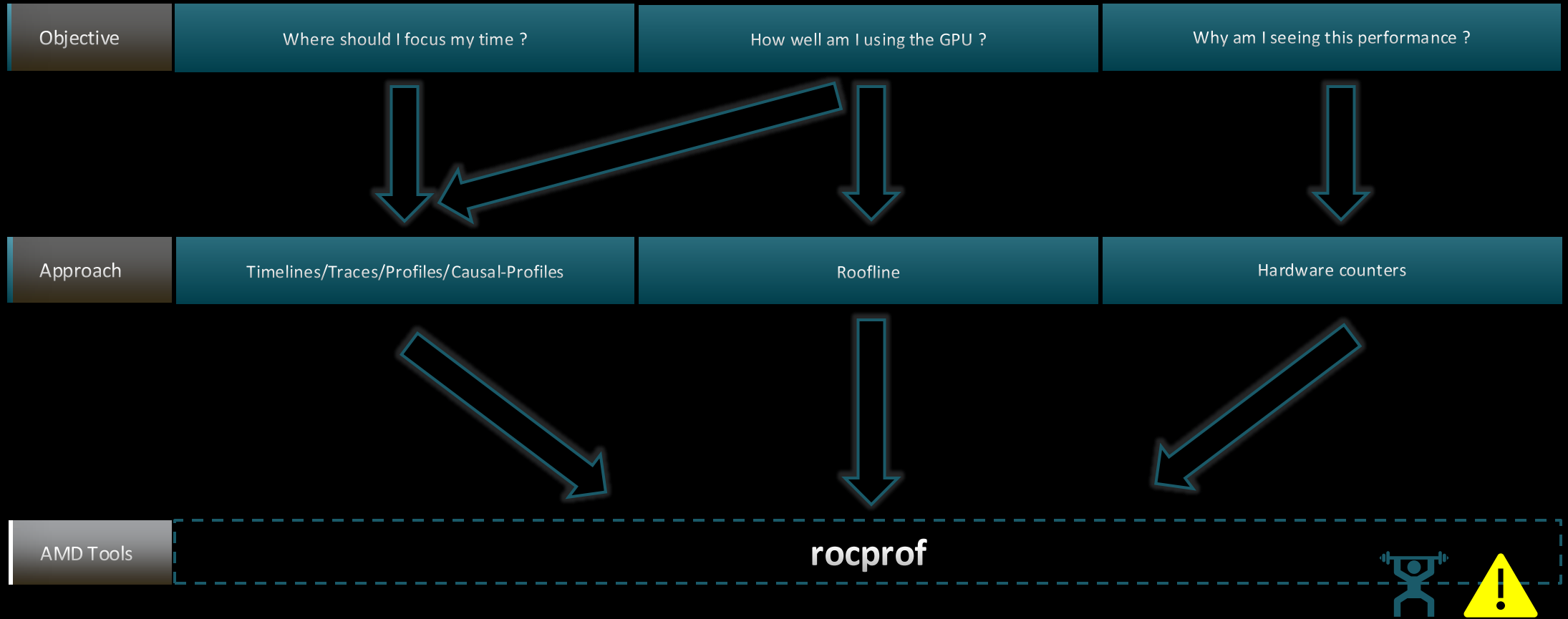
- Speed of Light
- Memory chart
- Rooflines
- Kernel comparison

Visualisation

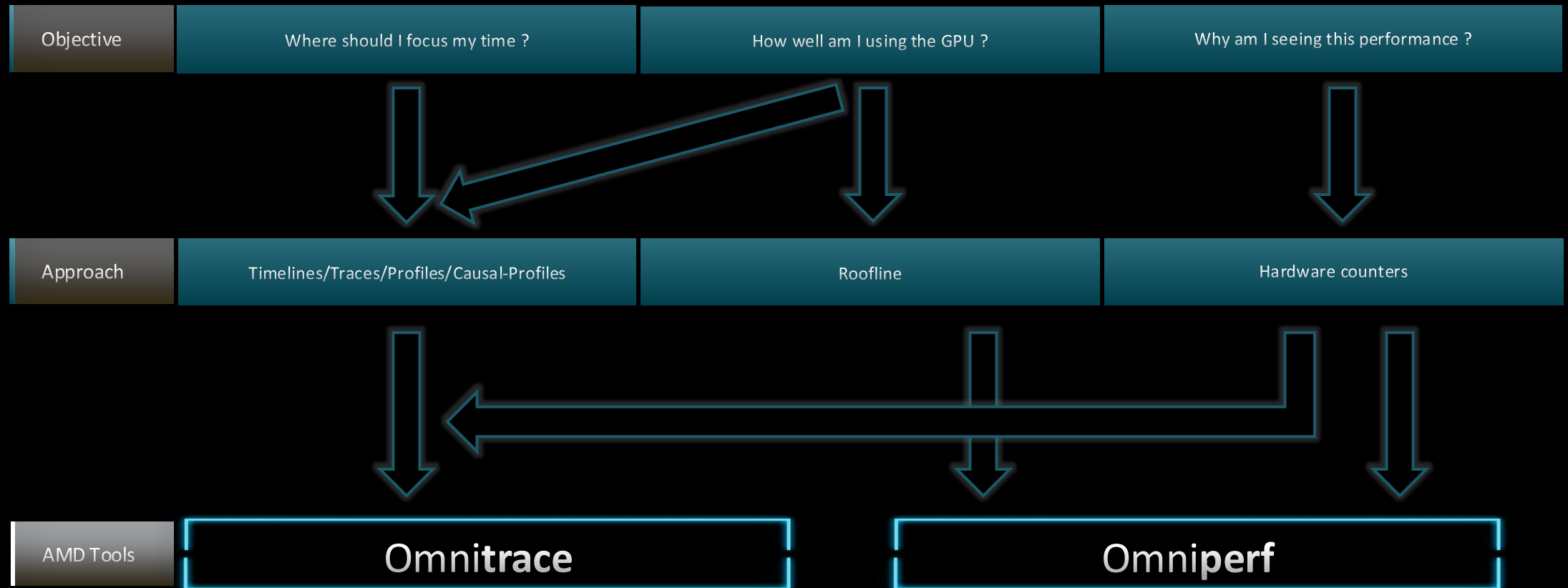
With Grafana or standalone GUI



Background – AMD Profilers



Background – AMD Profilers



ROCm on LUMI



Meant to support older version of apps and frameworks

Facilitate transition
GPU address sanitizer (beta)

Data pre-processing capabilities (MIVisionX)

~~GPU-Aware MPI~~

Latest Pytorch and other AI frameworks require this version

Introduced many performance improvements

ROCPROF V3

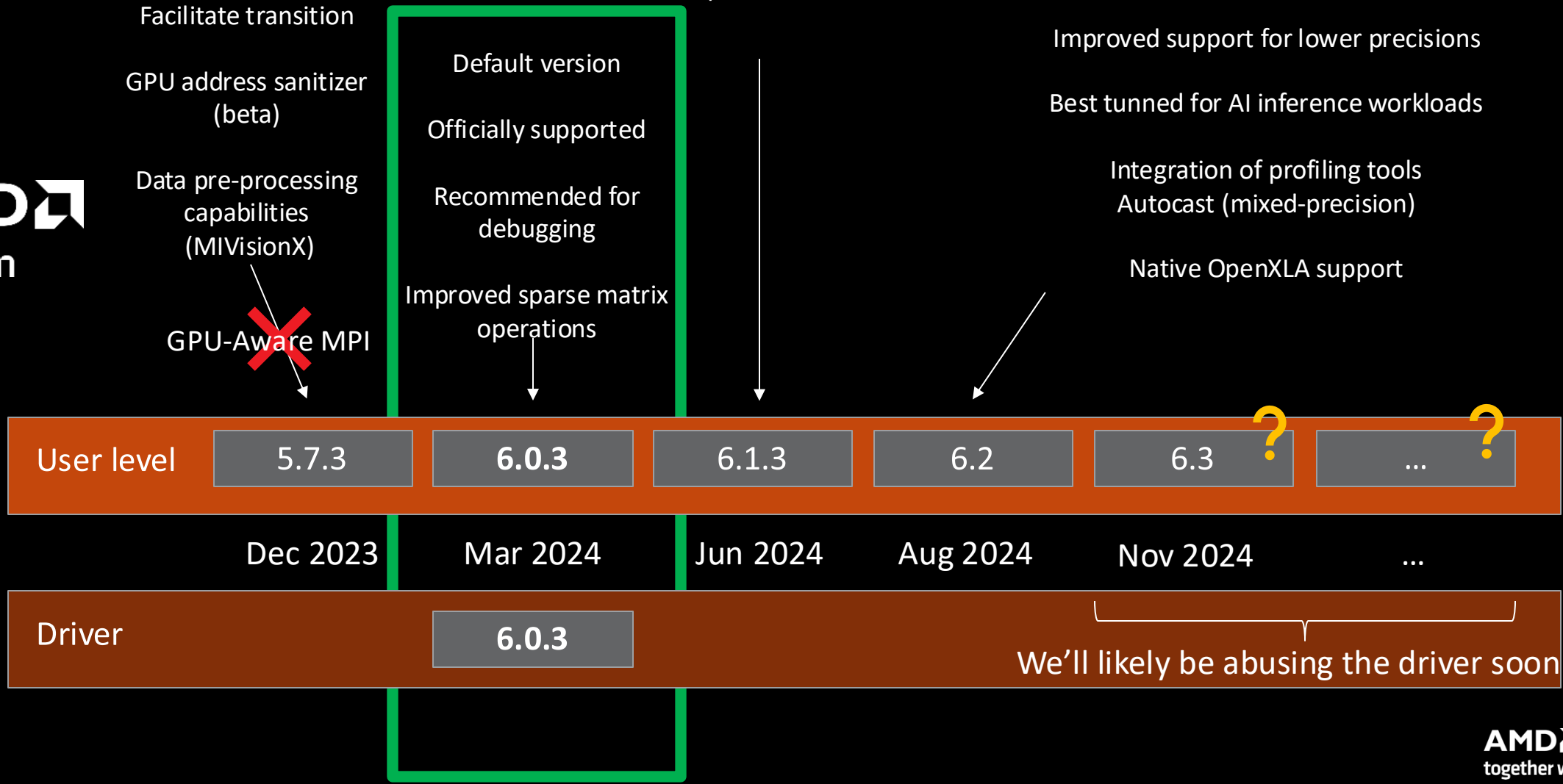
Many stability and performance improvements for performance libraries

Improved support for lower precisions

Best tuned for AI inference workloads

Integration of profiling tools
Autocast (mixed-precision)

Native OpenXLA support



What is ROC-Profiler (v1-v2-v3)?

- ROC-profiler (also referred to as `rocprof`) is the command line front-end for AMD's GPU profiling libraries
 - Repo: <https://github.com/ROCm-Developer-Tools/rocprofiler>
- rocprof contains the central components allowing application traces and counter collection
 - Under constant development
- Distributed with ROCm
- The output of rocprofv1 can be visualized in the Chrome browser with Perfetto (<https://ui.perfetto.dev/>)
- There are ROCProfiler V1 and V2 (roctracer and rocprofiler into single library, same API)
- ROC-profiler-SDK is a profiling and tracing library for HIP and ROCm application. The new API improved thread safety and includes more efficient implementations and provides a tool library to support on writing your tool implementations. It is still in beta release.
- `rocprofv3` uses this tool library to profile and trace applications.

rocprof (v1): Getting Started + Useful Flags

- To get help:
`${ROCM_PATH}/bin/rocprof -h`
- Useful housekeeping flags:
 - `--timestamp <on|off>` - turn on/off gpu kernel timestamps
 - `--basenames <on|off>` - turn on/off truncating gpu kernel names (i.e., removing template parameters and argument types)
 - `-o <output csv file>` - Direct counter information to a particular file name
 - `-d <data directory>` - Send profiling data to a particular directory
 - `-t <temporary directory>` - Change the directory where data files typically created in /tmp are placed. This allows you to save these temporary files.
- Flags directing rocprofiler activity:
 - `-i input<.txt|.xml>` - specify an input file (note the output files will now be named input.*)
 - `--hsa-trace` - to trace GPU Kernels, host HSA events (more later) and HIP memory copies.
 - `--hip-trace` - to trace HIP API calls
 - `--roctx-trace` - to trace roctx markers
 - `--kfd-trace` - to trace GPU driver calls
- Advanced usage
 - `-m <metric file>` - Allows the user to define and collect custom metrics. See [rocprofiler/test/tool/*.xml](#) on GitHub for examples.

rocprof (v1): : Kernel Information

- rocprof can collect kernel(s) execution stats
 - `$ /opt/rocm/bin/rocprof --stats --basenames on <app with arguments>`
- This will output two csv files:
 - `results.csv`: information per each call of the kernel
 - `results.stats.csv`: statistics grouped by each kernel
- Content of `results.stats.csv` to see the list of GPU kernels with their durations and percentage of total GPU time:

```
"Name","Calls","TotalDurationNs","AverageNs","Percentage"
"JacobiIterationKernel",1000,556699359,556699,43.291753895270446
"NormKernel1",1001,430797387,430367,33.500980655394606
"LocalLaplacianKernel",1000,280014065,280014,21.775307970480817
"HaloLaplacianKernel",1000,14635177,14635,1.1381052818810995
"NormKernel2",1001,3770718,3766,0.2932300765671734
"__amd_rocclr_fillBufferAligned.kd",1,8000,8000,0.0006221204058583505
```

- In a spreadsheet viewer, it is easier to read:

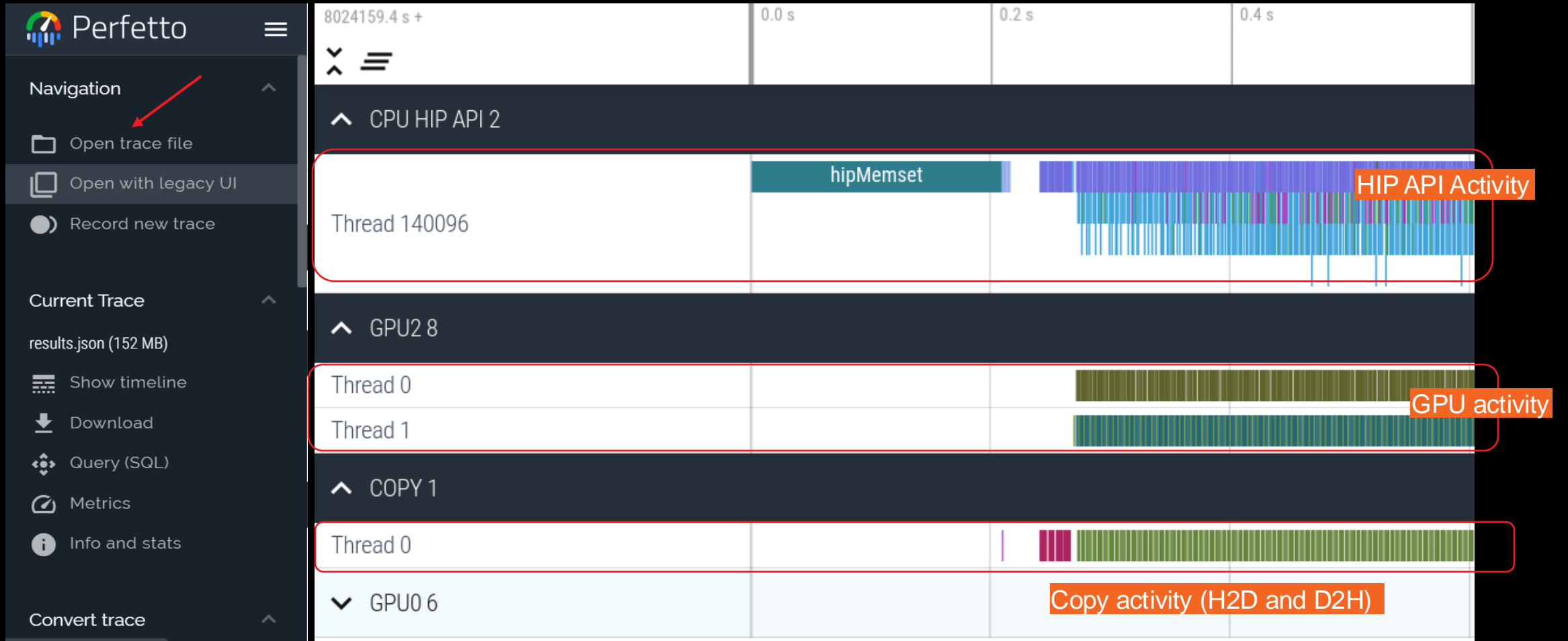
	A	B	C	D	E
1	Name	Calls	TotalDurationNs	AverageNs	Percentage
2	JacobiIterationKernel	1000	556699359	556699	43.2917538952704
3	NormKernel1	1001	430797387	430367	33.5009806553946
4	LocalLaplacianKernel	1000	280014065	280014	21.7753079704808
5	HaloLaplacianKernel	1000	14635177	14635	1.1381052818811
6	NormKernel2	1001	3770718	3766	0.293230076567173
7	__amd_rocclr_fillBufferAligned	1	8000	8000	0.000622120405858

rocprof (v1): + Perfetto: Collecting and Visualizing App Traces

- rocprof can collect traces

```
$ /opt/rocm/bin/rocprof --hip-trace <app with arguments>
```

This will output a .json file that can be visualized using the Chrome browser and Perfetto (<https://ui.perfetto.dev/>)



rocprofv3: Getting Started + Useful Flags

- To get help:

```
${ROCM_PATH}/bin/rocprofv3 -h
```

- Useful housekeeping flags:

- `--hip-trace` For Collecting HIP Traces (runtime + compiler)
- `--hip-runtime-trace` For Collecting HIP Runtime API Traces
- `--hip-compiler-trace` For Collecting HIP Compiler generated code Traces
- `--marker-trace` For Collecting Marker (ROCTX) Traces
- `--memory-copy-trace` For Collecting Memory Copy Traces
- `--stats` For Collecting statistics of enabled tracing types
- `--hsa-trace` For Collecting HSA Traces (core + amd + image + finalizer)
- `--hsa-core-trace` For Collecting HSA API Traces (core API)
- `--hsa-amd-trace` For Collecting HSA API Traces (AMD-extension API)
- `--hsa-image-trace` For Collecting HSA API Traces (Image-extension API)
- `--hsa-finalizer-trace` For Collecting HSA API Traces (Finalizer-extension API)

rocprofv3: Getting Started + Useful Flags (II)

- Useful housekeeping flags:
 - `-s, --sys-trace` For Collecting HIP, HSA, Marker (ROCTx), Memory copy, Scratch memory, and Kernel dispatch traces
 - `-M, --mangled-kernels` Do not demangle the kernel names
 - `-T, --truncate-kernels` Truncate the demangled kernel names
 - `-L, --list-metrics` List metrics for counter collection
 - `-i INPUT, --input INPUT` Input file for counter collection
 - `-o OUTPUT_FILE, --output-file OUTPUT_FILE`
For the output file name
 - `-d OUTPUT_DIRECTORY, --output-directory OUTPUT_DIRECTORY`
For adding output path where the output files will be saved
 - `--output-format {csv,json,pftrace} [{csv,json,pftrace} ...]`
For adding output format (supported formats: csv, json, pftrace)
 - `--log-level {fatal,error,warning,info,trace}`
Set the log level
 - `--kernel-names KERNEL_NAMES [KERNEL_NAMES ...]`
Filter kernel names
 - `--preload [PRELOAD ...]`
Libraries to prepend to LD_PRELOAD (usually for sanitizers)
- rocprofv3 requires double-hyphen (`--`) before the application to be executed, e.g.


```
$ rocprofv3 [<rocprofv3-option> ...] -- <application> [<application-arg> ...]
$ rocprofv3 --hip-trace -- ./myapp -n 1
```
- Instructions: <https://rocm.docs.amd.com/projects/rocprofiler-sdk/en/docs-6.2.1/how-to/using-rocprofv3.html>

rocpv3: Kernel Information

- rocpv3 can collect kernel(s) execution stats

```
$ /opt/rocm/bin/rocpv3 --stats --kernel-trace -T -- <app with arguments>
```

- This will output four csv files (XXXXX are numbers):

- XXXXX_agent_info.csv: information for the used hardware APU/GPU and CPU
- XXXXX_kernel_traces.csv: information per each call of the kernel
- XXXXX_kernel_stats.csv: statistics grouped by each kernel
- XXXXX_domain_stats.csv: statistics grouped by domain, such as KERNEL_DISPATCH, HIP_COMPILER_API

- Content of results.stats.csv to see the list of GPU kernels with their durations and percentage of total GPU time:

```
"Name", "Calls", "TotalDurationNs", "AverageNs", "Percentage", "MinNs", "MaxNs", "StdDev"
"NormKernel1", 1001, 365858158, 365492.665335, 53.49, 360561, 449240, 3460.551681
"JacobiIterationKernel", 1000, 171479968, 171479.968000, 25.07, 162040, 205241, 10113.842491
"LocalLaplacianKernel", 1000, 135771713, 135771.713000, 19.85, 130400, 145121, 3349.580100
"HaloLaplacianKernel", 1000, 7777189, 7777.189000, 1.14, 7000, 12120, 349.399610
"NormKernel2", 1001, 3107927, 3104.822178, 0.4544, 2200, 138681, 6466.048652
"__amd_rocclr_fillBufferAligned", 1, 2720, 2720.000000, 3.977e-04, 2720, 2720, 0.00000000e+00
```

- In a spreadsheet viewer, it is easier to read:

	A	B	C	D	E	F	G	H
1	Name	Calls	TotalDurationNs	AverageNs	Percentage	MinNs	MaxNs	StdDev
2	NormKernel1	1001	365858158	365492.665	53.49	360561	449240	3460.552
3	JacobiIterationKernel	1000	171479968	171479.968	25.07	162040	205241	10113.84
4	LocalLaplacianKernel	1000	135771713	135771.713	19.85	130400	145121	3349.58
5	HaloLaplacianKernel	1000	7777189	7777.189	1.14	7000	12120	349.3996
6	NormKernel2	1001	3107927	3104.82218	0.4544	2200	138681	6466.049
7	__amd_rocclr_fillBufferAligned	1	2720	2720	3.98E-04	2720	2720	0

rocprofv3: Collecting Application Traces

- rocprof can collect a variety of trace event types, and generate timelines in JSON format for use with Perfetto, currently, however better use the pfttrace output format (`--output-format pfttrace`):

Trace Event	rocprof Trace Mode
HIP API call	<code>--hip-trace</code>
GPU Kernels	<code>--kernel-trace</code>
Host <-> Device Memory copies	<code>--hip-trace</code> or <code>--memory-copy-trace</code>
CPU HSA Calls	<code>--hsa-trace</code>
User code markers	<code>--marker-trace</code>
Collect HIP, HSA, Kernels, Memory Copy, Marker API	<code>--sys-trace</code>
Scratch memory operations	<code>--scratch-memory-trace</code>

- You can combine modes like `--stats --hip-trace --hsa-trace --output-format pfttrace`

rocprof + Perfetto: Collecting and Visualizing Application Traces

- rocprof can collect traces

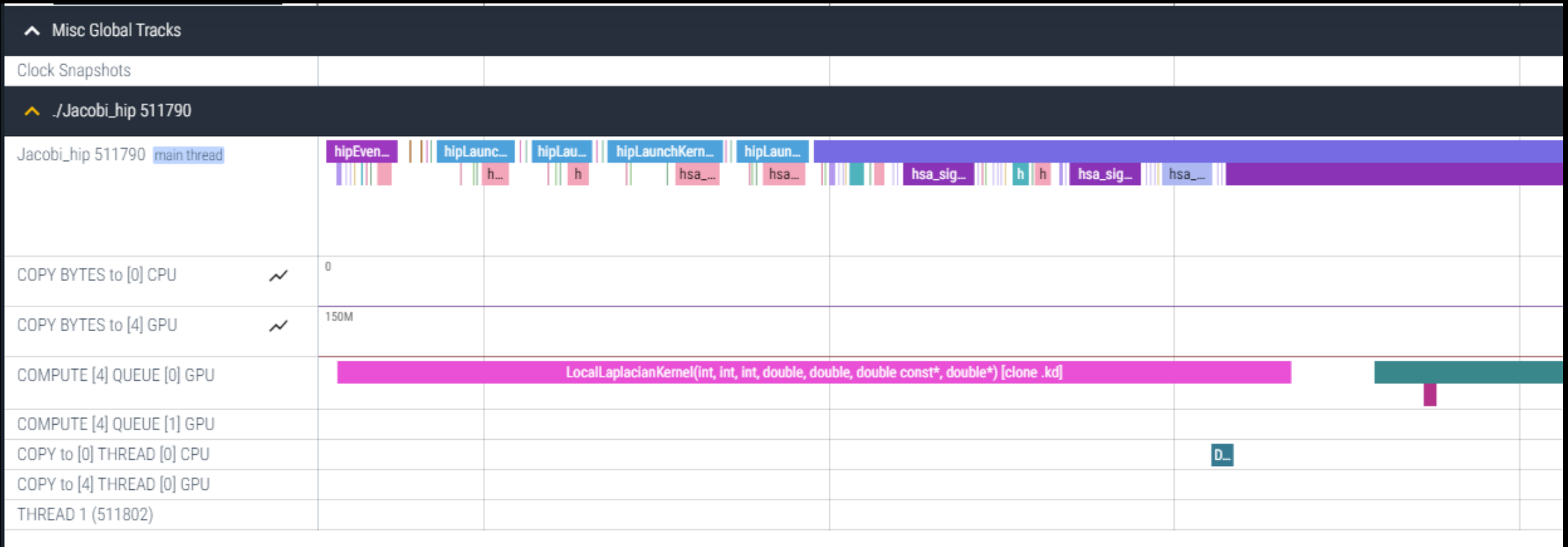
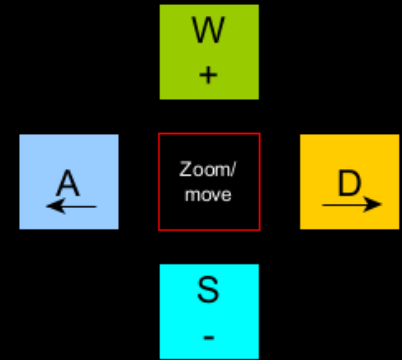
```
$ /opt/rocm/bin/rocprof --hip-trace --output-format pfttrace -- <app with arguments>
```

This will output a pfttrace file that can be visualized using the chrome browser and Perfetto (<https://ui.perfetto.dev/>)

The screenshot displays the Perfetto web interface for a trace titled `./Jacobi_hip 511790`. The interface is divided into a left sidebar and a main visualization area. The sidebar contains navigation options like 'Open trace file', 'Open with legacy UI', and 'Record new trace', along with 'Current Trace' information (results.json, 152 MB) and actions like 'Show timeline', 'Download', 'Query (SQL)', 'Metrics', and 'Info and stats'. The main area shows a detailed trace for the 'main thread' of 'Jacobi_hip 511790'. The trace is visualized as a series of horizontal bars representing different activity types over time. The top bar is labeled 'hipMe...' and is highlighted with a red box and an orange callout 'HIP API Activity'. Below it are bars for 'COPY BYTES to [0] CPU' (0 bytes) and 'COPY BYTES to [4] GPU' (150M bytes). The next two bars are 'COMPUTE [4] QUEUE [0] GPU' and 'COMPUTE [4] QUEUE [1] GPU', both highlighted with a red box and an orange callout 'GPU activity'. Below these are 'COPY to [0] THREAD [0] CPU' and 'COPY to [4] THREAD [0] GPU', also highlighted with a red box and an orange callout 'Copy activity (H2D and D2H)'. The bottom bar is 'THREAD 1 (511802)'. A red arrow in the sidebar points to the 'Open trace file' option.

Perfetto: Visualizing Application Traces

- Zoom in to see individual events
- Navigate trace using WASD keys



Perfetto: Kernel Information and Flow Events

- Zoom and select a kernel, you can see the link to the HIP call launching the kernel
- Try to open the information for the kernel (button at bottom right)

The screenshot displays the Perfetto Profiler interface. At the top, there is a 'Misc Global Tracks' section. Below it, the 'Clock Snapshots' track is visible. The main track is titled './Jacobi_hip 511790' and shows a 'main thread' with several events: 'hipLaunchKernel', 'hipEven...', 'hipLa...', 'hipLau...', 'hipLa...', 'hipLa...', and 'hipMemcpy'. A red arrow points from the 'hipLaunchKernel' event to a selected kernel track labeled 'LocalLaplacianKernel(int, int, int, double, double, double const*, double*) [clone .kd]'. Below the main track, there are tracks for 'COPY BYTES to [0] CPU' (0 bytes), 'COPY BYTES to [4] GPU' (150M bytes), and 'COMPUTE [4] QUEUE [0] GPU'. At the bottom right, there is a 'HaloL...' track. A red circle highlights an upward-pointing arrow icon in the bottom right corner of the interface.

Perfetto: Kernel Information

The screenshot displays the Perfetto kernel information interface. On the left, a metadata table provides details about the kernel slice. On the right, a table shows the slice's execution context, and a list of arguments is provided below.

Name	LocalLaplacianKernel(int, int, int, double, double, double const*, double*) [clone .kd]
Category	kernel_dispatch
Start time	00:00:00.969713738
Absolute Time	2024-10-01T10:53:58.837832382
Duration	138us 520ns
Process	./Jacobi_hip [511790]
SQL ID	slice[4481]

Kernel name and args (points to the Name field)

Slice	Delay	Thread
hsa_signal_store_screlease	4us 110ns	Jacobi_hip 511790 (./Jacobi_hip 511790)

Duration (points to the Duration field)

Arguments

- debug
 - begin_ns - 4556433481727591
 - end_ns - 4556433481866111
 - delta_ns - 138520
 - kind - 11
 - agent - 4
 - corr_id - 4364
 - queue - 4
 - tid - 511790
 - kernel_id - 13
 - private_segment_size - 0
 - group_segment_size - 0
 - workgroup_size - 256
 - grid_size - 16777216
 - legacy_event.passthrough_utid - 1

Workgroup size and grid size (points to workgroup_size and grid_size)

Rocprofv3: OpenMP Offloading

- The option `--kernel-trace` provides information of the OpenMP kernels, good to use `--hsa-trace` if you want information from HSA layer
- For example:

```
srun -n 1 rocprofv3 --stats --kernel-trace --output-format pfttrace -- <app with arguments>
```

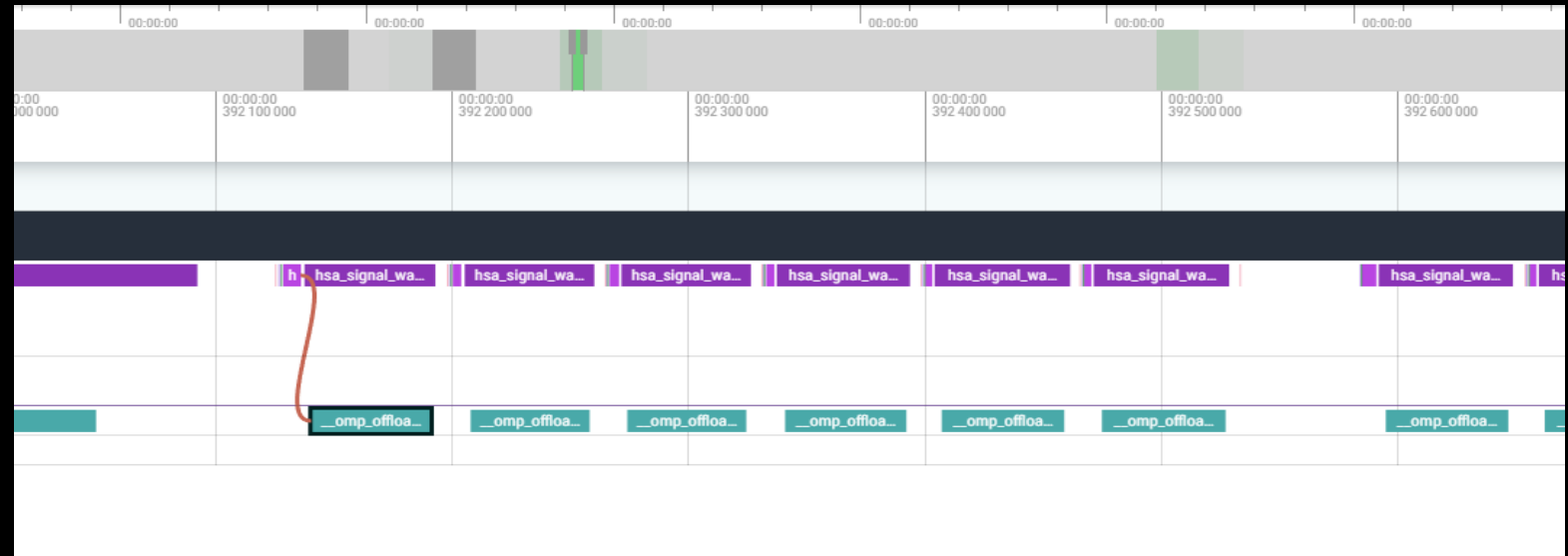
Content of XXXXX_kernel_stats.csv:

```
"Name","Calls","TotalDurationNs","AverageNs","Percentage","MinNs","MaxNs","StdDev"
"__omp_offloading_32_7f7a__Z6evolveR5FieldS0_dd_l24",500,45818062,91636.124000,100.00,49840,19483408,868965.767084
```

Content of XXXXX_kernel_trace.csv

```
"Kind","Agent_Id","Queue_Id","Kernel_Id","Kernel_Name","Correlation_Id","Start_Timestamp","End_Timestamp","Private_Segment_Size","Group_Segment_Size","
Workgroup_Size_X","Workgroup_Size_Y","Workgroup_Size_Z","Grid_Size_X","Grid_Size_Y","Grid_Size_Z"
"KERNEL_DISPATCH",4,1,1,"__omp_offloading_32_7f7a__Z6evolveR5FieldS0_dd_l24",1,4547852833814530,4547852853297938,0,0,256,1,1,233472,1,1
"KERNEL_DISPATCH",4,1,1,"__omp_offloading_32_7f7a__Z6evolveR5FieldS0_dd_l24",2,4547852853393869,4547852853446789,0,0,256,1,1,233472,1,1
"KERNEL_DISPATCH",4,1,1,"__omp_offloading_32_7f7a__Z6evolveR5FieldS0_dd_l24",3,4547852853461519,4547852853514599,0,0,256,1,1,233472,1,1
..."
```

Perfetto and OpenMP visualization



- Using: `--sys-trace --output-format pfttrace`
- We can use: `--kernel-trace --output-format pfttrace`

<code>end_ns -</code>	4552720951004323
<code>delta_ns -</code>	50880
<code>kind -</code>	11
<code>agent -</code>	4
<code>corr_id -</code>	631
<code>queue -</code>	1
<code>tid -</code>	503089
<code>kernel_id -</code>	1
<code>private_segment_size -</code>	0
<code>group_segment_size -</code>	0
<code>workgroup_size -</code>	256
<code>grid_size -</code>	233472

rocpv3: Collecting Application Traces with rocTX Markers and Regions

- rocprofv3 can collect user defined regions or markers using rocTX

- Annotate code with roctx regions:
`#include <rocprofiler-sdk-roctx/roctx.h>`

```
...
    roctxRangePush("reduce_for_c");
    reduce_function ();
    roctxRangePop();
...
```

- Annotate code with roctx markers:

```
...
    roctxMark("start of some code");
    // some_code
    roctxMark("end of some code");
...
```

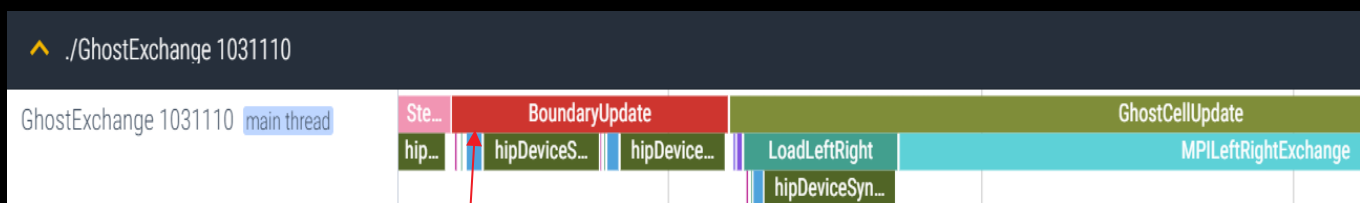
- Add roctx and roctracer libraries to link line:

```
-L${ROCM_PATH}/lib -lrocprofiler-sdk-roctx -lroctracer64
```

- Profile with `--roctx-range` option:

```
$ /opt/rocm/bin/rocprofv3 --hip-trace --marker-trace -- <app with arguments>
```

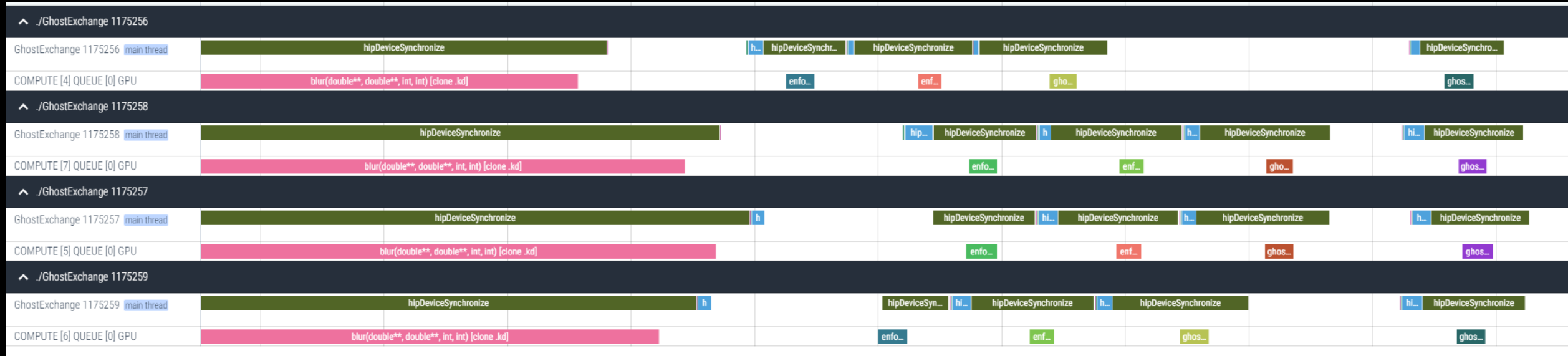
- Important: There is some difference regarding roctx between rocprof and rocprofv3



Roctx Range

Rocprofv3: Merge traces

- When you have one pfttrace per MPI processes you can merge them as follows:
 - For example `cat XXXXX_results.pfttrace > all_ghostexchange.pfttrace`
 - Then visualize the file called `all_ghostexchange.pfttrace`



rocprofv3: Commonly Used GPU Counters

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
MemUnitStalled	The percentage of GPUTime the memory unit is stalled
CU_OCCUPANCY	The ratio of active waves on a CU to the maximum number of active waves supported by the CU
MeanOccupancyPerCU	Mean occupancy per compute unit
MeanOccupancyPerActiveCU	Mean occupancy per active compute unit

rocpv3: Collecting Hardware Counters

- rocprofv3 can collect a number of hardware counters and derived counters
 - `$ /opt/rocm/bin/rocprofv3 -L`
- Specify counters in a counter file. For example:
 - `$ /opt/rocm/bin/rocprofv3 -i rocprof_counters.txt -- <app with args>`
 - `$ cat rocprof_counters.txt`
pmc: VALUUtilization VALUBusy FetchSize WriteSize MemUnitStalled
pmc: GPU_UTIL CU_OCCUPANCY MeanOccupancyPerCU MeanOccupancyPerActiveCU
- A limited number of counters can be collected during a specific pass of code
 - Each line in the counter file will be collected in one pass
 - You will receive an error suggesting alternative counter ordering if you have too many / conflicting counters on one line
- One directory per pmc line will be created, for example pmc_1 and pmc_2 for the two lines in the file with the counters.
- One agent_info and one counter_collection csv file per MPI process will be created containing all the requested counters for each invocation of every kernel

rocprof: Profiling Overhead

- As with every profiling tool, there is an overhead
- The percentage of the overhead depends on the profiling options used
 - For example, tracing is faster than hardware counter collection
- When collecting many counters, the collection may require multiple passes
- With rocTX markers/regions, tracing can take longer and the output may be large
 - Sometimes too large to visualize
- The more data collected, the more the overhead of profiling
 - Depends on the application and options used
- rocprofv3 has less overhead than rocprof (v1) on various examples with extensive ROCm calls

Summary

- rocprofv3 is the open source, command line AMD GPU profiling tool distributed with ROCm 6.2 and later
- rocprofv3 provides tracing of GPU kernels, through various options, HIP API, HSA API, Copy activity and others
- rocprofv3 can be used to collect GPU hardware counters with additional overhead
- Perfetto seems to visualize pftrace files without significant issues
- Other output files are in text/CSV format

Hands-on exercises

<https://hackmd.io/@sfantao/lumi-training-sto-2025#Rocprof>

We welcome you to explore our HPC Training Examples repo:

<https://github.com/amd/HPCTrainingExamples>

A table of contents for the READMEs if available at the top-level README in the repo

Relevant exercises for this presentation located in **Rocprof** directory.

Link to instructions on how to run the tests: **Rocprof/README.md** and subdirectories

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

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.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED "AS IS" WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

© 2025 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ROCm, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

AMD 