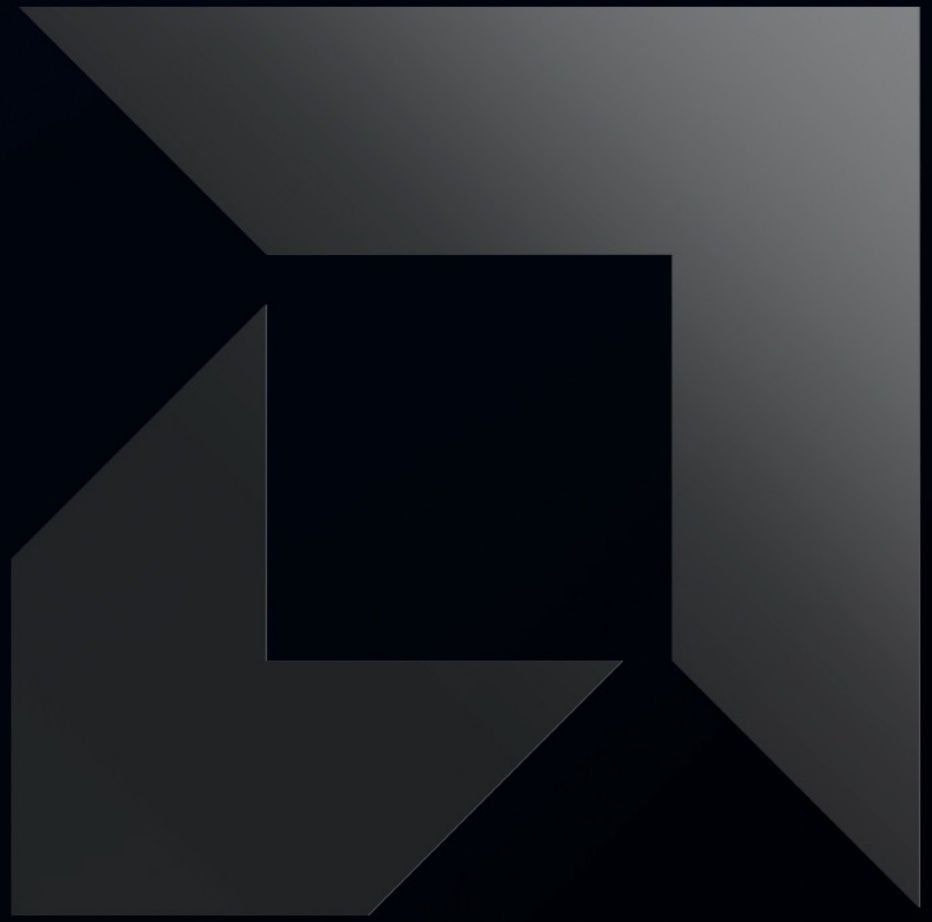


System Profiling with rocprof-sys

Presenter: Sam Antao

LUMI Performance Workshop

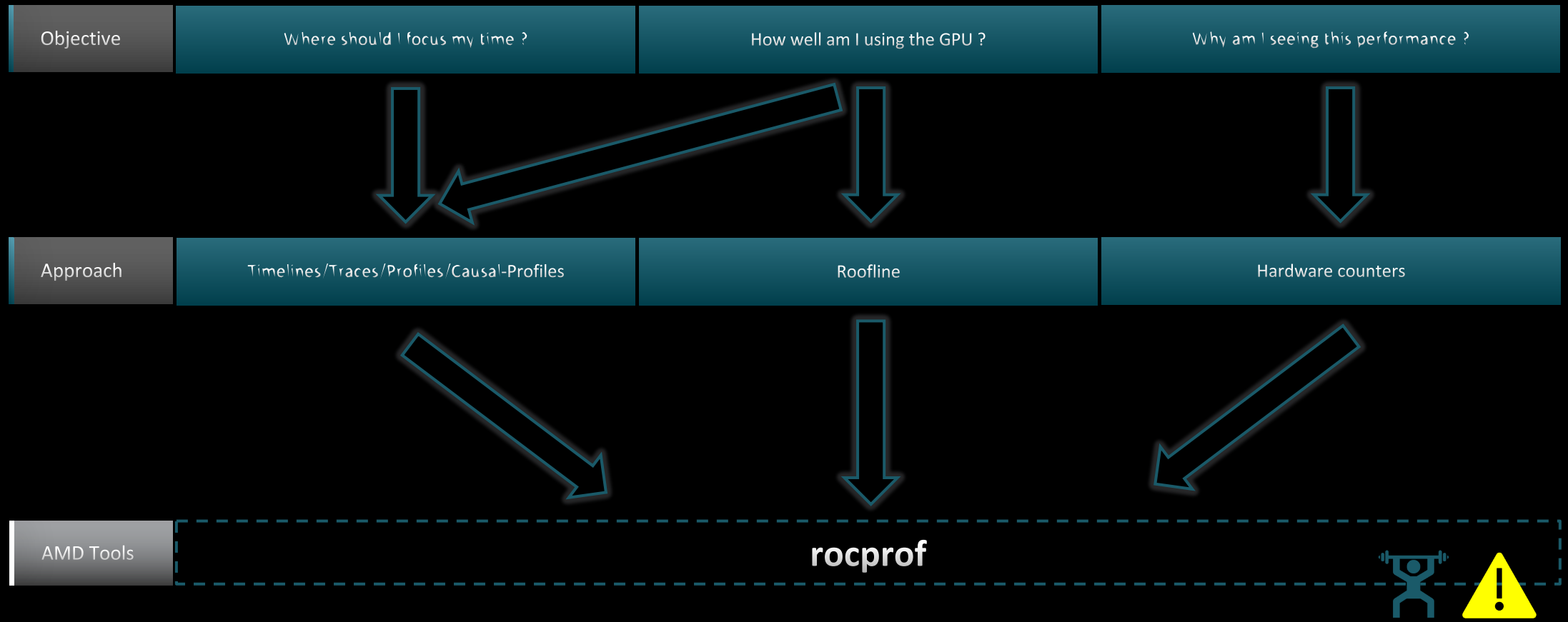
Nov 12th, 2026

AMD

together we advance_

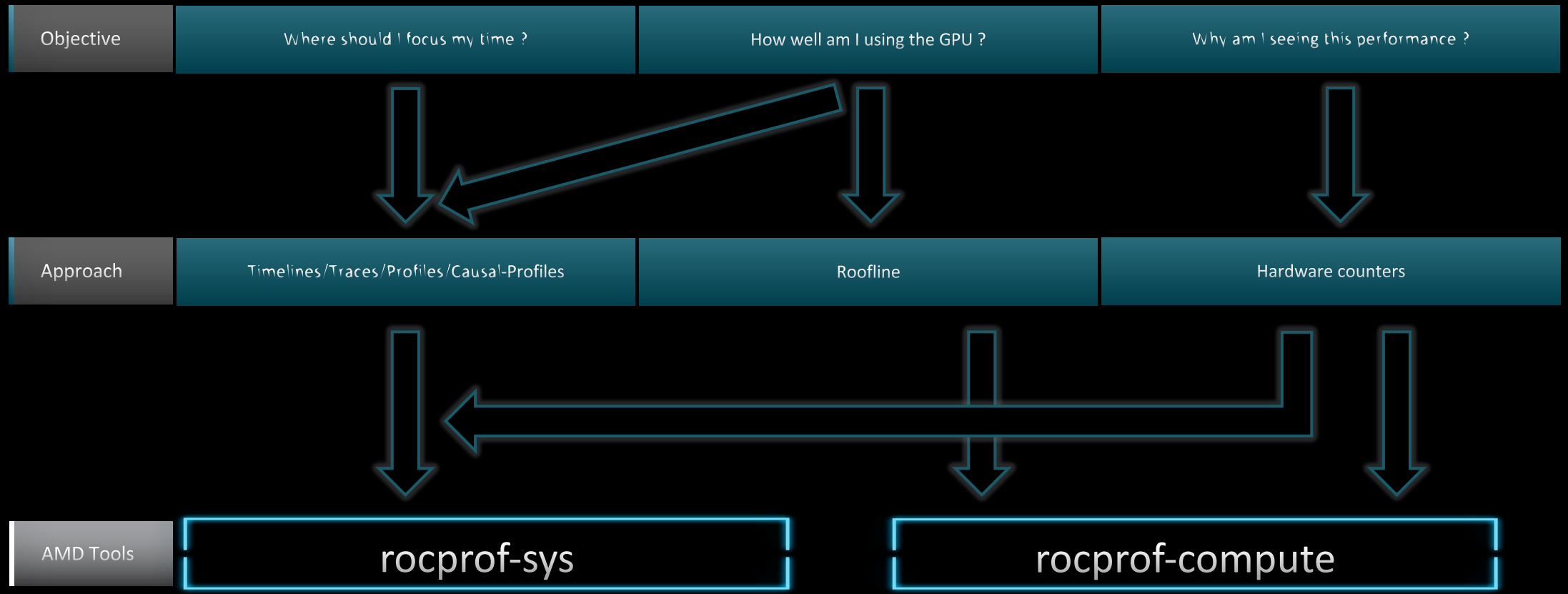
[Public]

Background – AMD Profilers



[Public]

Background – AMD Profilers



[Public]

AMD Profilers with Timeline Profiling Support

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 to GPU copy
  - HIP API
  - HSA API
  - GPU Kernels

rocprof-sys

Trace collection

- Hardware counters (rocprof)
- Sampling
- Runtime instrumentation with standard executable
- Binary rewrite Pre-instruments executable

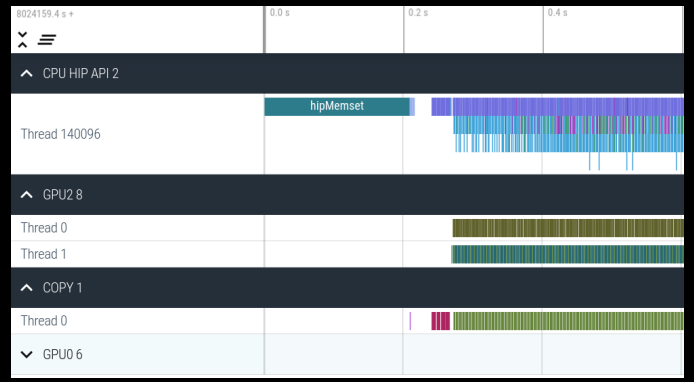
Adds to rocprof

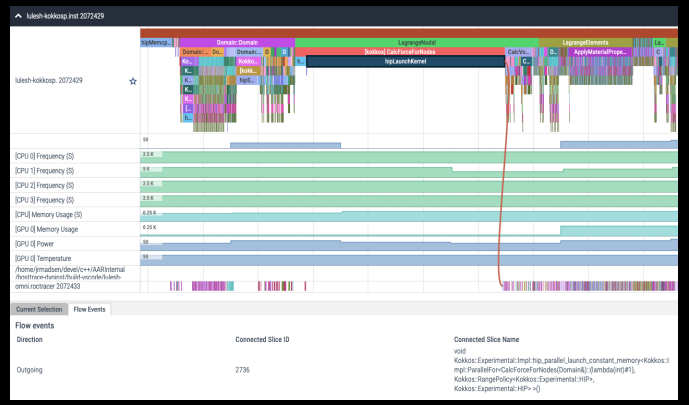
- PAPI
- DynInst
- Perf
- Profile
- uProf
- OpenMP®
- MPI
- Kokkos
- p-threads
- rocm-smi

Visualization

- Traces visualized with Perfetto

	A	B	C	D	E
1	Name	Calls	TotalDura	AverageN	Percentag
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

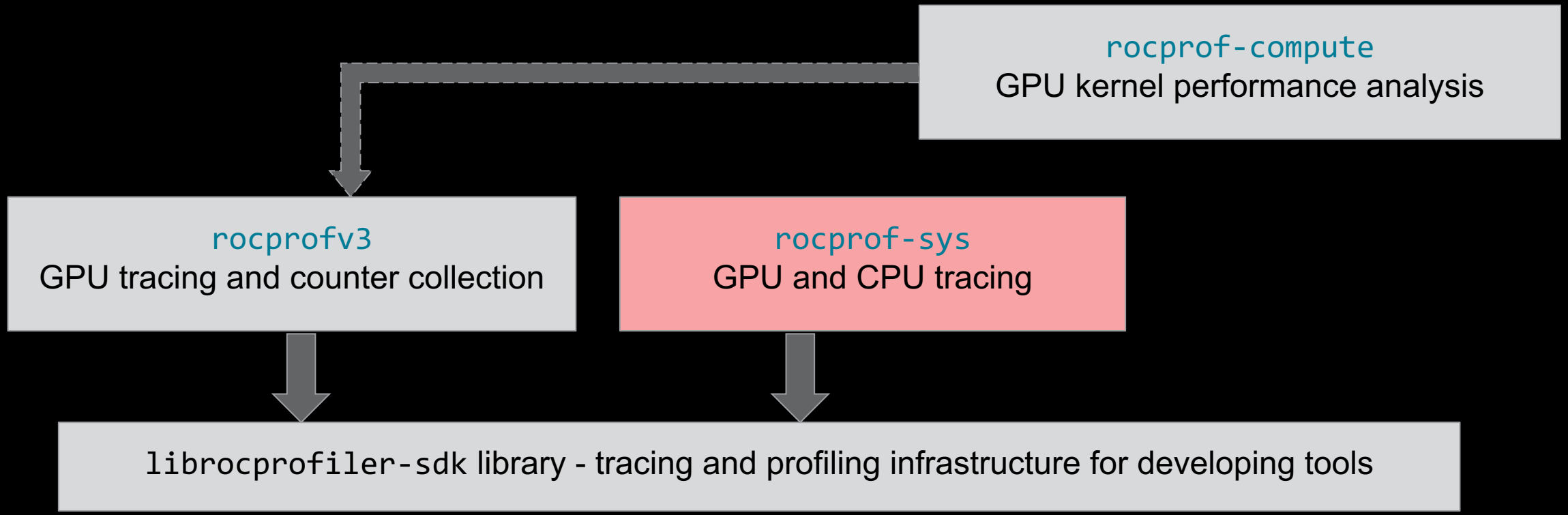




AMD
together we advance_

[Public]

ROCm Systems Profiler Environment



[Public]

ROCm Systems Profiler (`rocprof-sys`)

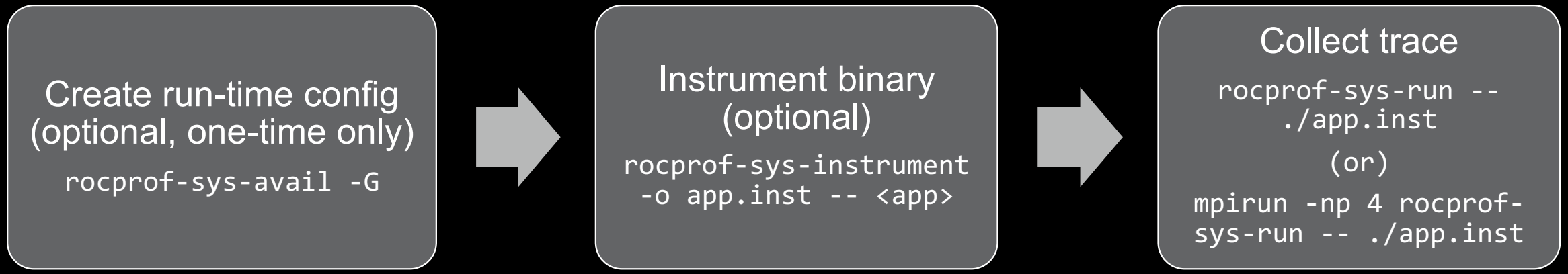
- Profiling and comprehensive tracing of applications on CPU and GPU
- Several data collection modes: sampling, dynamic instrumentation, binary rewrite, causal profiling, etc.
- Collect CPU and GPU metrics
- Visualization format: protobuf files (`.proto`) viewed in Perfetto

```
$ rocprof-sys-run --profile --trace --include ompt -- <app with arguments>
```



[Public]

Typical rocprof-sys workflow



- `rocprof-sys` documentation: https://rocm.docs.amd.com/projects/rocprofiler-systems/en/latest/index.html
- Run-time configuration parameters: https://rocm.docs.amd.com/projects/rocprofiler-systems/en/latest/how-to/configuring-runtime-options.html

[Public]

rocprof-sys: Configuration file

Create a config file in $HOME:

```
$ rocprof-sys-avail -G $HOME/.rocprofsys.cfg
```

Modify the config file to tune runtime settings:

<snip>

```
ROCPROFSYS_CONFIG_FILE                          =
ROCPROFSYS_TRACE                                = true
ROCPROFSYS_TRACE_DELAY                          = 0
ROCPROFSYS_TRACE_DURATION                       = 0
ROCPROFSYS_TRACE_PERIOD_CLOCK_ID                = CLOCK_REALTIME
ROCPROFSYS_TRACE_PERIODS                        =
ROCPROFSYS_PROFILE                              = false
ROCPROFSYS_USE_SAMPLING                         = false
```

<snip>

Contents of the config file

Declare which config file to use by setting the environment:

```
$ export ROCPROFSYS_CONFIG_FILE=$HOME/.rocprofsys.cfg
```

[Public]

rocprof-sys: Binary rewrite

Binary Rewrite

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

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

```
$ rocprof-sys-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 500 or more cycles, add option "`-i 500`" (more overhead)

```
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libnss_files.so.2'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libnss_hesiod.so.2'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libpapi.so.6.0.0.0'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libpthread.so.0'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libresolv.so.2'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/librt.so.1'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.30'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libthread_db.so.1'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libutil.so.1'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libz.so.1.2.11'...
[rocprof-sys][exe] [internal] parsing library: '/usr/lib/x86_64-linux-gnu/libzstd.so.1.4.8'...
[rocprof-sys][exe] [internal] binary info processing required 2.947 sec and 662.972 MB
[rocprof-sys][exe] Processing 9 modules...
[rocprof-sys][exe] Processing 9 modules... Done (0.003 sec, 0.288 MB)
[rocprof-sys][exe] Found 'MPI_Init' in '/home/sysadmin/HPCTrainingExamples/HIP/jacobi/Jacobi_hip'. Enabling MPI support...
[rocprof-sys][exe] Finding instrumentation functions...
[rocprof-sys][exe] Outputting 'rocprofsys-Jacobi_hip.inst-output/instrumentation/available.json'... Done
[rocprof-sys][exe] Outputting 'rocprofsys-Jacobi_hip.inst-output/instrumentation/available.txt'... Done
[rocprof-sys][exe] Outputting 'rocprofsys-Jacobi_hip.inst-output/instrumentation/instrumented.json'... Done
[rocprof-sys][exe] Outputting 'rocprofsys-Jacobi_hip.inst-output/instrumentation/instrumented.txt'... Done
[rocprof-sys][exe] Outputting 'rocprofsys-Jacobi_hip.inst-output/instrumentation/excluded.json'... Done
[rocprof-sys][exe] Outputting 'rocprofsys-Jacobi_hip.inst-output/instrumentation/excluded.txt'... Done
[rocprof-sys][exe] Outputting 'rocprofsys-Jacobi_hip.inst-output/instrumentation/overlapping.json'... Done
[rocprof-sys][exe] Outputting 'rocprofsys-Jacobi_hip.inst-output/instrumentation/overlapping.txt'... Done
[rocprof-sys][exe]
[rocprof-sys][exe] The instrumented executable image is stored in '/home/sysadmin/HPCTrainingExamples/HIP/jacobi/Jacobi_hip.inst'
[rocprof-sys][exe] Getting linked libraries for /home/sysadmin/HPCTrainingExamples/HIP/jacobi/Jacobi_hip...
[rocprof-sys][exe] Consider instrumenting the relevant libraries...
[rocprof-sys][exe]
[rocprof-sys][exe]      /opt/rocmplus-6.3.1/openmpi-5.0.6-ucc-1.3.0-ucx-1.17.0/lib/libmpi.so.40
[rocprof-sys][exe]      /opt/rocm-6.3.1/lib/llvm/bin/../../../lib/libroctx64.so.4
[rocprof-sys][exe]      /opt/rocm-6.3.1/lib/llvm/bin/../../../lib/libroctracer64.so.4
[rocprof-sys][exe]      /opt/rocm-6.3.1/lib/llvm/bin/../../../lib/libamdhip64.so.6
[rocprof-sys][exe]      /lib/x86_64-linux-gnu/libstdc++.so.6
[rocprof-sys][exe]      /lib/x86_64-linux-gnu/libm.so.6
[rocprof-sys][exe]      /lib/x86_64-linux-gnu/libgcc_s.so.1
[rocprof-sys][exe]      /lib/x86_64-linux-gnu/libc.so.6
```

Path to new instrumented binary

[Public]

rocprof-sys: Run instrumented binary

Binary Rewrite

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

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

```
$ rocprof-sys-instrument -o Jacobi_hip.inst --
./Jacobi_hip
```

Run the instrumented binary:

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

To instrument routines with 500 or more instructions, add option "`-i 500`"

Binary rewrite is recommended for runs with multiple ranks as `rocprof-sys` produces separate output files for each rank

```
ROCPROFSYS: HSA_TOOLS_LIB=/opt/rocm-6.3.1/lib/librocprof-sys-dl.so.0.1.0
ROCPROFSYS: HSA_TOOLS_REPORT_LOAD_FAILURE=1
ROCPROFSYS: LD_PRELOAD=/opt/rocm-6.3.1/lib/librocprof-sys-dl.so.0.1.0
ROCPROFSYS: OMP_TOOL_LIBRARIES=/opt/rocm-6.3.1/lib/librocprof-sys-dl.so.0.1.0
ROCPROFSYS: ROCP_HSA_INTERCEPT=1
ROCPROFSYS: ROCP_TOOL_LIB=/opt/rocm-6.3.1/lib/librocprof-sys.so.0.1.0
[rocprof-sys][dl][292290] rocprofsys_main
[rocprof-sys][292290][rocprofsys_init_tooling] Instrumentation mode: Trace

 ___   ___  ___  ___  
```

[Public]

rocprof-sys: Kernel durations

```
$ cat rocprofsys-Jacobi_hip.inst-output/2025-01-21_07.40/wall_clock-0.txt
```

Enable `ROCPROFSYS_PROFILE` in your config file

```
...
ROCPROFSYS_PROFILE                                    = true
...
```

then re-run. Alternatively, prepend `ROCPROFSYS_PROFILE=true` to the `mpirun` command:

Durations

Call Stack

	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

[Public]

rocprof-sys: Kernel durations – flat profile

Edit your config file (or prepend to your `mpirun` command):

```
ROCPROFSYS_PROFILE                = true
ROCPROFSYS_FLAT_PROFILE           = true
```

Use flat profile to see aggregate duration of kernels and functions

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

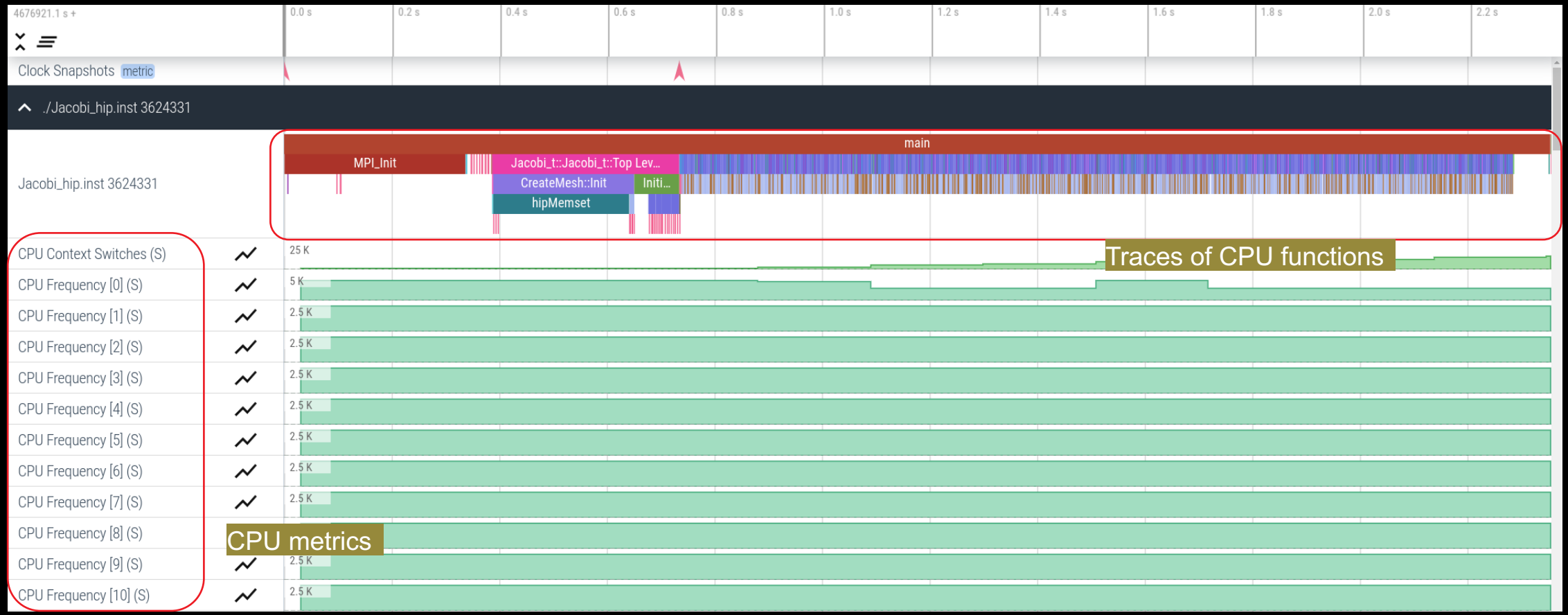
LABEL	COUNT	DEPTH	METRIC	UNITS	SUM	MEAN	MIN	MAX	VAR	STDDEV	% SELF
|0>>> main	1	0	wall_clock	sec	82.739099	82.739099	82.739099	82.739099	0.000000	0.000000	100.0
|0>>> MPI_Init	1	0	wall_clock	sec	34.056610	34.056610	34.056610	34.056610	0.000000	0.000000	100.0
|0>>> pthread_create	3	0	wall_clock	sec	0.014644	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	-	-	-	-	-	-	-	-	-

[Public]

Visualizing trace (1/3)

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`

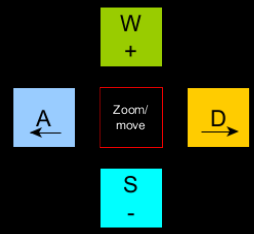


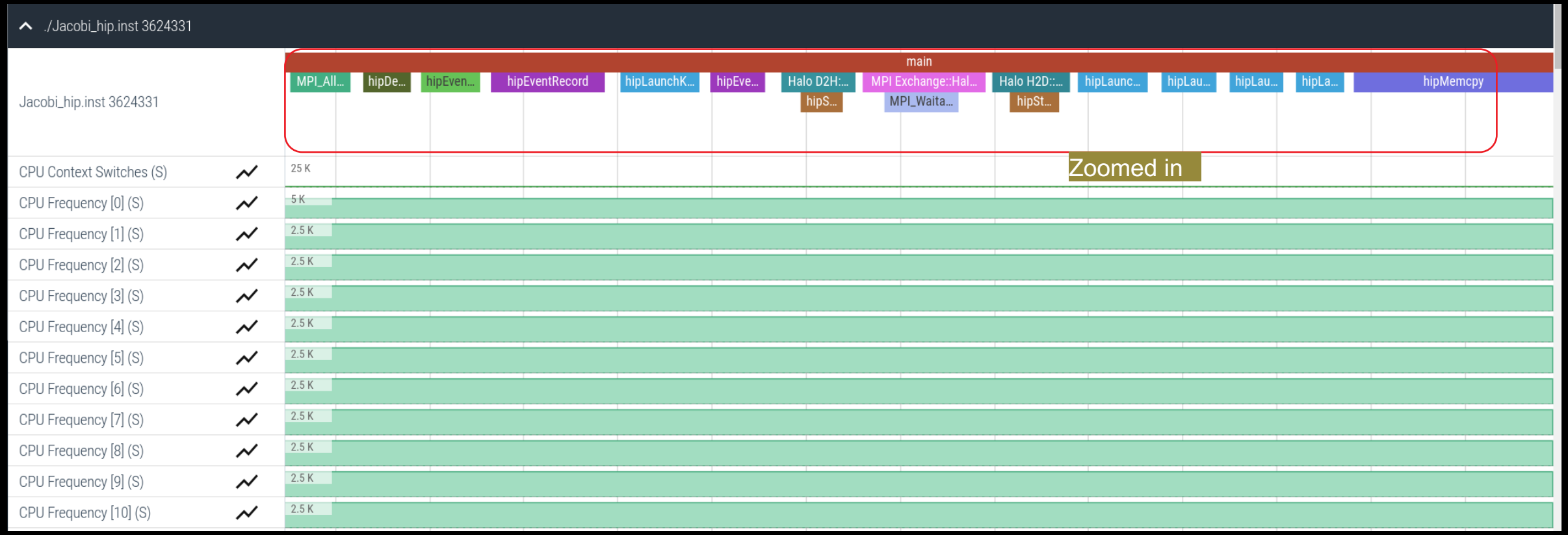
[Public]

Visualizing trace (2/3)

Use Perfetto

Zoom in to investigate regions of interest



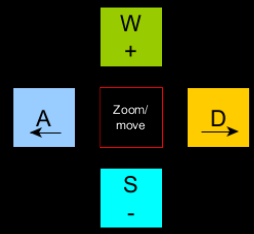


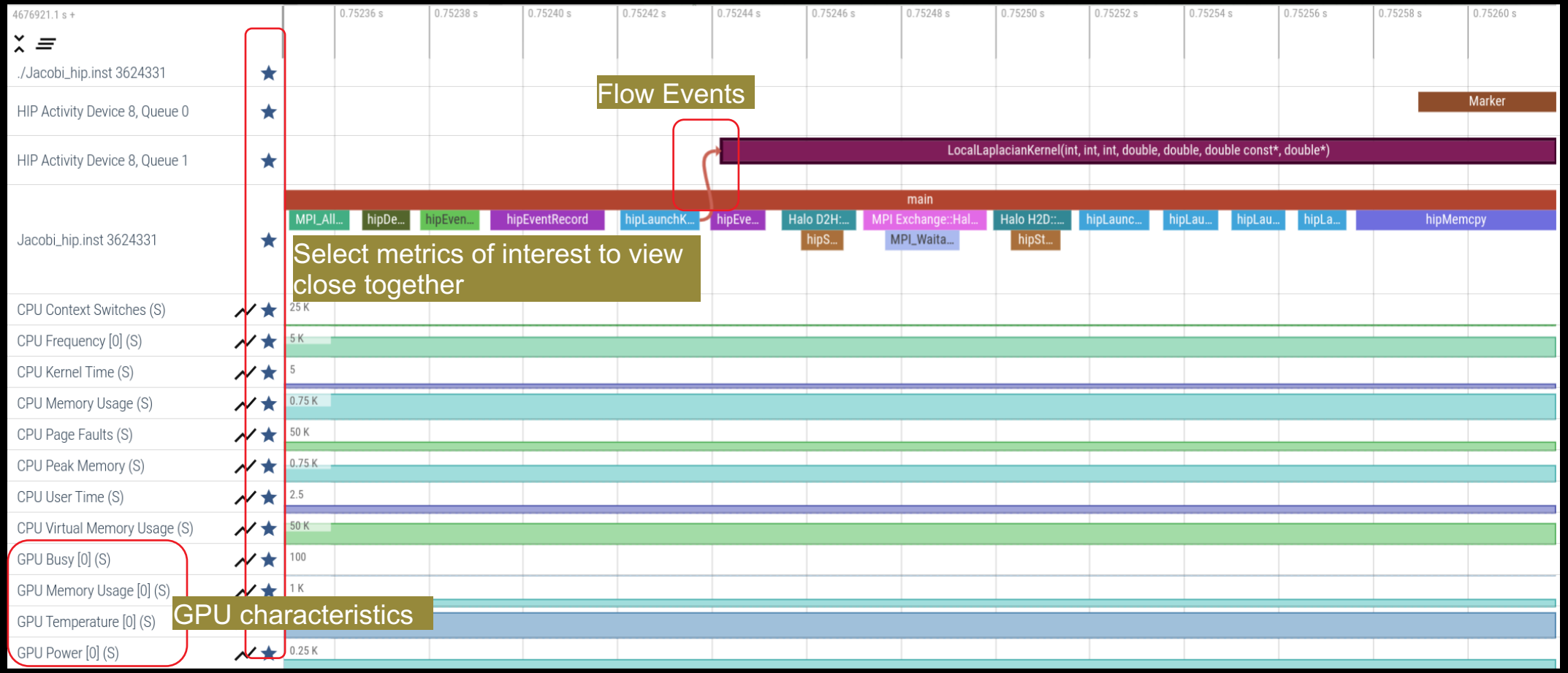
[Public]

Visualizing trace (3/3)

Use Perfetto

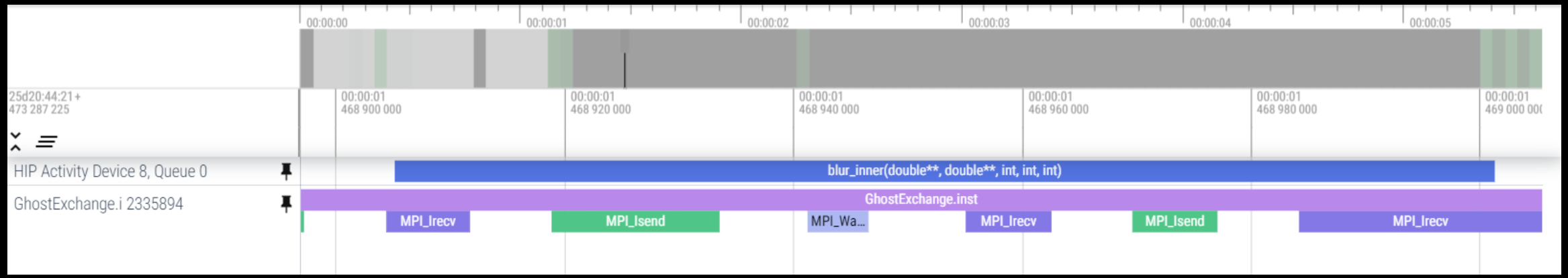
Zoom in to investigate regions of interest





[Public]

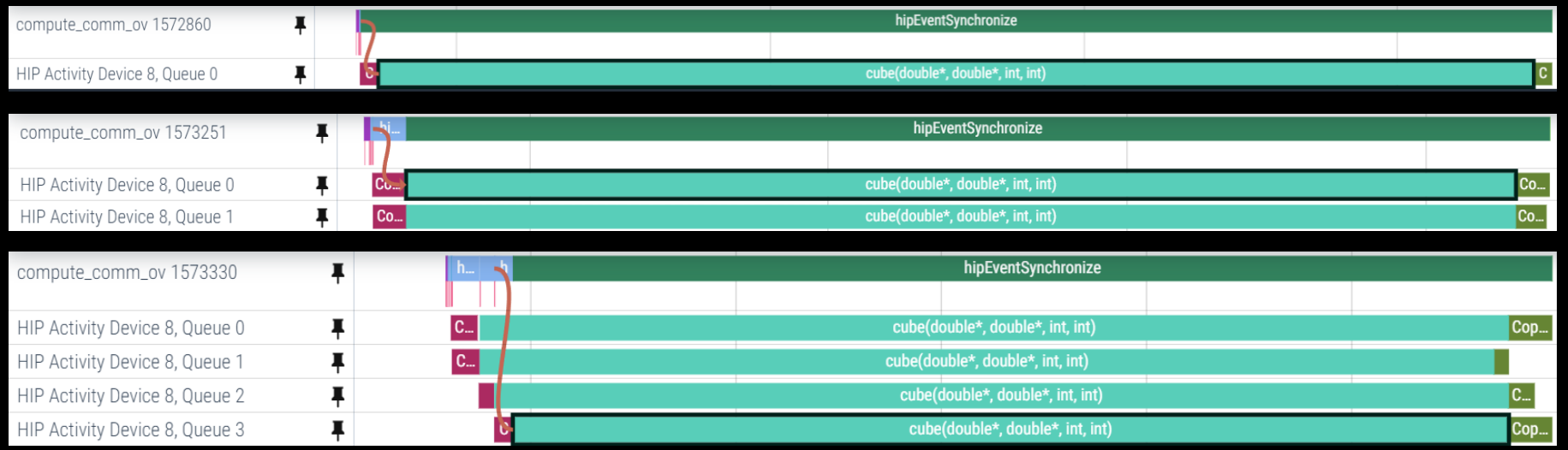
rocprof-sys: Study overlap between compute & communication

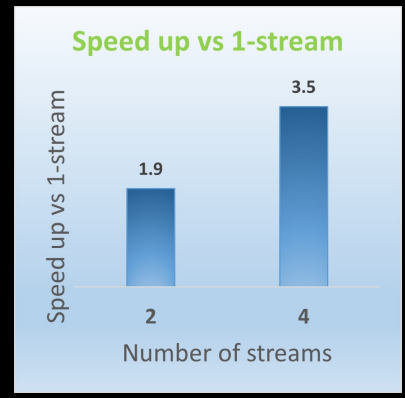


[Public]

rocprof-sys: Study concurrent kernels (Stream_Overlap)

- Example of GPU kernels launched into multiple HIP streams





Good overlap between multiple streams saves multifold total compute time

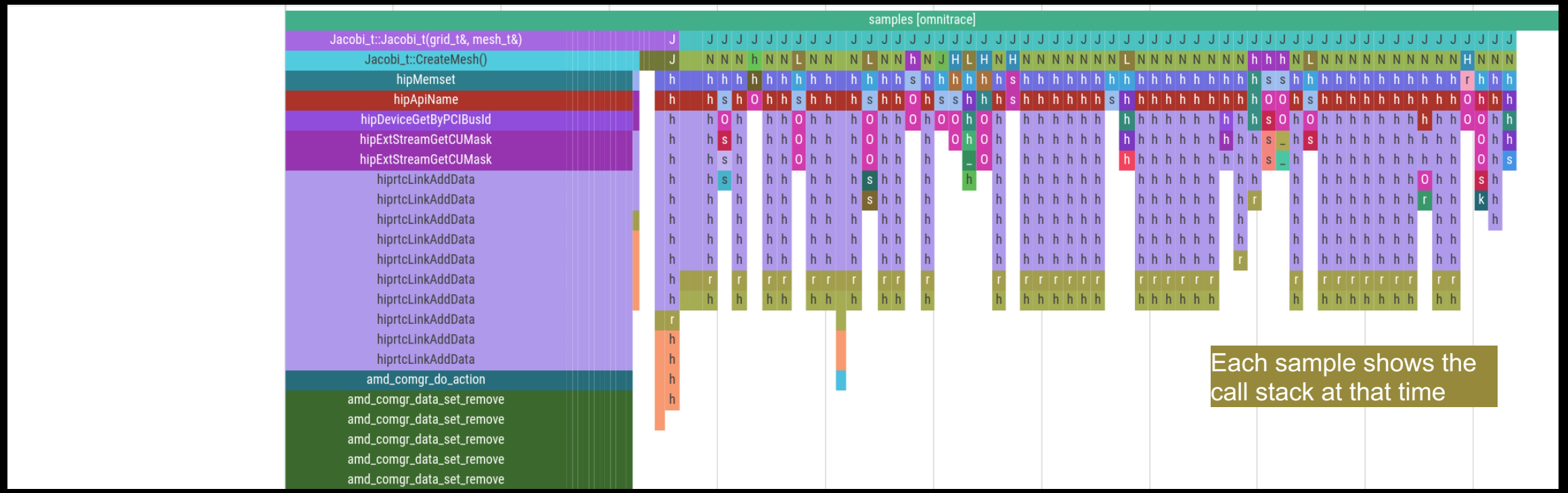
```
cd /path/to/HPCTrainingExamples/HIP/Stream_Overlap/0-Orig
mkdir build; cd build; cmake ..; make -j

./compute_comm_overlap <nstreams> <block size (optional, default: 64)>
rocprof-sys-instrument -o compute_comm_overlap.inst -- compute_comm_overlap
rocprof-sys-run -- ./compute_comm_overlap.inst <nstreams>
```

[Public]

rocprof-sys: Sampling CPU call-stack (1/2)

- ROCPROFSYS_USE_SAMPLING = true; ROCPROFSYS_SAMPLING_FREQ = 100 (100 samples per second)
- Alternatively run with rocprof-sys-sample

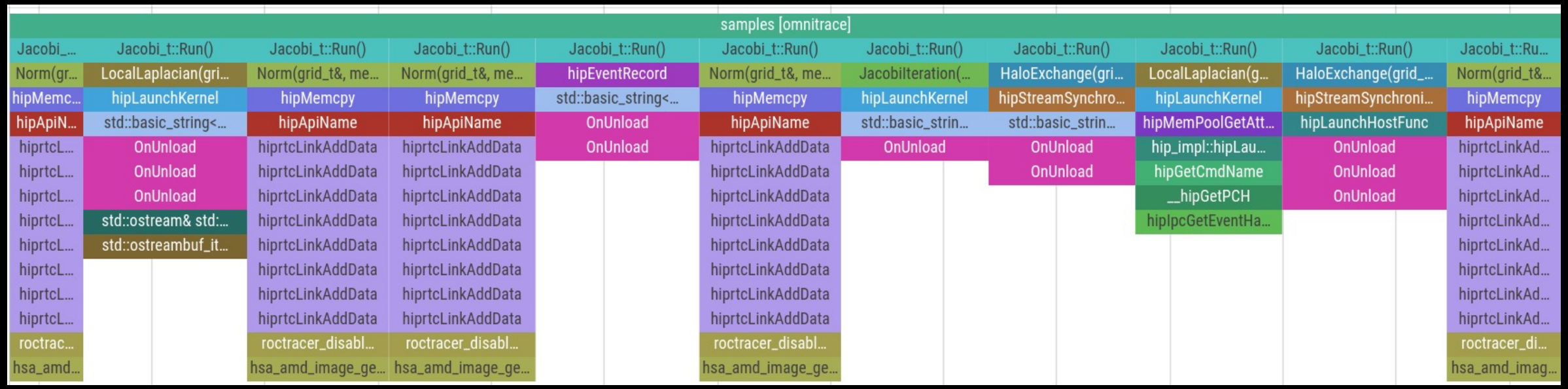


Scroll down all the way in Perfetto to see the sampling output

[Public]

rocprof-sys: Sampling CPU call-stack (2/2)

Zoom in call-stack sampling



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

[Public]

rocprof-sys: Network performance profiling (in ROCm 6.4)

- First identify NIC information: `rocprof-sys-avail -H -r net`

```
|---------------------------------|---------|-----------|---------------------------------|
|        HARDWARE COUNTER         | DEVICE  | AVAILABLE |             SUMMARY             |
|---------------------------------|---------|-----------|---------------------------------|

...
| net:::hsn0:rx:byte              |   CPU   |   true    | hsn0 receive byte               |
| net:::hsn0:rx:packet            |   CPU   |   true    | hsn0 receive packet             |
| net:::hsn0:rx:error             |   CPU   |   true    | hsn0 receive error              |
```

- Set up parameters for profiling:

Collecting PAPI counters requires /proc/sys/kernel/perf_event_paranoid to be <= 2 (supported on LUMI)

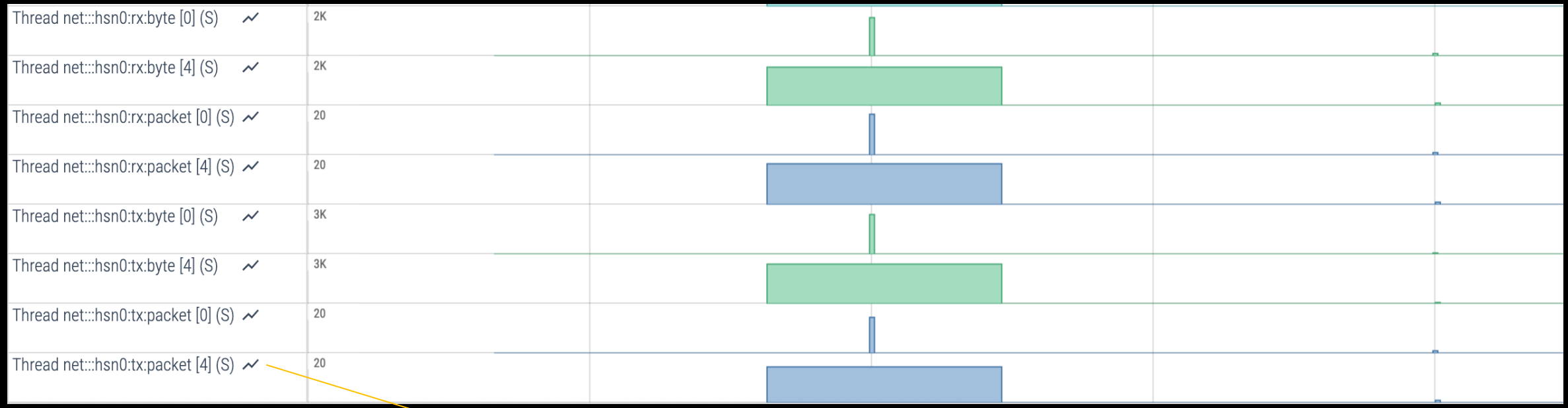
```
ROCPROFSYS_NETWORK_INTERFACE=hsn0
ROCPROFSYS_PAPI_EVENTS=net:::hsn0:rx:byte net:::hsn0:rx:packet net:::hsn0:tx:byte net:::hsn0:tx:packet
ROCPROFSYS_TIMEMORY_COMPONENTS=wall_clock network_stats
```

- Requires CPU sampling to be enabled (limitation to be removed soon):

```
ROCPROFSYS_USE_SAMPLING=true
```

[Public]

rocprof-sys: Network performance profiling (in ROCm 6.4) contd.



- Overhead can be reduced by decreasing sampling frequency and reducing number of events collected

```
ROCPROFSYS_SAMPLING_FREQ=500
```

- More details in rocprof-sys documentation: Network performance profiling

AMD
together we advance_

[Public]

`rocprof-sys`: Tips & tricks

- If `rocprof-sys` does nothing – check app or environment
  - Did you forget to add "--" between `rocprof-sys-*` and `<app>`?
  - In a Slurm environment, sometimes incorrect order of loading libraries at runtime causes unexpected behavior
  - If application fails when running with `sbatch`, try profiling interactively using `srun`

- Perfetto visualization has known limitations
  - Max file size 4GB
  - To visualize large `.proto` files (approx. >1GB), load into memory first using Perfetto's `trace_processor`
  - If the latest Perfetto UI does not work, try the older version https://ui.perfetto.dev/v46.0-35b3d9845/#!/

- Disable options not useful for your analysis (e.g., `ROCPROFSYS_SAMPLING_CPUS=none`)

- To collect OpenMP traces add `--include ompt` argument to `rocprof-sys-run`

[Public]

Additional features

- Dynamic runtime instrumentation to instrument dependent libraries too
- `ROCPROFSYS_USE_KOKKOSP=true` supports Kokkos profiling
- `rocprof-sys-causal` for causal profiling (experimental)
- User API to control instrumentation

- Initial Fortran OpenMP® offload tracing available, full support coming in ROCm 7.1

- Changelog and release notes for reference: https://github.com/ROCm/rocm-systems/blob/develop/projects/rocprofiler-systems/CHANGELOG.md

[Public]

Summary

- rocprof-sys - powerful tool to understand CPU + GPU activity on AMD GPUs
  - Ideal for an initial look at how an application runs
  - Easy to visualize traces in Perfetto

- Leverages several other tools and combines their data into a comprehensive output files
  - Some tools used are AMDμProf, rocprofiler, rocm-smi, roctracer, perf, etc.

- **Helps users analyze overlaps between CPU/GPU compute and communication**

[Public]

Hands-on exercises

https://hackmd.io/@sfantao/lumi-training-cph-2026#Rocprof-sys

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 **rocprofiler-systems** directory.

Link to instructions on how to run the tests: **rocprofiler-systems/Jacobi/README.md**

[Public]

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.

© 2026 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.
The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board
Windows is a registered trademark of Microsoft Corporation in the US and/or other countries.
Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries


AMD