



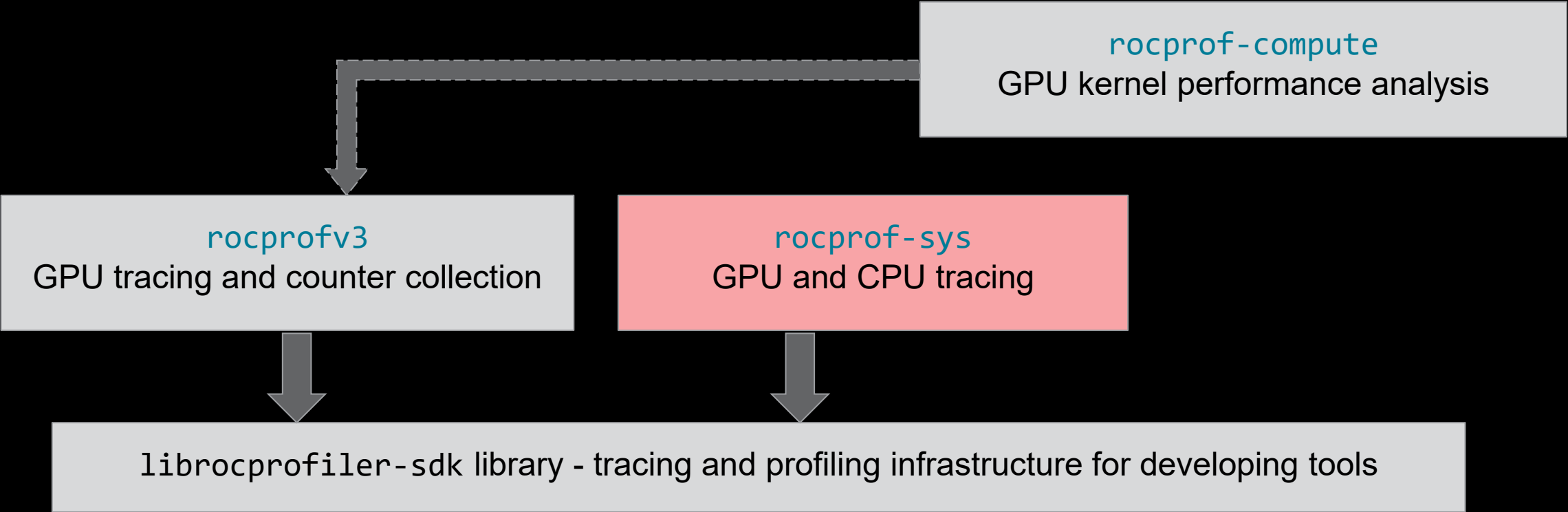
ROCprof-sys

GPU and CPU Timeline Profiling

Presenter: Bob Robey
Oct 21-23, 2025
AMD @ Tsukuba University

AMD 
together we advance_

AMD has three GPU profiling tools



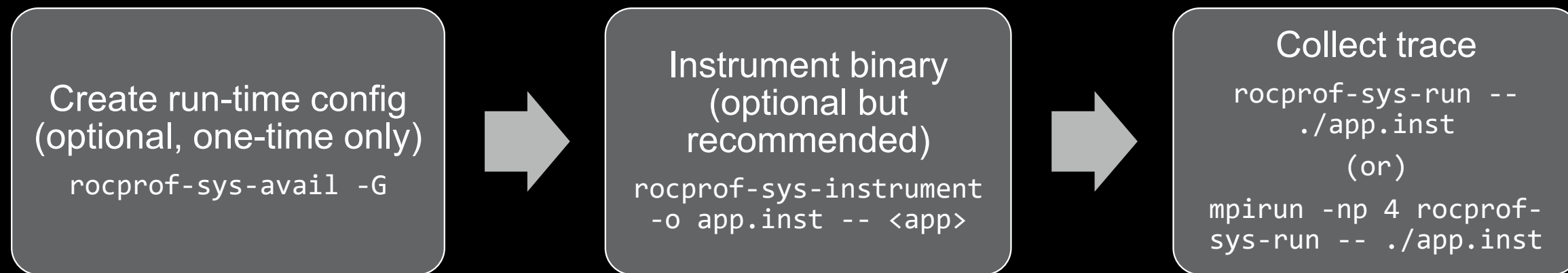
ROCm Systems Profiler (rocprof-sys)

- 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>



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>

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
```

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'...
[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] 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] /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

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: ROC_P_HSA_INTERCEPT=1
ROCPROFSYS: ROC_P_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
```

ROCM SYSTEMS PROFILER

```
rocprof-sys v0.1.0 (rev: b569c837e455f71dd76d06392d0b901ae927deca, x86_64-linux-gnu, compiler: GNU v11.4.0, rocm: v6.3.x)
[rocprof-sys][292290] /proc/sys/kernel/perf_event_paranoid has a value of 4. Disabling PAPI (requires a value <= 2)...
[rocprof-sys][292290] In order to enable PAPI support, run 'echo N | sudo tee /proc/sys/kernel/perf_event_paranoid' where N is <= 2
[630.094] perfetto.cc:47606 Configured tracing session 1, #sources:1, duration:0 ms, #buffers:1, total buffer size:1024000 KB,
[rocprof-sys][0][pid=292290] MPI rank: 0 (0), MPI size: 1 (1)
Topology size: 1 x 1
Local domain size (current node): 4096 x 4096
Global domain size (all nodes): 4096 x 4096
Rank 0 selecting device 0 on host e3797990608f
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: 1.5166 sec.
```

Generates traces for application run

Path to your output trace is printed by rocprof-sys-run:

```
[rocprofiler-systems][2100024][perfetto]> Outputting '/datasets/teams/dcgpu_training/amd/ssitaram/HPCTrainingExamples/HIP/jacobi/rocprofsys-Jacobi_hip.inst-output/2025-02-22_22.40/perfetto-trace-0.proto' (6223.06 KB / 6.22 MB / 0.01 GB)... Done
```

rocpf-sys: Kernel durations

```
$ cat rocprof-sys-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

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

rocprof-sys: Kernel durations – flat profile

```

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

ROCPROFSYS_PROFILE           = true
ROCPROFSYS_FLAT_PROFILE      = true
    
```

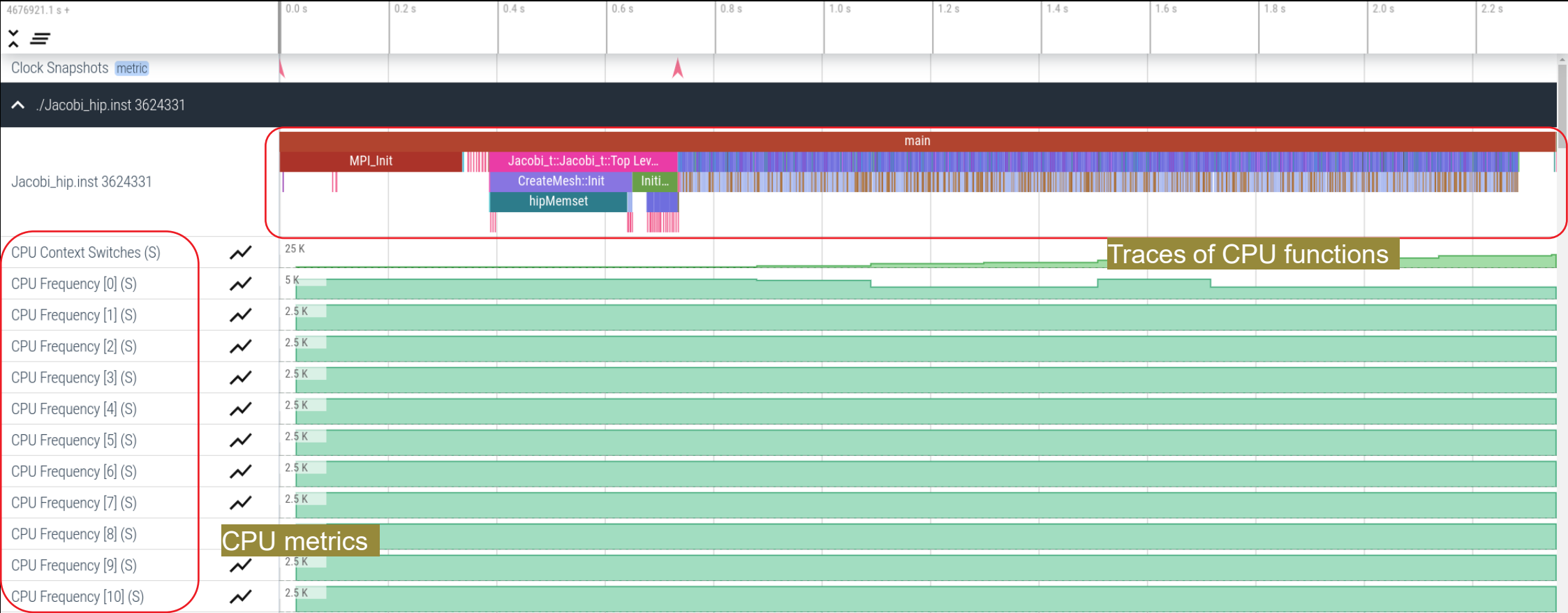
Use flat profile to see aggregate duration of kernels and functions

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	-	-	-	-	-	-	-	-	-

Visualizing trace (1/3)

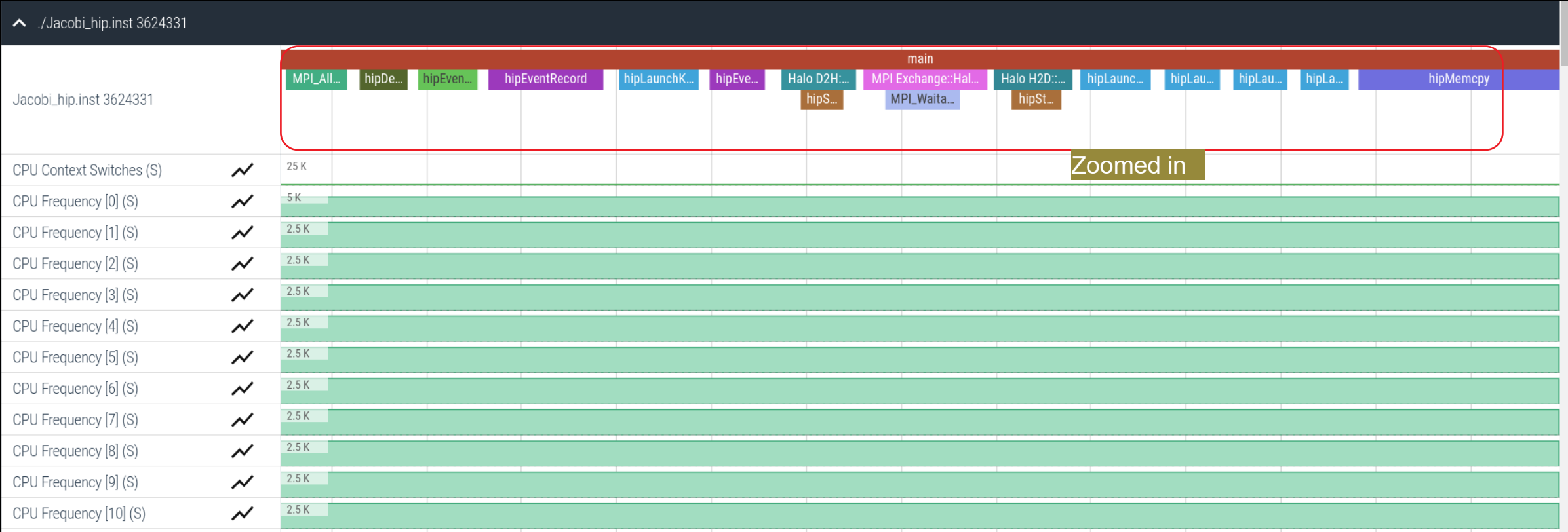
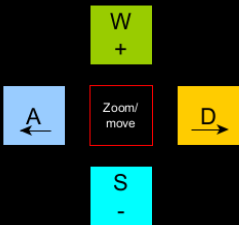
Use Perfetto

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



Visualizing trace (2/3)

Use Perfetto
Zoom in to investigate regions of interest



Visualizing trace (3/3)

Use Perfetto
Zoom in to investigate regions of interest

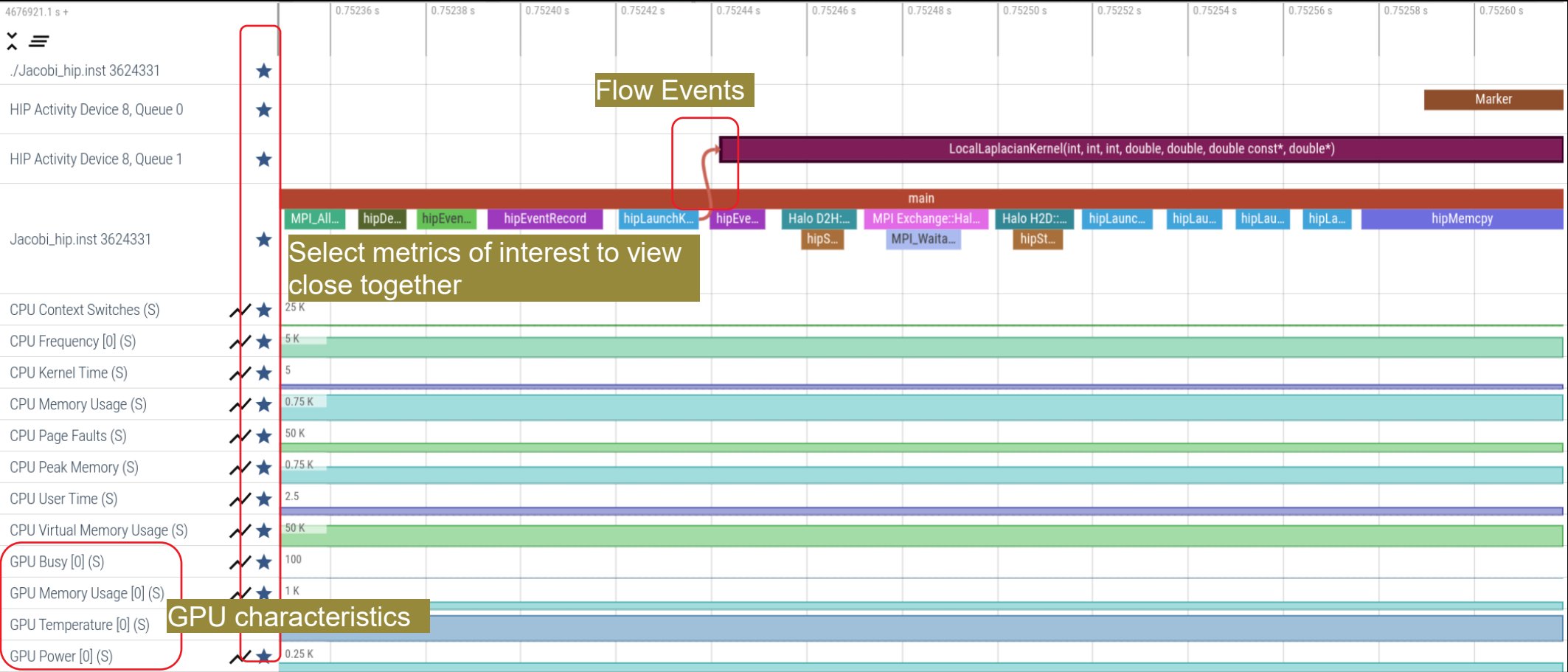
W
+

A

Zoom/
move

D

S
-

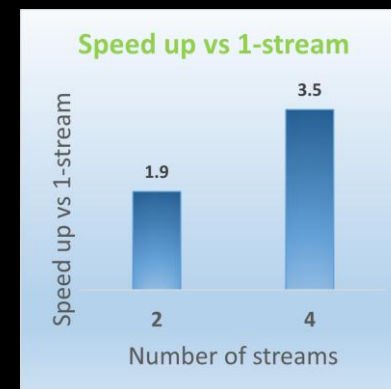
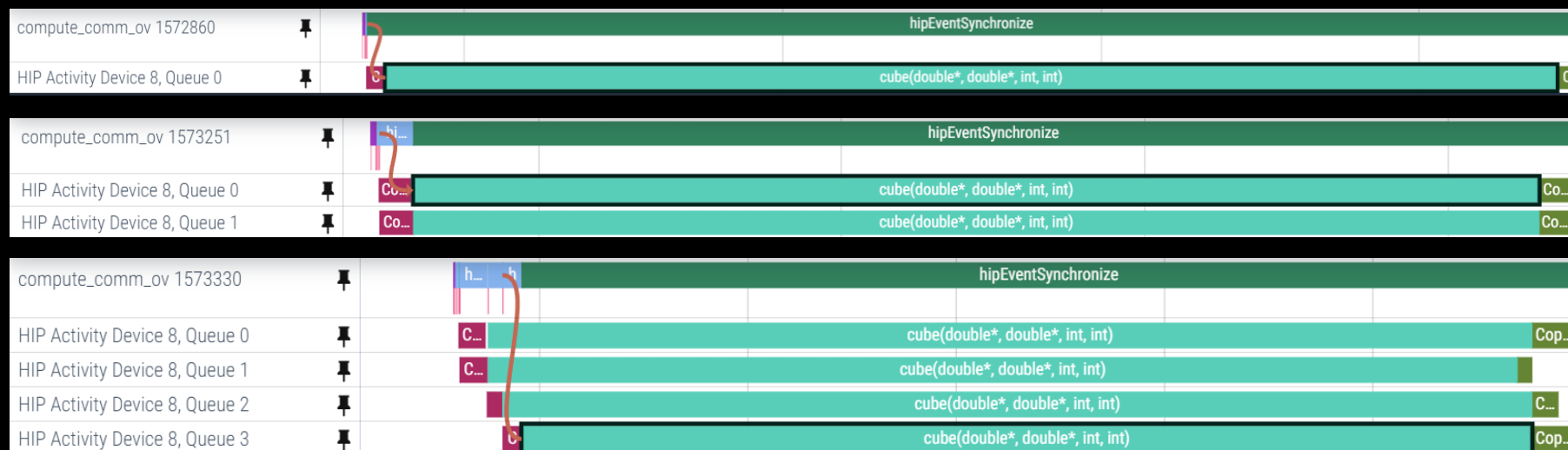


rocprof-sys: Study overlap between compute & communication



rocprof-sys: Study concurrent kernels (Stream_Overlap)

- Example of GPU kernels launched into multiple HIP streams



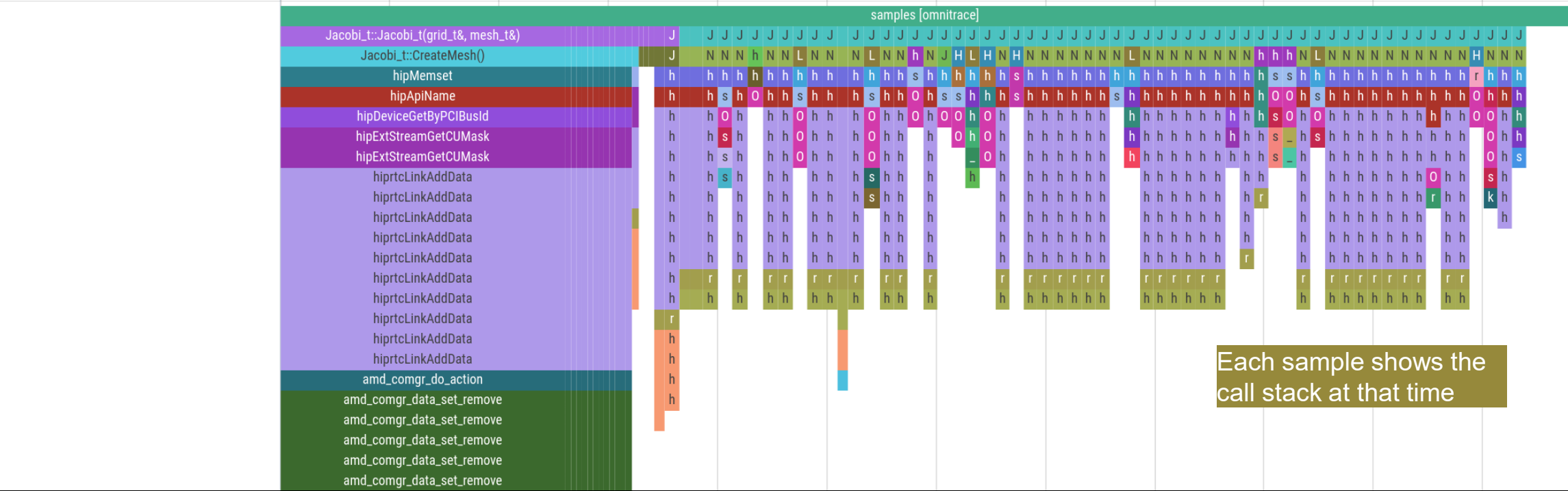
Good overlap between multiple streams saves multifold total compute time

```
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>
```

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

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

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...
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)

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:

```
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
```

Collecting PAPI counters requires
/proc/sys/kernel/perf_event_paranoid to be <= 2

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

```
ROCPROFSYS_USE_SAMPLING=true
```

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



A track per network counter collected

- Overhead can be reduced by increasing sampling frequency and reducing number of events collected
`ROCPROFSYS_SAMPLING_FREQ=500`

• More details in rocprof-sys documentation: [Network performance profiling](#)
Oct 21-23, 2025

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

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>

Summary

- **rocprof-sys**: Powerful tool to understand **GPU + CPU** activity
 - Ideal for an initial look at how an application runs
 - Analyze overlaps between CPU/GPU compute and communication
 - Obtain a comprehensive trace with system characteristics

Hands-on exercises

- Located in 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
- rocprof-sys exercises: [rocprofiler-systems/Jacobi/README.md](#) or [GhostExchange](#)
- Log into the AAC node and clone the repo:

```
ssh <username>@aac6.amd.com -p 7000 -i <path_to_ssh_key>  
git clone https://github.com/amd/HPCTrainingExamples.git  
module load rocm/6.4.0 rocprofiler-systems/6.4.0
```

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.

© 2025 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
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

