Introduction to Rocprof Profiling Tool

Suyash Tandon, Justin Chang, Julio Maia, Noel Chalmers, Paul T. Bauman, Nicholas Curtis, Nicholas Malaya, Alessandro Fanfarillo, Jose Noudohouenou, Chip Freitag, Damon McDougall, Noah Wolfe, Jakub Kurzak, Samuel Antao, George Markomanolis, Bob Robey

Comprehensive General LUMI Course
16/2/2023

AMD
together we advance_

AMD GPU Profiling

- ROC-profiler (or simply 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 the collection of application tracing and counter collection
  - Under constant development
- Provided in the ROCm releases
- The output of rocprof can be visualized using the chrome browser with Perfetto (https://ui.perfetto.dev/)

rocProf: Getting started + useful flags

- To get help:
  - `$ /opt/rocm-5.2.0/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: Collecting application traces

- rocProf can collect a variety of trace event types, and generate timelines in JSON format for use with Perfetto, currently:

Trace Event	rocprof Trace Mode
HIP API call	--hip-trace
GPU Kernels	--hip-trace
Host <-> Device Memory copies	--hip-trace
CPU HSA Calls	--hsa-trace
User code markers	--roctx-trace

- You can combine modes like --hip-trace --hsa-trace

rocProf: Information about the kernels

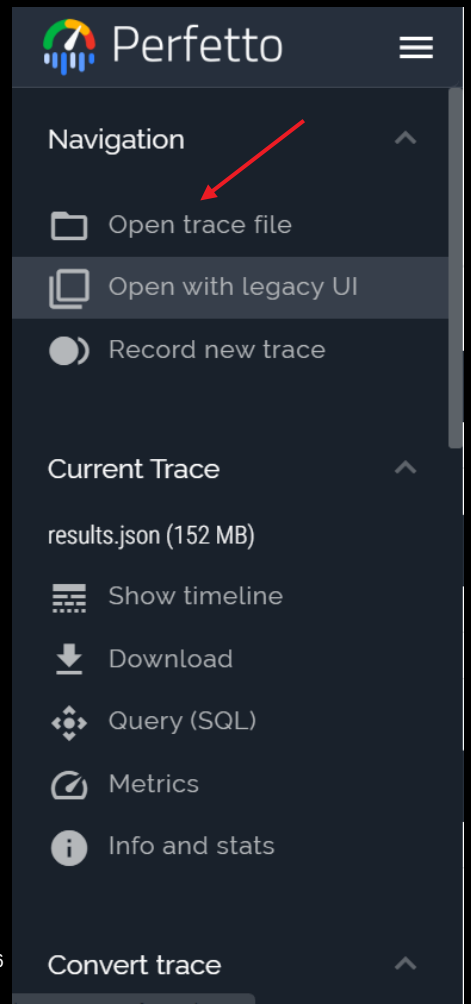
- rocprofiler can collect kernels information
  - **`$ /opt/rocm/bin/rocprof --stats --basenames on <app with arguments>`**
  - This will output two csv files, one with information per each call of the kenel *results.csv* and one with statistics grouped by each kernel *results.stats.csv.*
  - Content of results.stats.csv:

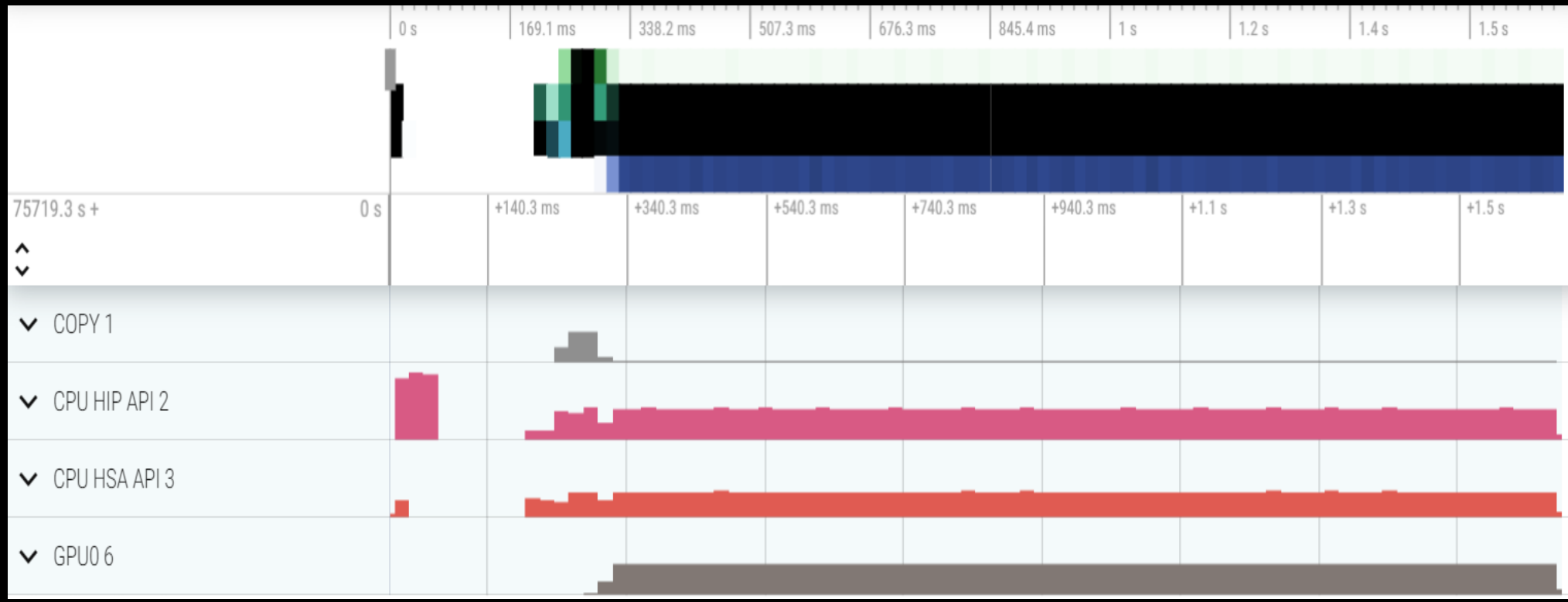
"Name",	"Calls",	"TotalDurationNs",	"AverageNs",	"Percentage"
"LocalLaplacianKernel",	1000,	817737586,	817737,	40.908259879301134
"JacobiIterationKernel",	1000,	699515425,	699515,	34.994060790890174
"NormKernel1",	1001,	454737348,	454283,	22.748756969583884
"HaloLaplacianKernel",	1000,	14561933,	14561,	0.7284773865206329
"NormKernel2",	1001,	12395374,	12382,	0.620092789636225
"__amd_rocclr_fillBufferAligned.kd",	1,	7040,	7040,	0.00035218406794656007

  - This way you know directly which kernels consume most of the time, it does not mean that the performance is slow, for now.

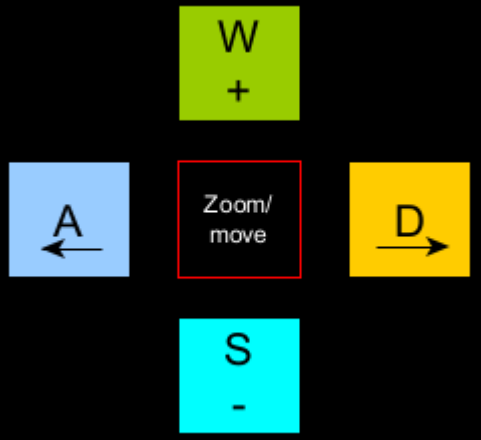
rocProf and Perfetto: Collecting and visualizing application traces

- rocprofiler can collect traces
  - `$ /opt/rocm/bin/rocprof --hip-trace --hsa-trace <app with arguments>`
  - This will output a .json file that can be visualized using the chrome browser and Perfetto (https://ui.perfetto.dev/)

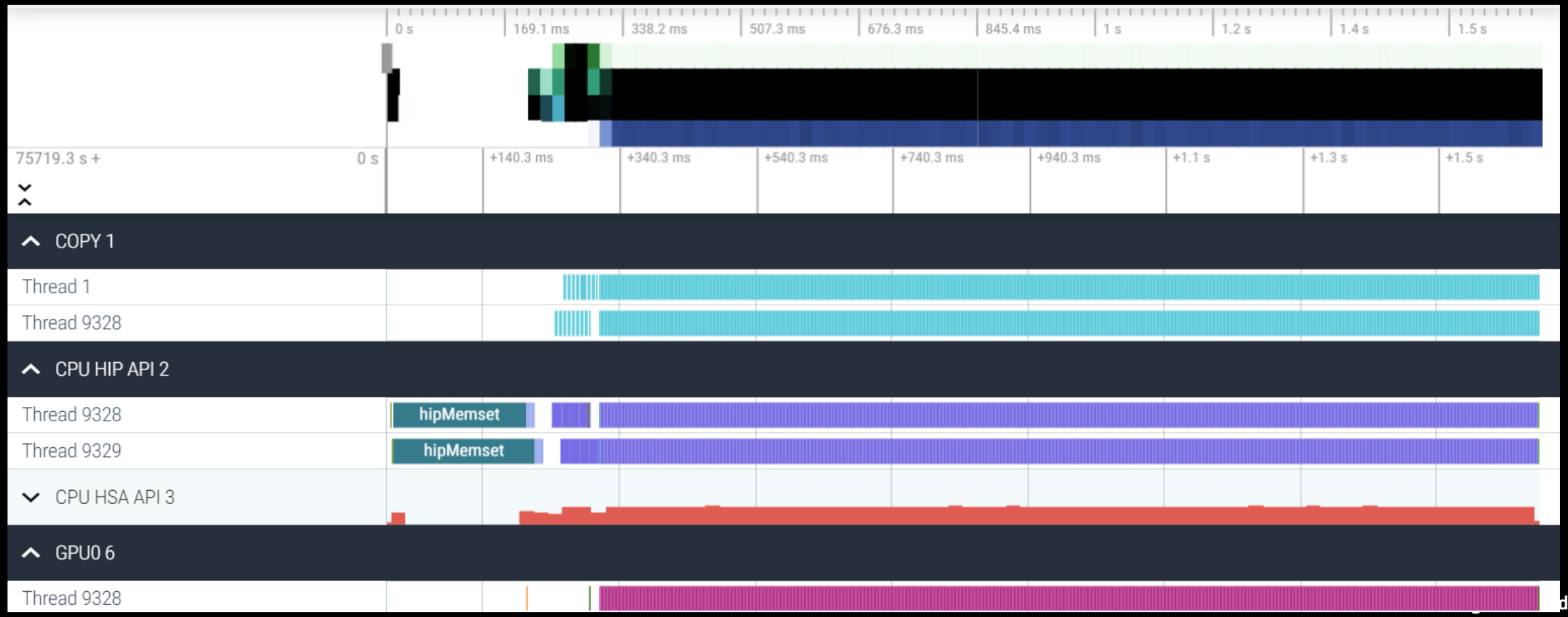




Perfetto: Visualizing application traces

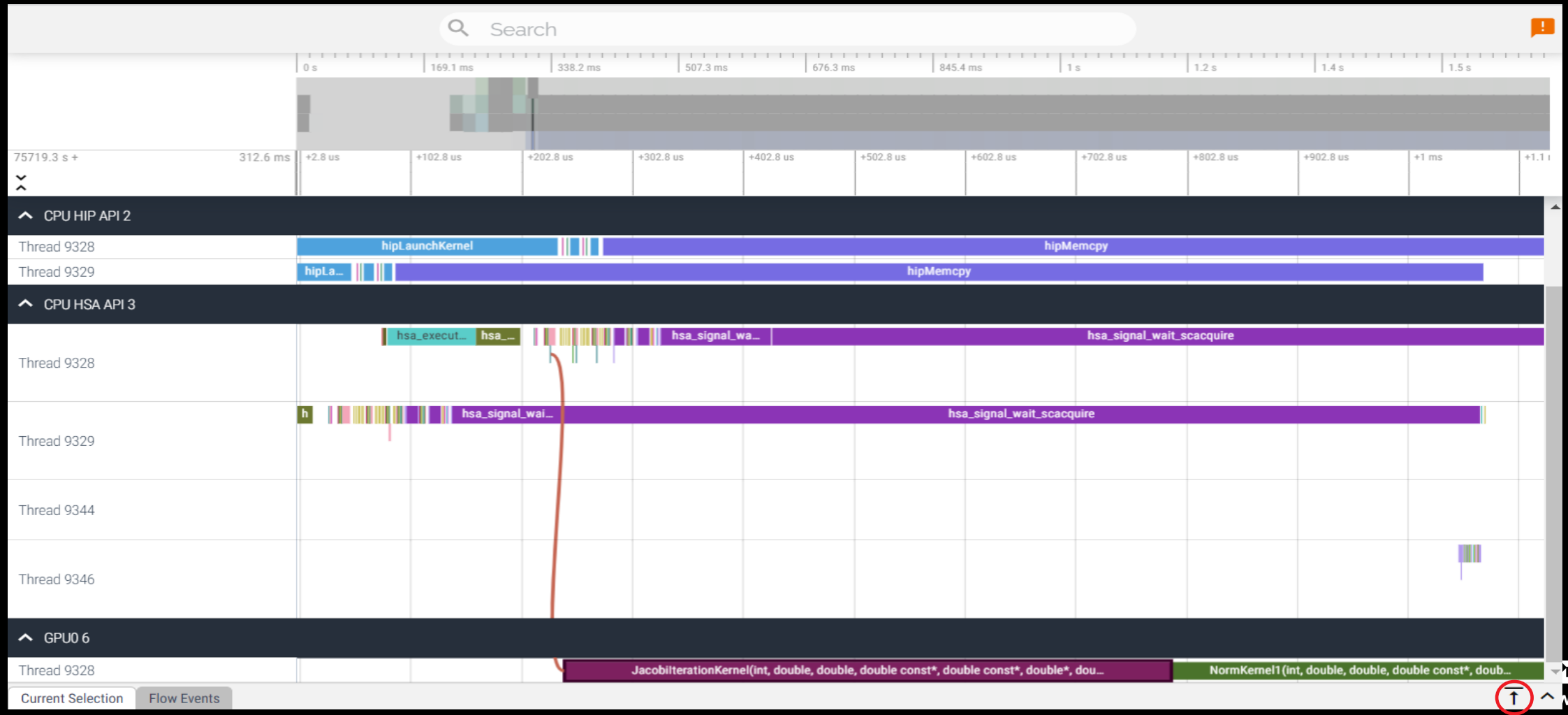


- We have expanded the COPY 1, CPU HIP API 2 and GPU0 6
- X axis is time and it displays events or counters.
- Handle the zoom by keystrokes: W zoom in, S zoom out, A move left, D move right

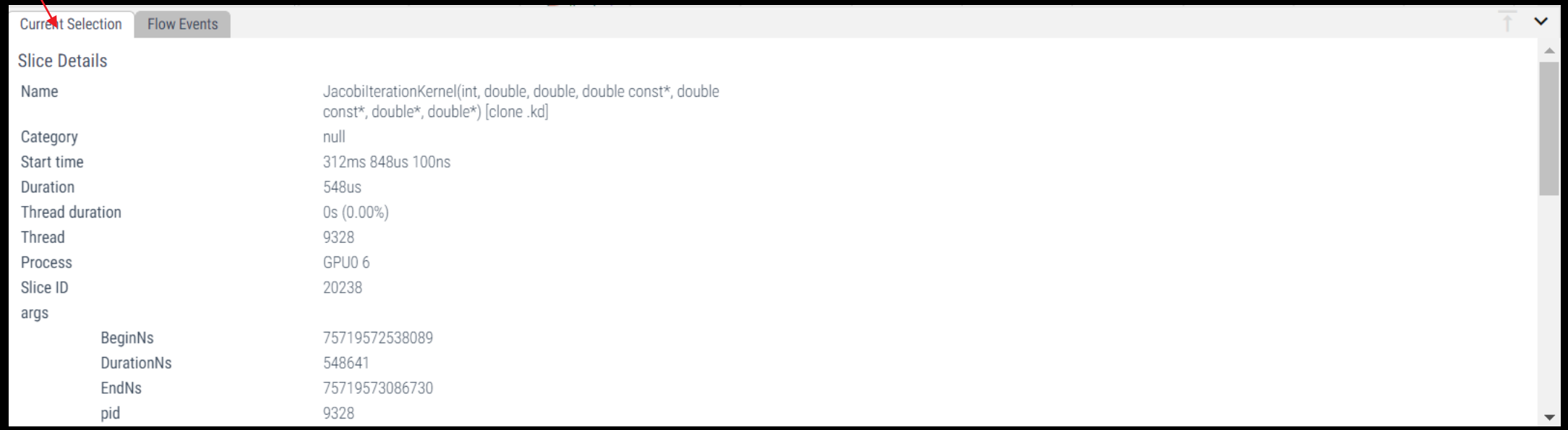


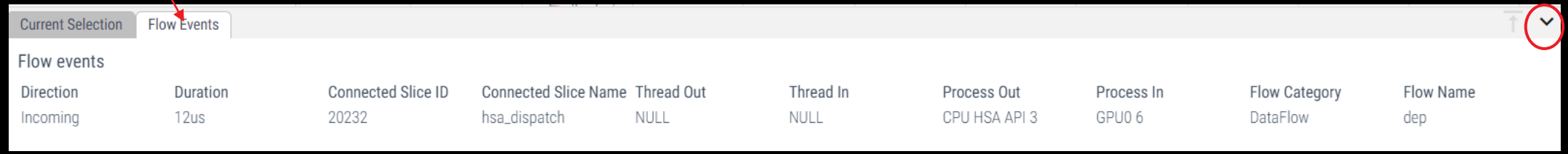
Perfetto: Kernel and flows

- Zoom and select a kernel, you can see the link to the HSA call enables the kernel
- Try to open the information for the kernel (button right down)



Perfetto: Information about kernels and flow events





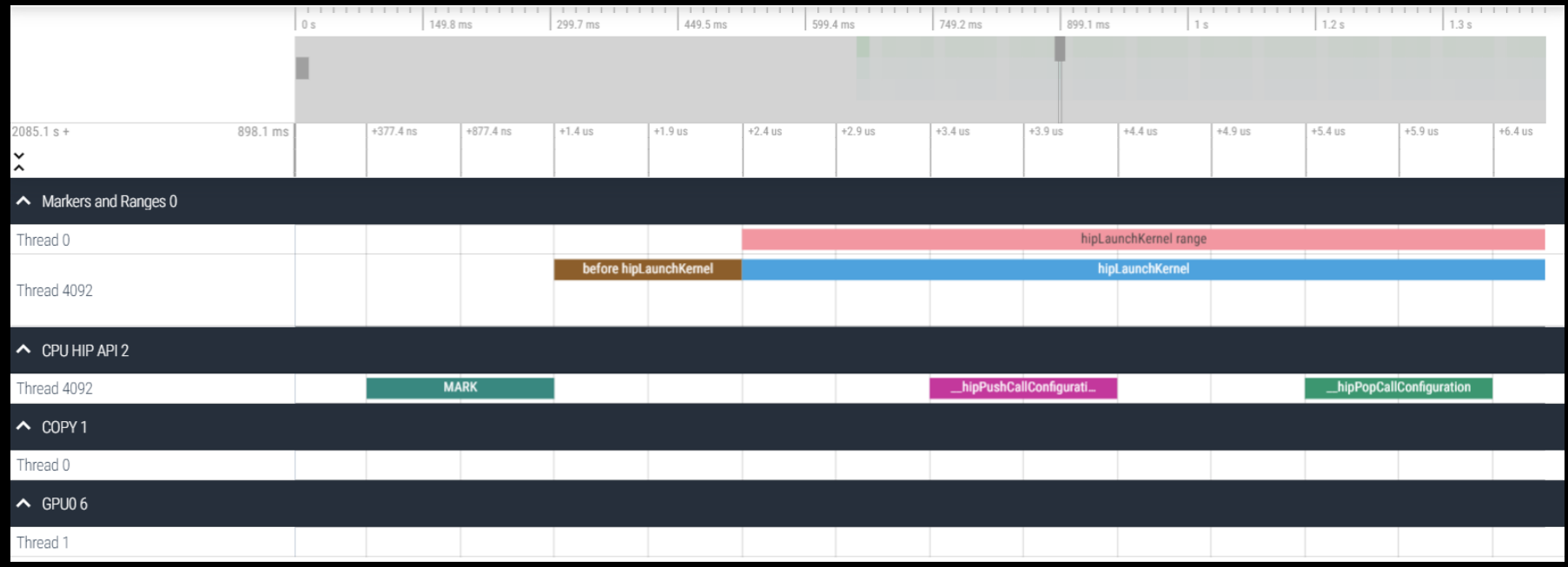
rocprof: Collecting application traces with markers

- Rocprof can collect user code-markers using rocTX
  - See MatrixTranspose.cpp example on roctracer GitHub page for sample in-code usage
  - **`$ /opt/rocm/bin/rocprof --hip-trace --roctx-trace <app with arguments>`**

```
roctracer_mark("before HIP
LaunchKernel");

roctxMark("before hipLaunchKernel");
int rangeId =
roctxRangeStart("hipLaunchKernel
range");

roctxRangePush("hipLaunchKernel");
hipLaunchKernelGGL(matrixTranspose,…)
;
roctracer_mark("after HIP
LaunchKernel");
 roctxMark("after hipLaunchKernel");
```



rocprof: Collecting hardware counters

- rocprofiler can collect a number of hardware counters and derived counters
  - **$ /opt/rocm/bin/rocprof --list-basic**
  - **$ /opt/rocm/bin/rocprof --list-derived**
- Specify counters in a counter file. For example:
  - **$ /opt/rocm/bin/rocprof -i rocprof_counters.txt <app with args>**
  - **$ cat rocprof_counters.txt**

```
pmc : Wavefronts VALUInsts VFetchInsts VWriteInsts VALUUtilization VALUBusy WriteSize
pmc : SALUInsts SFetchInsts LDSInsts FlatLDSInsts GDSInsts SALUBusy FetchSize
pmc : L2CacheHit MemUnitBusy MemUnitStalled WriteUnitStalled ALUStalledByLDS LDSBankConflict
...
```

  - 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
  - A csv file will be created by this command containing all of the requested counters

rocprof: Commonly Used 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
- 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

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

Performance counters tips and tricks

- GPU Hardware counters are global
  - Kernel dispatches are serialized to ensure that only one dispatch is ever in flight
  - It is recommended that no other applications are running that use the GPU when collecting performance counters.
- Use **“`--basenames on`”** which will report only kernel names, leaving off kernel arguments.
- How do you time a kernel’s duration?
  - **`$ /opt/rocm/bin/rocprof --timestamp on -i rocprof_counters.txt <app with args>`**
  - This produces four times: DispatchNs, BeginNs, EndNs, and CompleteNs
  - Closest thing to a kernel duration: EndNs - BeginNs
  - If you run with **“`--stats`”** the resultant results file will automatically include a column that calculates kernel duration
    - Note: the duration is aggregated over repeated calls to the same kernel

rocprof: Multiple MPI Ranks

- rocprof can collect counters and traces for multiple MPI ranks
- Say you want to profile an application usually called like this:
  - `mpiexec –np <n> ./Jacobi_hip –g <x> <y>`
  - Then invoke the profiler by executing:

```
mpiexec -np <n> rocprof --hip-trace ./Jacobi_hip -g <x> <y>
or
srun --ntasks=n rocprof --hip-trace ./Jacobi_hip -g <x> <y>
```

- This will produce a single CSV file per MPI process
- Multi-node profiling currently isn’t supported

Profiling Per MPI Rank: From Another Node(1)

- Let’s consider a 3-step run:
  - sbatch_profiling.sh with sbatch command line to launch the app
  - rocprof_batch.slurm This file contains sbatch parameters and the call to srun command line
  - rocprof_wrapper.sh calls rocprof command line with input parameters to run the application to be profiled

- $cat sbatch_profiling.sh
  - sbatch -p <partition> -w <node> rocprof_batch.slurm

- $cat rocprof_batch.slurm

```
#!/bin/bash
#SBATCH --job-name=run
#SBATCH --ntasks=2
#SBATCH --ntasks-per-node=2
#SBATCH --gpus-per-task=1
#SBATCH --cpus-per-task=1
#SBATCH --distribution=block:block
#SBATCH --time=00:20:00
#SBATCH --output=out.txt
#SBATCH --error=err.txt
#SBATCH -A XXXXX
cd ${SLURM_SUBMIT_DIR}
```

- **load necessary modules**
- **export necessary environment variables**

```
 make clean all
srun ./rocprof_wrapper.sh ${repository} triad_off_mpi triad_off_mpi
```

Profiling Per MPI Rank: From Another Node(2)

- $cat rocprof_wrapper.sh

```
#!/bin/bash
set -euo pipefail
# depends on ROCM_PATH being set outside;  input arguments are the output directory & the name
outdir="$1"
name="$2"
if [[ -n ${OMPI_COMM_WORLD_RANK+z} ]]; then
  # mpich
  export MPI_RANK=${OMPI_COMM_WORLD_RANK}
elif [[ -n ${MV2_COMM_WORLD_RANK+z} ]]; then
  # ompi
  export MPI_RANK=${MV2_COMM_WORLD_RANK}
elif [[ -n ${SLURM_PROCID+z} ]]; then
  export MPI_RANK=${SLURM_PROCID}
else
  echo "Unknown MPI layer detected! Must use OpenMPI, MVAPICH, or SLURM"
  exit 1
fi
rocprof="${ROCM_PATH}/bin/rocprof"

pid="$$"
outdir="${outdir}/rank_${pid}_${MPI_RANK}"
outfile="${name}_${pid}_${MPI_RANK}.csv"
${rocprof} -d ${outdir} --hsa-trace -o ${outdir}/${outfile} "${@:3}"
```

rocprof: Profiling Overhead

- As with every profiling tool that collects data, there is an overhead
- The percentage of the overhead depends on many aspects, for example if you try to instrument tiny tasks in a loop, this can take more time than tasks outside a loop
- If you try to collect many counters and especially ones that need more than one pass, then this could cause overhead if there a lot of related calls
- Also, if a lot of markers are added and especially in a loop then the roctx-trace can take significantly more time than the non instrumented execution time
- In general, more the data you collect, more the overhead can be, and it depends on the application.

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.

Questions?


AMD