ROCm™ Library Support & Profiling Tools

Julio Maia, Noel Chalmers, Paul T. Bauman, Nicholas Curtis, Nicholas Malaya, Damon McDougall, Rene van Oostrum, Noah Wolfe

May 2021

AMD

Agenda

- Library Support
- Profiling tools

ROCm GPU Libraries

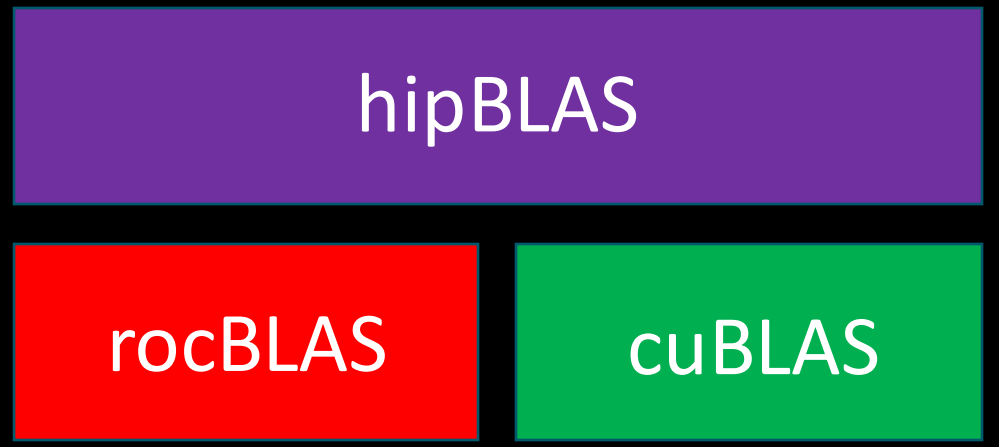
ROCm provides several GPU math libraries

- Typically two versions:
  - roc* -> AMD GPU library, usually written in HIP
  - hip* -> Thin interface between roc* and Nvidia cu* library

When developing an application meant to target both CUDA and AMD devices, use the hip* libraries

When developing an application meant to target only AMD devices, may prefer the roc* library API.

- Some roc* libraries perform better by using addition APIs not available in the cu* equivalents.



AMD Math Library Equivalents: “Decoder Ring”

CUBLAS	**ROCBLAS**	Basic Linear Algebra Subroutines
CUFFT	**ROCFFT**	Fast Fourier Transforms
CURAND	**ROCRAND**	Random Number Generation
THRUST	**ROCTHRUST**	C++ Parallel Algorithms
CUB	**ROCPRIM**	Optimized Parallel Primitives

AMD Math Library Equivalents: “Decoder Ring”

CUSPARSE	**ROCSPARSE**	Sparse BLAS, SpMV, etc.
CUSOLVER	**ROCSOLVER**	Linear Solvers
AMGX	**ROCALUTION**	Solvers and preconditioners for sparse linear systems

GITHUB.COM/ROCM-DEVELOPER-TOOLS/HIP → HIP_PORTING_GUIDE.MD FOR A COMPLETE LIST

Some Links to Key Libraries

- BLAS
  - rocBLAS (https://github.com/ROCmSoftwarePlatform/rocBLAS)
  - hipBLAS (https://github.com/ROCmSoftwarePlatform/hipBLAS)
- FFTs
  - rocFFT (https://github.com/ROCmSoftwarePlatform/rocFFT)
- Random number generation
  - rocRAND (https://github.com/ROCmSoftwarePlatform/rocRAND)
- Sparse linear algebra
  - rocSPARSE (https://github.com/ROCmSoftwarePlatform/rocSPARSE)
  - hipSPARSE (https://github.com/ROCmSoftwarePlatform/hipSPARSE)
- Iterative solvers
  - rocALUTION (https://github.com/ROCmSoftwarePlatform/rocALUTION)
- Parallel primitives
  - rocPRIM (https://github.com/ROCmSoftwarePlatform/rocPRIM)
  - hipCUB (https://github.com/ROCmSoftwarePlatform/hipCUB)

AMD Machine Learning Library Support

Machine Learning Frameworks:

- Tensorflow: https://github.com/ROCmSoftwarePlatform/tensorflow-upstream
- Pytorch: https://github.com/ROCmSoftwarePlatform/pytorch
- Caffe: https://github.com/ROCmSoftwarePlatform/hipCaffe

Machine Learning Libraries:

- MIOpen (similar to cuDNN): https://github.com/ROCmSoftwarePlatform/MIOpen
- Tensile (GEMM Autotuner): https://github.com/ROCmSoftwarePlatform/Tensile
- RCCL (ROCm analogue of NCCL): https://github.com/ROCmSoftwarePlatform/rccl
- Horovod (Distributed ML): https://github.com/ROCmSoftwarePlatform/horovod

Benchmarks:

- DeepBench: https://github.com/ROCmSoftwarePlatform/DeepBench
- MLPerf: https://mlperf.org

Profiling

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 chrome tracing

rocprof: Getting started + useful flags

- To get help:
  - `$ /opt/rocm/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
- Advanced usage
  - -m <metric file>: Allows the user to define and collect custom metrics. See rocprofiler/test/tool/*.xml on GitHub for examples.

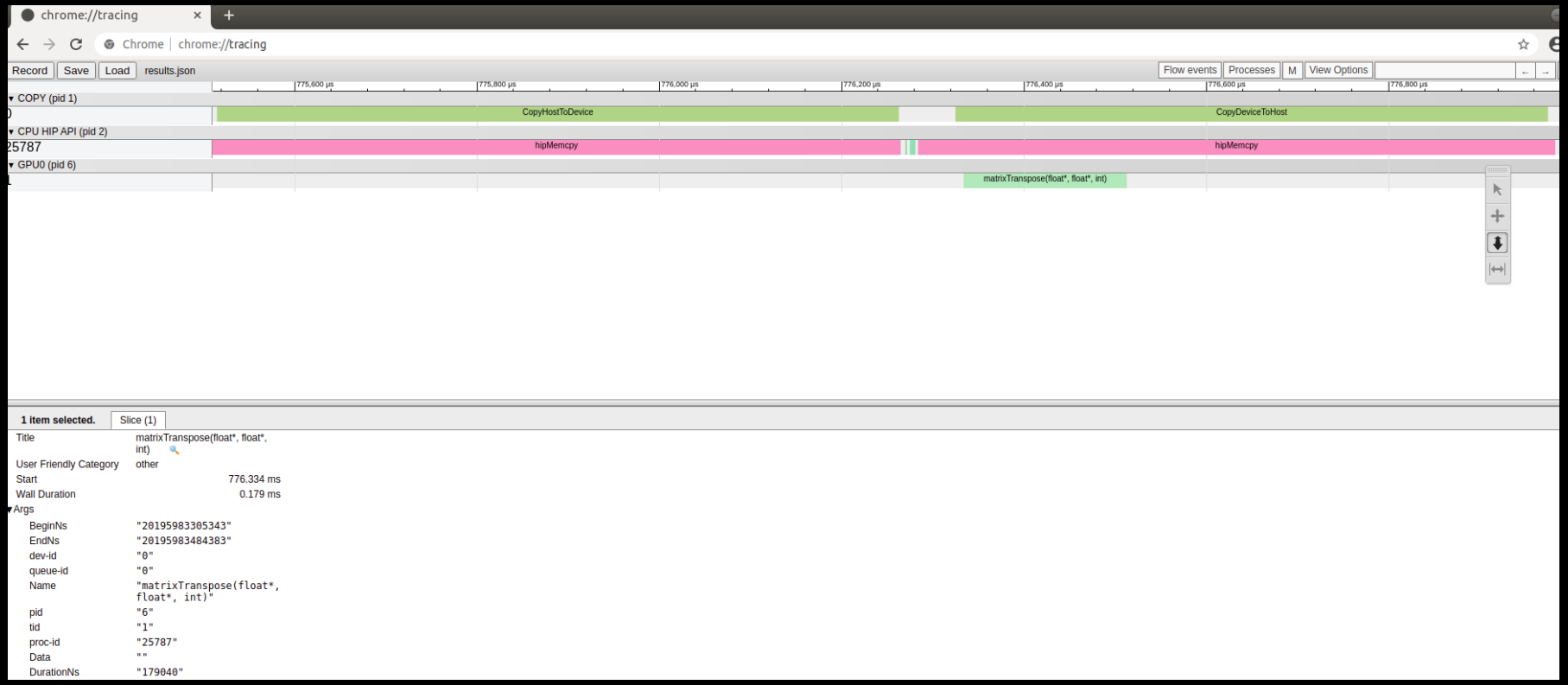
rocprof: Collecting application traces

- rocprof can collect a variety of trace event types, and generate timelines in JSON format for use with chrome-tracing, 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

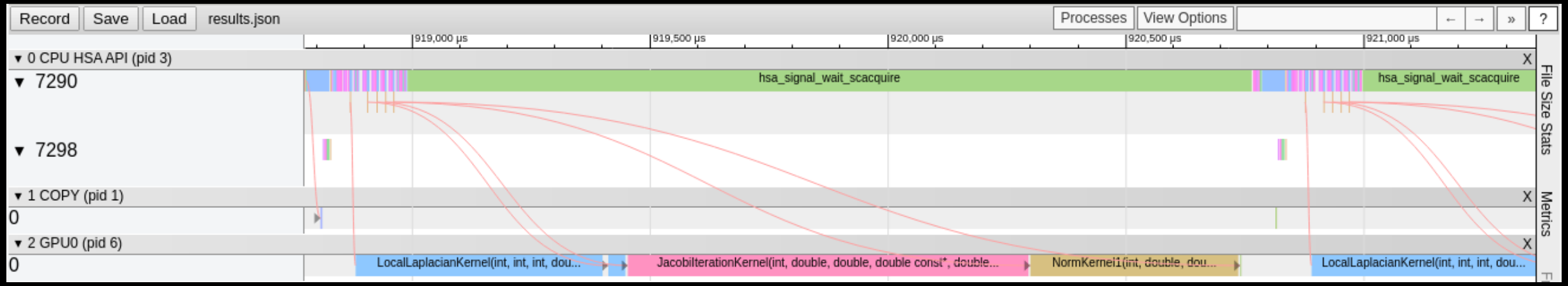
rocprof: Collecting application traces

- rocprofiler 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
  - Go to chrome://tracing and then load in the .json file.
    - The trace will display HIP calls, mem copies, kernels.



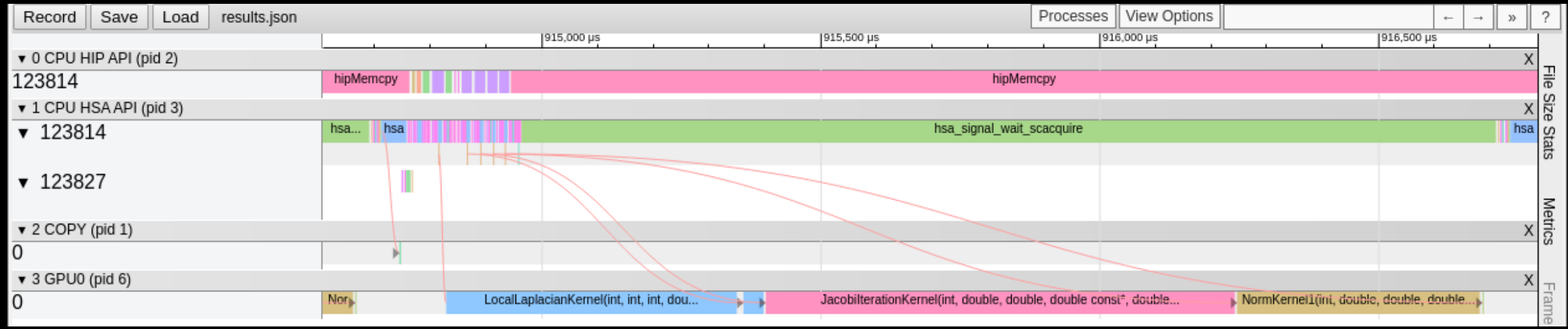
rocprof: Collecting application traces

- rocprofiler can collect traces
  - `$ /opt/rocm/bin/rocprof --hsa-trace <app with arguments>`
  - This will output a .json file that can be visualized using the chrome browser
  - Go to chrome://tracing and then load in the .json file.
    - The trace will display copies, hsa signals, and kernel calls.
    - Slow your calculation a lot! Use with caution



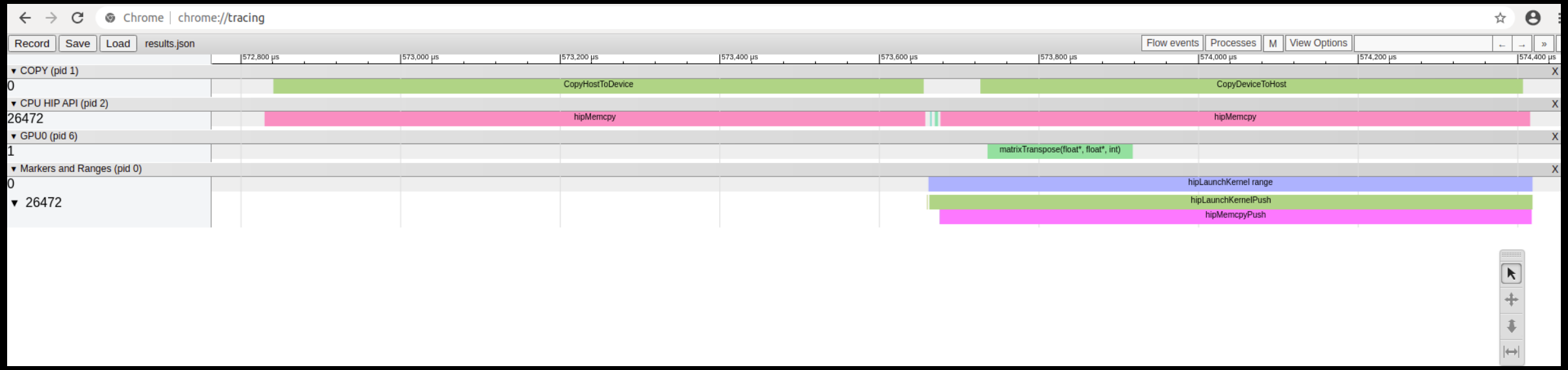
rocprof: Collecting application traces

- rocprofiler can collect multiple trace modes simultaneously
  - **`$ /opt/rocm/bin/rocprof --hsa-trace --hip-trace <app with arguments>`**
  - This command will additionally add HIP API calls to the trace



rocprof: Collecting application traces

- 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>`**



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 --timestamps 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

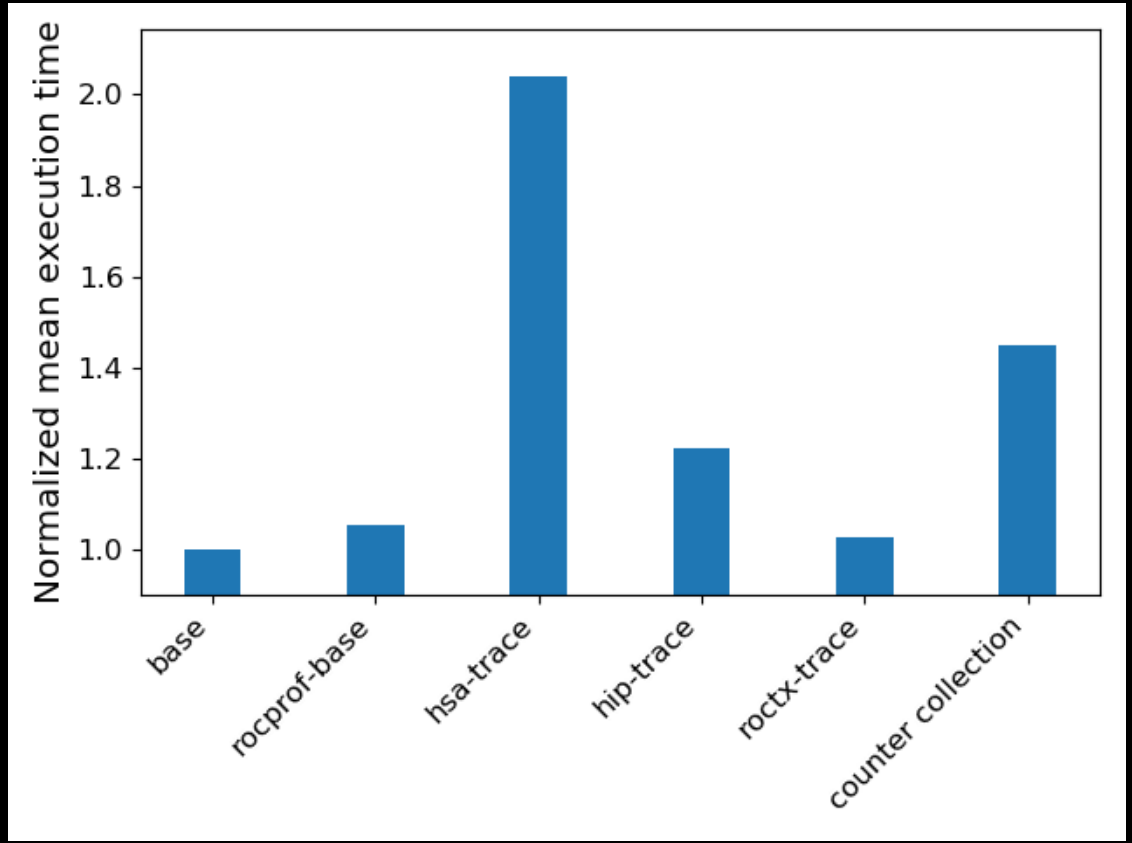
rocprof: Multiple MPI Ranks

- 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:

    `rocprof --hip-trace mpiexec -np <n> ./Jacobi_hip -g <x> <y>`
- This will produce a single unified CSV file for all ranks
- Multi-node profiling currently isn’t supported

rocprof: Profiling Overhead

Simple estimation of profiling overhead, obtained via wall-clock timing of entire application run via linux ‘time’ utility:



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.


AMD