HIP Lecture Series

GPU Profiling – Performance Timelines

For the HIP Lecture Series, the examples can be retrieved from this repository.

```
git clone https://github.com/olcf/hip-training-series
```

The exercises for this lecture are in the Lecture 4 directory. I have only had a quick test, so expect a few rough places.

We are working with the portable build system from earlier exercises, so the tests on Perlmutter are to run the Nvidia tools to get a profile there. An alternative that would be good to try also is a portable HPC Community timeline profiling tool. Comparing the results from these tools and between GPUs on a moderately complex problem would be worthy of a paper or report.

We focus on the AMD tools primarily because we don't want to speak for our esteemed colleagues at Nvidia. They have the expertise on their tools.

This markdown document is located at `'Lecture4/04 Performance_Timelines_Exercises.md'` contains the instructions to run the examples. You can view it in github for better readability or download the pdf file `'Lecture4/04 Performance_Timelines_Exercises.pdf'` which has been generated from the markdown document.

As is usual in this lecture series, the linked Google doc for comments, questions and answers will be at https://docs.google.com/document/d/1aUfzofSgxCn-gkejJHDlh54YX5mRMaj5xkAO7zEZUkQ/edit

The reservation id for the schedule session is hip_training_2023_10_02.

Rocprof

In these examples, we'll first just try a simple example to make sure everything is working. Then we'll add a more complex example with MPI.

Simple HIP code

Get an allocation. Use your project id in the project field. If you do not remember it, run the command without the -A option and it should report your valid projects.

```
salloc -N 1 -n 2 -p batch --reservation=hip_training_2023_10_02 --gpus=2 -t 15:00 -A <project>
```

Get exercises

```
git clone https://github.com/olcf/hip-training-series
```

Go to first example

```
cd hip-training-series/Lecture4/saxpy
```

Load environment

```
module load PrgEnv-amd
module load amd
module load cmake
export CXX=${ROCM_PATH}/llvm/bin/clang++
```

Build saxpy application

```
make
```

Run application to make sure it is workig

```
./saxpy
```

It should report "PASSED!"

Run with rocprof

```
rocprof --stats ./saxpy
```

Check output from the stats report

```
cat results.stats.csv
```

You should see a report like:

```
"Name","Calls","TotalDurationNs","AverageNs","Percentage"
"saxpy(int, float const*, int, float*, int) [clone .kd]",1,4800,4800,100.0
```

But we want to focus on the timeline profile for these exercises.

```
rocprof --hip-trace --hsa-trace ./saxpy
```

Copy back to local system. From your local system:

```
scp [username@]frontier.olcf.ornl.gov:hip-training-series/Lecture4/saxpy/results.json results.json
```

Start up the chrome browser. Put ui.perfetto.com in the location. Then open the trace file

You should see a very simple timeline presentation in the browser. Try moving around with the WASD keys. Now we are ready to move to a more complex example.

A more complex code with MPI

We'll run a multi-process code to see a more complex timeline

Setup environment

```
salloc -N 1 -n 2 -p batch --reservation=hip_training_2023_10_02 --gpus=2 -t 15:00 -A <project>
module load PrgEnv-amd
module load amd
module load cmake
export CXX=${ROCM_PATH}/llvm/bin/clang++
```

Download examples repo and navigate to the `HIPIFY` exercises

```
cd hip-training-series/Lecture4/jacobi
```

Compile all

```
mkdir build && cd build
cmake ..
make
```

Currently seeing error finding roctx64 and roctracer64

First run executable to check if running correctly

```
srun -n 2 ./Jacobi_hip -g 2
```

Run `rocprof` on jacobi to obtain trace files

```
srun -n 2 rocprof_wrapper.sh output ./Jacobi_hip -g 2
```

But we need a rocprof_wrapper.sh for creating separate files for the output. Here is one for some versions of MPI. We'll cover how to modify it for other systems and MPI versions.

```
#!/bin/bash
set -euo pipefail
outdir=$1
name=$2
outdir="${outdir}_${OMPI_COMM_WORLD_RANK}"
outfile="${name}_${OMPI_COMM_WORLD_RANK}.csv"
rocprof -d ${outdir} --hsa-trace --hip-trace -o ${outdir}/${outfile} ./Jacobi_hip $3 $4
```

The key here is to have a different output file name and/or directory to feed into the rocprof command. For a different MPI, run `mpirun -n 2 printenv` and see what environment variable has the rank number in it. Substitute it for the `${OMPI_COMM_WORLD_RANK}` variable above.

```
srun -n 2 printenv
```

Another option is to encode the process id into the strings. This approach is shown in the wrapper.sh script.

```
pid=$$
outdir="${outdir}_${pid}"
outfile="${name}_${pid}.csv"
```

To run with the PIDs:

```
srun -n 2 wrapper.sh output ./Jacobi_hip -g 2
```

Check Results

```
cat results.csv
```

Check the statistics result file, one line per kernel, sorted in descending order of durations

```
cat results.stats.csv
```

Using `--basenames on` will show only kernel names without their parameters.

```
rocprof --stats --basenames on nbody-orig 65536
```

Check the statistics result file, one line per kernel, sorted in descending order of durations

```
cat results.stats.csv
```

Trace HIP calls with `--hip-trace`

```
srun rocprof --stats --hip-trace nbody-orig 65536
```

Check the new file `results.hip_stats.csv`

```
cat results.hip_stats.csv
```

Profile also the HSA API with the `--hsa-trace`

```
srun rocprof --stats --hip-trace --hsa-trace nbody-orig 65536
```

Check the new file `results.hsa_stats.csv`

```
cat results.hsa_stats.csv
```

On your laptop, download `results.json`

```
scp <username>@frontier.olcf.ornl.gov/hip-training-series/Lecture4/jacobi/results.json ./
```

Open a browser and go to https://ui.perfetto.dev/. Click on `Open trace file` in the top left corner. Navigate to the `results.json` you just downloaded. Use WASD to navigate the GUI

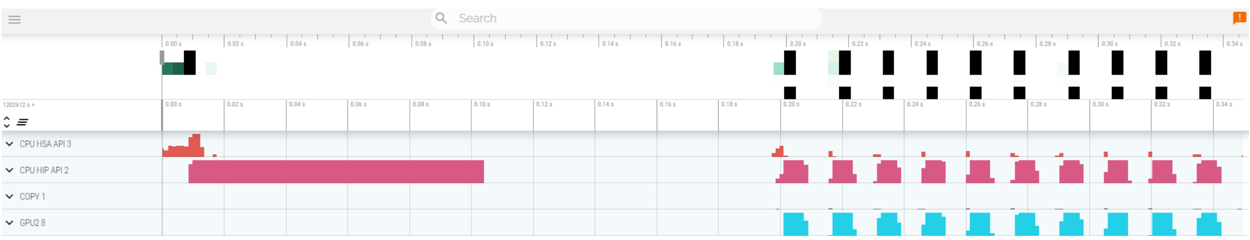

Figure 1: image

Read about hardware counters available for the GPU on this system (look for gfx90a section)

```
less $ROCM_PATH/lib/rocprofiler/gfx_metrics.xml
```

Create a `rocprof_counters.txt` file with the counters you would like to collect

```
vi rocprof_counters.txt
```

Content for `rocprof_counters.txt`:

```
pmc : Wavefronts VALUInsts
pmc : SALUInsts SFetchInsts GDSInsts
pmc : MemUnitBusy ALUStalledByLDS
```

Execute with the counters we just added:

```
srun rocprof --timestamp on -i rocprof_counters.txt  nbody-orig 65536
```

You'll notice that `rocprof` runs 3 passes, one for each set of counters we have in that file.

Contents of `rocprof_counters.csv`

```
cat rocprof_counters.csv
```

```
exit
```

Omnitrace

Setup environment

```
module load rocm openmpi
```

Basic Omnitrace setup

List the various options and environment settings available for the `omnitrace` category:

```
omnitrace-avail --categories omnitrace
```

To add brief descriptions, use `-bd` option

```
omnitrace-avail -bd --categories omnitrace
```

Create an Omnitrace configuration file with description per option.

```
omnitrace-avail -G ~/omnitrace_all.cfg --all
```

To create a configuration file without descriptions, drop the `--all` option:

```
omnitrace-avail -G ~/omnitrace.cfg
```

Declare to use this configuration file:

```
export OMNITRACE_CONFIG_FILE=~/omnitrace.cfg
```

Setup Jacobi Example

Go to the jacobi code in the examples repo:

```
cd hip-lecture-series/Lecture4/jacobi
```

Compile

```
make
```

Execute the binary to make sure it runs successfully: - Note: To get rid of `Read -1, expected 4136, errno = 1` add `--mca pml ucx --mca pml_ucx_tls ib,sm,tcp,self,cuda,rocm` to the `mpirun` command line

```
srun -np 1 ./Jacobi_hip -g 1 1
```

Dynamic Instrumentation

(WARNING) - this may in the current container Run the code with omnitrace to get runtime instrumentation. Time it to see overhead of dyninst loading all libraries in the beginning.

```
srun -np 1 omnitrace-instrument -- ./Jacobi_hip -g 1 1
```

Check available functions to instrument using the `--print-available functions` option. Note, the `--simulate` option will not execute the binary.

```
srun -np 1 omnitrace-instrument -v 1 --simulate --print-available functions -- ./Jacobi_hip -g 1 1
```

Binary Rewrite

Create instrumented binary

```
omnitrace-instrument -o ./Jacobi_hip.inst -- ./Jacobi_hip
```

Executing the new instrumented binary, time it to see lower overhead:

```
srun -np 1 omnitrace-run -- ./Jacobi_hip.inst -g 1 1
```

See the list of the instrumented GPU calls:

```
cat omnitrace-Jacobi_hip.inst-output/<TIMESTAMP>/roctracer-0.txt
```

Visualization

Copy the `perfetto-trace-0.proto` to your laptop, open the web page https://ui.perfetto.dev/

```
scp <username>@frontier.olcf.ornl.gov/hip-training-series/Lecture4/jacobi/omnitrace-Jacobi_hip.inst-outp
```

Click `Open trace file` and select the `.proto` file

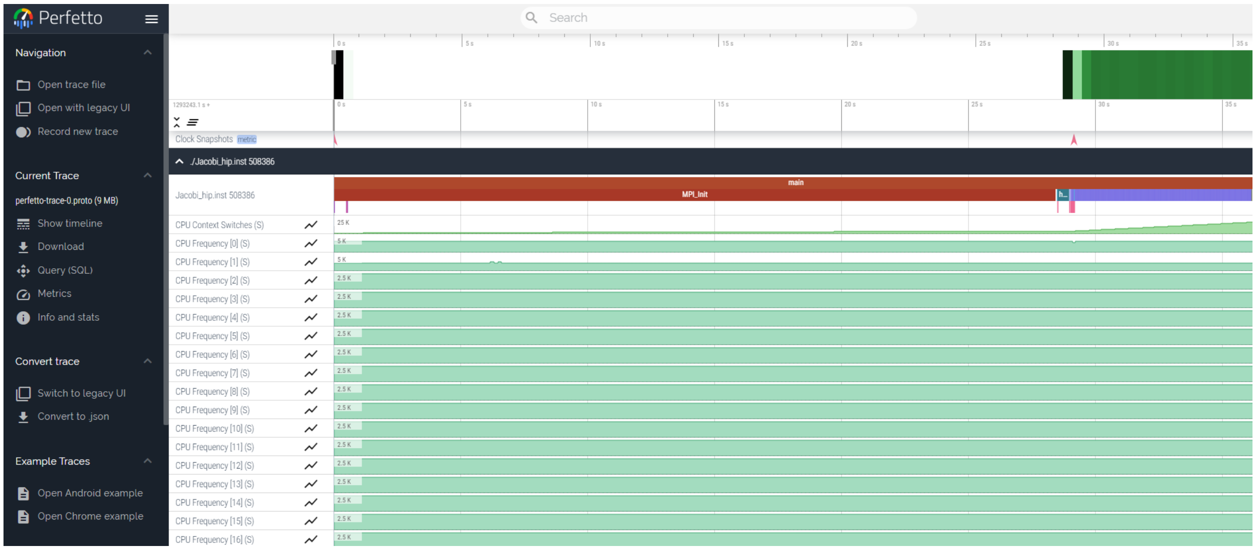

Figure 2: image

Hardware Counters

See a list of all the counters

```
omnitrace-avail --all
```

Declare in your configuration file:

```
OMNITRACE_ROCM_EVENTS = GPUBusy,Wavefronts,MemUnitBusy
```

Check again:

```
grep OMNITRACE_ROCM_EVENTS $OMNITRACE_CONFIG_FILE
```

Run the instrumented binary, and visualize the Perfetto trace produced to see the hardware counters:

```
srun -np 1 omnitrace-run -- ./Jacobi_hip.inst -g 1 1
```

Profiling Multiple Ranks

Run the instrumented binary with multiple ranks. You'll find multiple `perfetto-trace-*.proto` files, one for each rank.

```
srun -np 2 omnitrace-run -- ./Jacobi_hip.inst -g 2 1
```

You can visualize them separately in Perfetto, or combine them using cat and visualize them in the same Perfetto window.

```
cat perfetto-trace-0.proto perfetto-trace-1.proto > allprocesses.proto
```

Sampling

Set the following in your configuration file:

```
OMNITRACE_USE_SAMPLING = true
OMNITRACE_SAMPLING_FREQ = 100
```

Execute the instrumented binary and visualize the perfetto trace.

```
mpirun -np 1 omnitrace-run -- ./Jacobi_hip.inst -g 1 1
```

Scroll down to the very bottom to see the sampling output. Those traces will be annotated with a (S) as well.

Kernel Timings

Open the `wall_clock-0.txt` file:

```
cat omnitrace-Jacobi_hip.inst-output/<TIMESTAMP>/wall_clock-0.txt
```

In order to see the kernel durations aggregated in your configuration file, make sure to set in your config file or in the environment:

```
OMNITRACE_USE_TIMEMORY = true
OMNITRACE_FLAT_PROFILE = true
```

Execute the code and check the `wall_clock-0.txt` file again.

```
OMNITRACE_USE_TIMEMORY=true OMNITRACE_FLAT_PROFILE=true mpirun -np 1 omnitrace-run -- ./Jacobi_hip.inst
```