Tracing and Profiling of GPU-Accelerated Software

Progress Report Meeting
May 5, 2017

Paul Margheritta       Michel Dagenais

DORSAL lab
École Polytechnique de Montréal
Introduction

• **GPU**: for graphics and general purpose (GPGPU)

• So many **cores**! We lack tools for that.

• Example: Radeon R9 Nano from AMD, **4096** cores

• We need to address issues related to **GPU specifics** and **highly parallel systems**
Software context

- **ROCm** (Radeon Open Compute): open-source platform for GPU development
- **HSA** (Heterogeneous System Architecture): runtime and API used to launch compute kernels
- **CodeXL**: open-source debugging and performance analysis tool for HSA and OpenCL
Research goals

- Analyze current **tracing and profiling mechanisms**
- Explore **AMD initiatives** for performance analysis on GPUs
- Provide **tracing** with LTTng in the HSA runtime
- Design **views** in Trace Compass for better understanding
Common techniques

- **Intercepting** and replacing symbols in the HSA runtime
- Early solution: changing links in the **API function table**
- More flexibility with **preloaded libraries**: build a collection of libraries that intercept API calls and other functions and preload them with **LD_PRELOAD**
Call stack events

- All **API functions** instrumented at entry and exit

- Generation of interception sources **automated** with Python scripts

```
[14:09.46.9998478591 (+0.0000000212) paul-gpu hsa_runtime:function_exit: (cpu_id = 0), (vtid = 9639), (name = "hsa_agent_iterate_regions")
[14:09.46.999848185 (+0.0000000246) paul-gpu hsa_runtime:function_entry: (cpu_id = 0), (vtid = 9639), (name = "hsa_memory_allocate")
[14:09.46.999873645 (+0.000025540) paul-gpu hsa_runtime:function_exit: (cpu_id = 0), (vtid = 9639), (name = "hsa_memory_allocate")
[14:09.46.999876281 (+0.000025556) paul-gpu hsa_runtime:function_entry: (cpu_id = 0), (vtid = 9639), (name = "hsa_queue_load_write_index_relaxed")
[14:09.46.999879499 (+0.000032298) paul-gpu hsa_runtime:function_entry: (cpu_id = 0), (vtid = 9639), (name = "hsa_queue_load_write_index_relaxed")
[14:09.46.999879565 (+0.000032363) paul-gpu hsa_runtime:function_exit: (cpu_id = 0), (vtid = 9639), (name = "hsa_queue_load_write_index_relaxed")
[14:09.46.999879172 (+0.000006634) paul-gpu hsa_runtime:function_exit: (cpu_id = 0), (vtid = 9639), (name = "hsa_queue_store_write_index_relaxed")
```

Progress Report Meeting – May 2017 – Paul Margheritta, Michel Dagenais
Queue profiling events

- **User-mode queues** are used to dispatch functions to be executed on the GPU
- Gives the **state** of the user-mode queues
- Information about the **AQL packets** sent to the queues is also available
Kernels timing

- A **profiled queue** allows us to get timing information

- **Start/end timestamps** are aligned on the monotonic clock of the system

- This information is obtained in an **asynchronous** way
Performance counters

- Performance counters provide **low-level**, hardware-related data

- The **SoftCP mode** is used to define pre- and post-dispatch callbacks

- Those callbacks open and close **contexts** useful for the collection of performance counters

- In the **multi-threaded case**, we need a lock on the opening of a context
Linux kernel events

- Some trace points are already defined in the **AMD Linux kernel drivers**

- Some other trace points may be added

- Comes in addition with **user space tracing** for more information
Post-tracing processing

- The 4 user-space tracing targets are mutually **incompatible**

- Traces will have to be collected separately, in multiple runs, and then merged or reduced

- **call_stack** (no prerequisite)
- **queue_profiling** (requires a profiled queue with no kernel timing)
- **kernel_times** (requires a profiled queue with kernel timing)
- **perf_counters** (requires specific contexts to be open)
Combining data from multiple runs

- Early solution: using **Babeltrace** Python bindings

- **Trace Compass experiments** allow merging and offsetting of traces

- Mechanisms for sorting and merging have been proposed for **Chromium traces** and could be re-used
Future work

• Adapt to current work to **OpenCL** applications

• Find more **generic solutions** for trace merging

• Provide more advanced **Linux kernel** tracing
Thank you!
Any questions?
paul.margheritta@polymtl.ca