GPU Tracing and Profiling

Progress Report Meeting
December 12, 2016

Paul Margheritta Michel Dagenais

DORSAL lab
École Polytechnique de Montréal

Hardware context

- **AMD Radeon R9 Nano** graphics card
- **Graphics Core Next** architecture
- **4096** stream processors = **4096** cores
- **4 GB** video memory
- Released in **October 2015**

Research goals

- Understanding current **tracing and profiling mechanisms** on GPUs
- Adapting mechanisms to our tools: **LTTng**, **Trace Compass**...
- Developing **new tools** for **performance analysis** on GPUs and heterogeneous systems

Software context

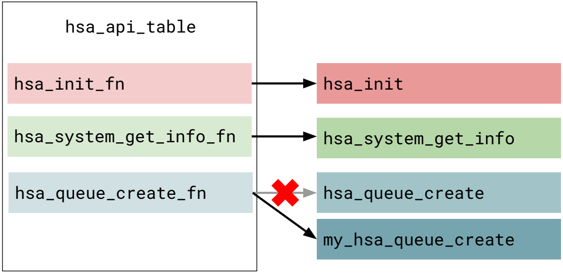
GPUOpen

RO
Cm
ON

CODE XL

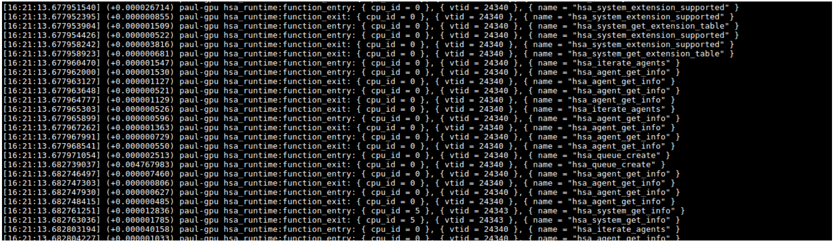
- **ROCm** (Radeon Open Compute): open-source platform for GPU development
- **HSA** (Heterogeneous System Architecture): runtime and API used to launch compute kernels
- **CodeXL**: open-source debugging and performance analysis tool for HSA and OpenCL

Intercepting API calls



- Examples of **API functions**: `hsa_init`, `hsa_system_get_info`, `hsa_queue_create`...
- Function pointers are stored in a **table**
- **Intercepting** an API call: changing the function pointer in the table

Automating interception

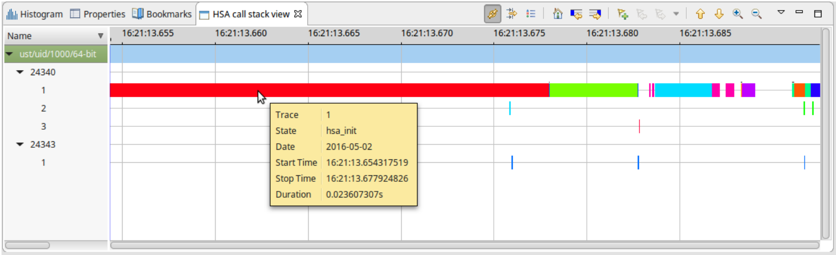


```
[16:21:13.677951540] (+0.000026714) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_system_extension_supported" }
[16:21:13.677952395] (+0.000000855) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_system_extension_supported" }
[16:21:13.677953904] (+0.000001509) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_system_get_extension_table" }
[16:21:13.677954426] (+0.000000522) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_system_extension_supported" }
[16:21:13.677958242] (+0.000003816) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_system_extension_supported" }
[16:21:13.677958923] (+0.000000681) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_system_get_extension_table" }
[16:21:13.677960470] (+0.000001547) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_iterate_agents" }
[16:21:13.677962000] (+0.000001530) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.677963127] (+0.000001127) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.677963648] (+0.000000521) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.677964777] (+0.000001129) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.677965303] (+0.000000526) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_iterate_agents" }
[16:21:13.677965899] (+0.000000596) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.677967262] (+0.000001363) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.677967991] (+0.000000729) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.677968541] (+0.000000550) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.677971054] (+0.000002513) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_queue_create" }
[16:21:13.682739037] (+0.004767983) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_queue_create" }
[16:21:13.682746497] (+0.000007460) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.682747303] (+0.000000806) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.682747930] (+0.000000627) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.682748415] (+0.000000485) paul-gpu hsa_runtime:function_exit: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
[16:21:13.682761251] (+0.000012836) paul-gpu hsa_runtime:function_entry: { cpu_id = 5 }, { vtid = 24343 }, { name = "hsa_system_get_info" }
[16:21:13.682763036] (+0.000001785) paul-gpu hsa_runtime:function_exit: { cpu_id = 5 }, { vtid = 24343 }, { name = "hsa_system_get_info" }
[16:21:13.682803194] (+0.000040158) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_iterate_agents" }
[16:21:13.682804227] (+0.000001033) paul-gpu hsa_runtime:function_entry: { cpu_id = 0 }, { vtid = 24340 }, { name = "hsa_agent_get_info" }
```

- Typical interception case: **instrumenting entries and exits** for API functions

- **Easy generation** of header and sources for the interception

An API call stack with LTTng + Trace Compass



Trace	1
State	hsa_init
Date	2016-05-02
Start Time	16:21:13.654317519
Stop Time	16:21:13.677924826
Duration	0.023607307s

- The **XML analysis** feature of Trace Compass is used to build a **call stack view**
- Function names are pushed and popped on a **stack** in the state system

Launching a compute kernel on the GPU

1. Creating a **queue**
2. Obtaining the current **write index**
3. Writing an AQL **kernel dispatch packet**
4. Ringing the **doorbell** to launch the kernel

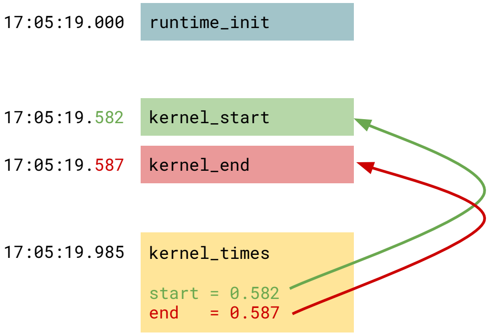
HSA FOUNDATION

---

1. Creating a **queue**
2. Creating a **kernel object**
3. **Enqueuing** the kernel in the queue

OpenCL

Timing kernels



17:05:19.000 runtime_init

17:05:19.582 kernel_start

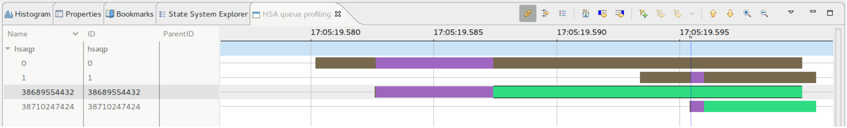
17:05:19.587 kernel_end

17:05:19.985 kernel_times

start = 0.582
end = 0.587

- Goal: including **kernel start/end times** as events in the trace
- A **profiled queue** can be created to gather timing information about kernels
- The kernel start/end times are **synchronized with the initialization** using the monotonic clock
- The new events are included in the initial trace using the **Python Babeltrace bindings**

Visualizing the status of kernels



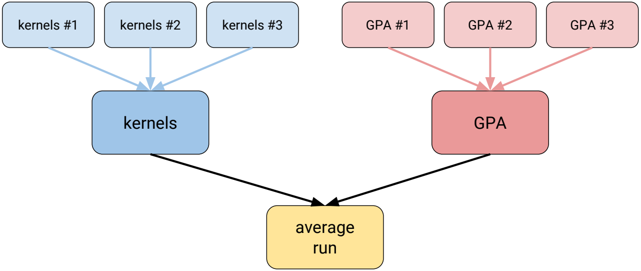
Name	ID	ParentID
hsaqp	hsaqp	
0	0	
1	1	
38689554432	38689554432	
38710247424	38710247424	

- Two states for **queues**: `WAITING` and `RUNNING`
- Three states for **kernels**: `WAITING`, `RUNNING` and `DONE`
- Reflecting the HSA structure in the **state system**: agent → queue → kernel

Sampling performance counters

- Low-level, hardware-related data can be obtained with **GPUPerfAPI** (GPA)
- Few performance counters available in **HSA**: `Wavefronts`, `CacheHit`...
- Opening a **GPA context**: easy with API interception on the queue creation and destruction
- Opening a **GPA sample**: intercepting the kernel dispatch is harder in HSA

Combining data from multiple runs



- Goal: having **kernel timing** and **performance counters** data at the same time
- Problem: it requires **two types of queues**
- Solution: running the program **multiple times** with the two types of queues and **merging** the traces

Future work

- Working on **bigger applications**
- Gathering **lower-level data** about GPU activity
- Tracing the ROCm **Linux kernel** driver
- Analyzing other types of **GPU traces** (JSON...)

Thank you!

Any questions?

paul.margheritta@polymtl.ca