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

- **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

- 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

- 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

1. Creating a queue
2. Creating a kernel object
3. Enqueueing the kernel in the queue
**Timing kernels**

- **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

- 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