GPU Compute Kernel Instrumentation and Performance Analysis for HPC

Low-overhead Trace Collection on GPU

Sébastien Darche <sebastien.darche *at* polymtl.ca>

August 12th, 2024

Dorsal - Polytechnique Montréal

Current projects at DORSAL lab

- The Distributed Open Reliable Systems Analysis Lab
- Strong focus on trace collection and performance analysis
- LTTng, Trace Compass

Tooling for HPC

- Score-P traces support through CTF conversion, ROCm runtime instrumentation
- Multiple analyses available
  - Critical path for linux kernel traces
  - Hardware performance counters through Score-P
  - Call stack among ranks, statistics
  - Flame graph
  - Communicators, bandwidth
  - Critical path for MPI
  - ...
- Scalability of Trace Compass through distributed analyses (ongoing work)
- Current work on kernel instrumentation

GPU Tracing with `hip-analyzer`

- Few tools for tracing on GPUs, and often at the cost of very high performance impact (at minima 10× and up to 120×) [1] [2]
- GPU Tracing is unwieldy : clumsy memory management, massive parallelism (concurrency control, high throughput)
- Separate buffer allocation and event collection using two kernel runs
- LLVM IR instrumentation

What's new ?

- Improved instrumentation through asm constructors & reg. reservation
- Improved results over state-of-the-art methods
- Successfully instrument (and run) all tested kernels
- Method and results published in ACM TOPC [3]

Results

- Instrumentation tested against the Rodinia benchmark [4]

	Mean overhead	Median overhead
Counters instr. (kernel)	2.00×	1.67×
Tracing instr. (kernel)	1.50×	1.29×
Program execution time	1.60×	1.26×

- Good improvements over state of the art
- Correlation between kernel complexity and overhead

Limitations

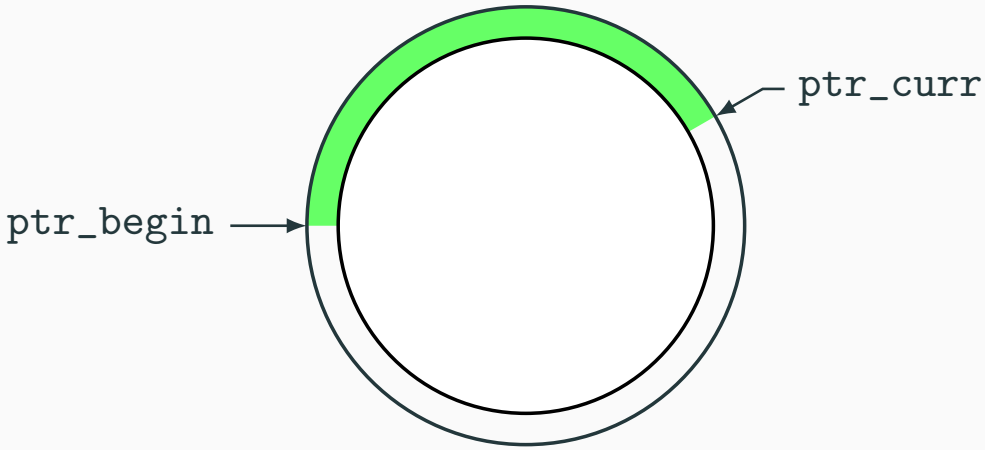
- Two kernel runs problematic for a number of reasons
  - Run time cost
  - Memory requirements
  - Non-deterministic kernels
- Invasive instrumentation

Dynamic Tracing

- Solution would be to adapt traditional tracing methods to highly parallel GPU applications
- Many setbacks
  - Memory requirements
  - Concurrency & perf. impacts
  - Data movement (host/device) and consistency

Tracing

- Ring buffer based
- Usually one per CPU core
- A collection thread runs asynchronously to empty the buffers as they are being filled by the traced application



How about on GPUs ?

- Closest equivalent to cores would be Compute Units (CU/SM)
- Shared L1 cache, execute one wavefront at a time
- ... but way more CUs on a GPU
- Memory consistency is ensured at kernel bounds

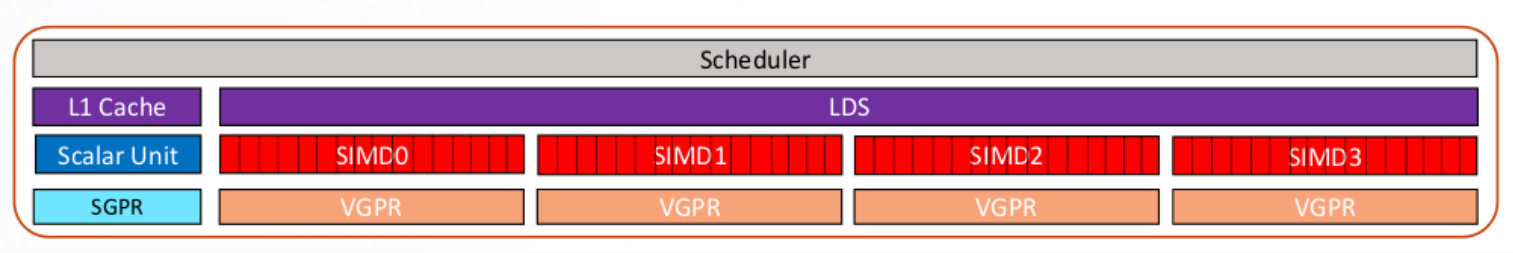


Figure 1: AMD GCN Compute unit [1]

[1] Reproduced from *AMD GPU Hardware Basics*, 2019 Frontier Application Readiness Kick-off Workshop

Challenging Scale

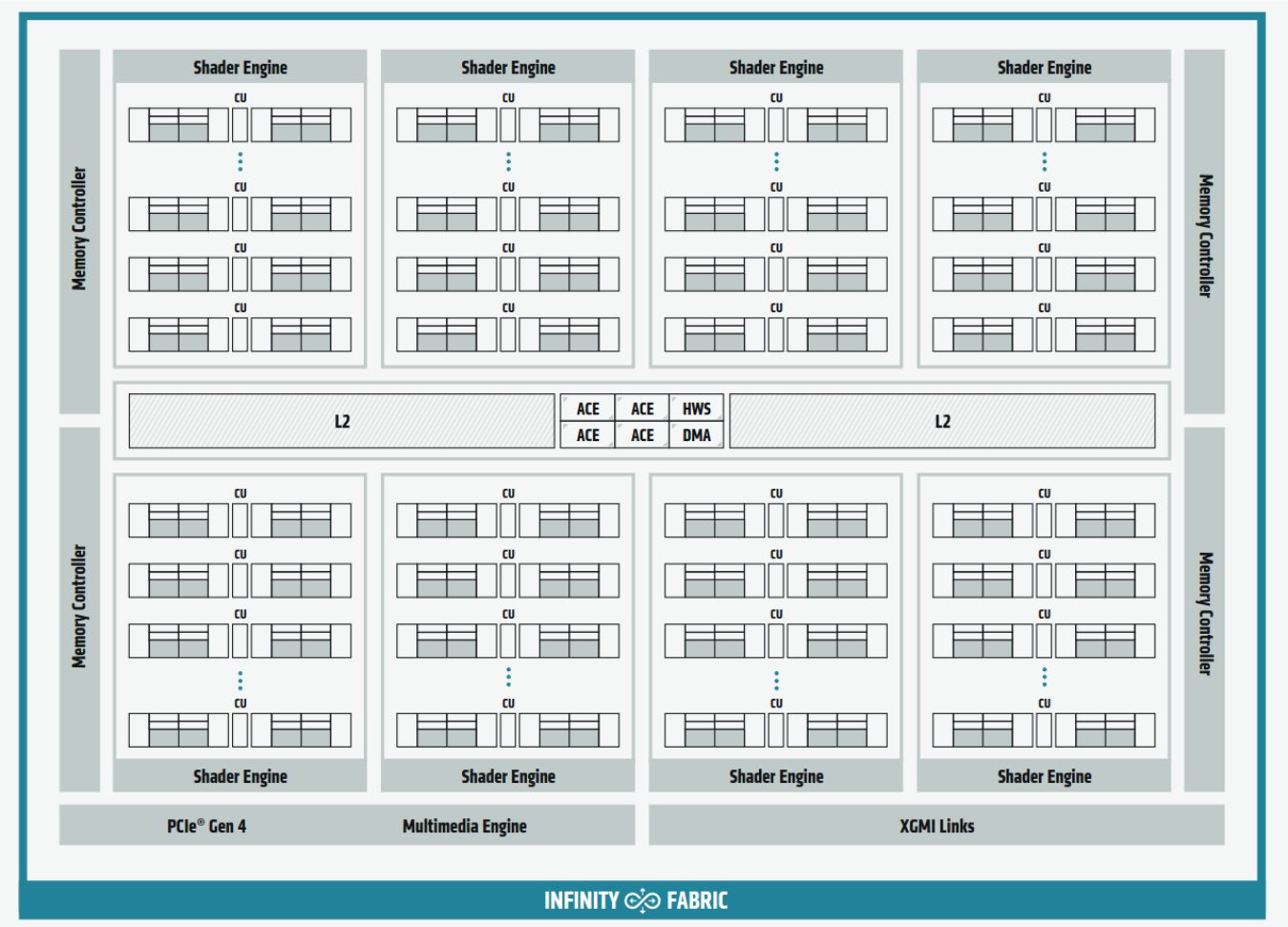
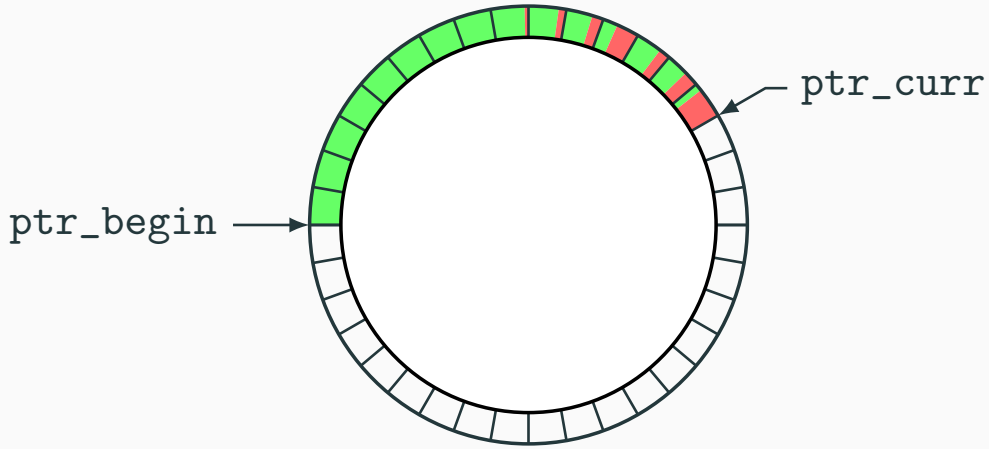


Figure 2: AMD CDNA1 Architecture block diagram [2]

[2] Reproduced from *Introducing AMD CDNA Architecture*, 2020 AMD Whitepaper

Reducing concurrent interactions

- Allocate a "chunk" of the buffer
- Implementation is more tricky (... still in asm)
- More fragmentation
- Can be combined with the previous approach

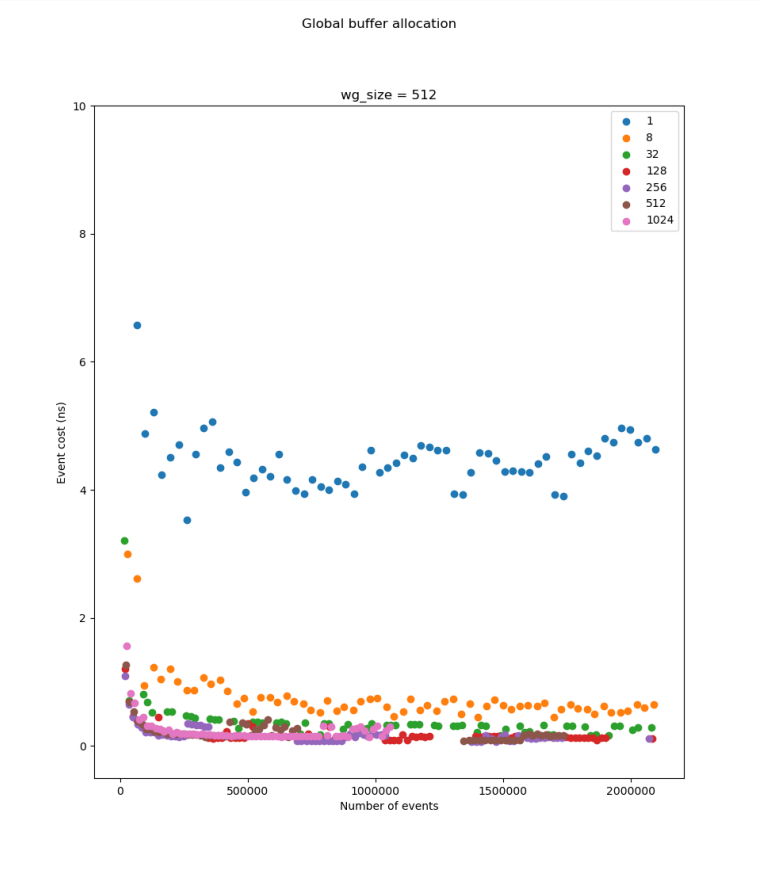


Preliminary results

- Per-event allocation works best for smaller kernels (less bookkeeping)
- Chunk allocation is the most viable option for long-running kernels (if you can amortize the initial allocation cost)
- The original hip-analyzer remains a good option for simpler kernels (as long as memory / deterministic execution is not an issue)

- Overall, the performance remains good ($< 3\times$)

Performance



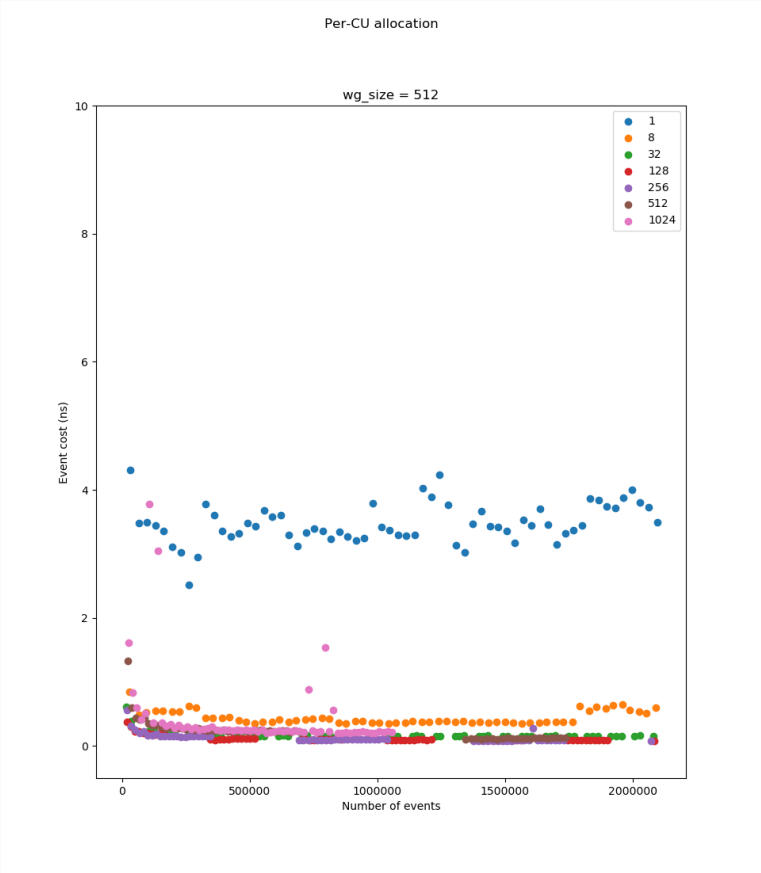


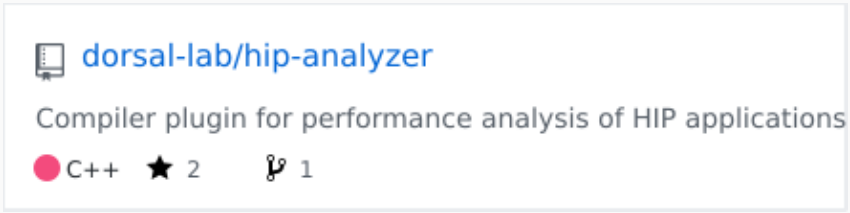
Figure 3: Event cost (time per event) without and with CU data locality on the device

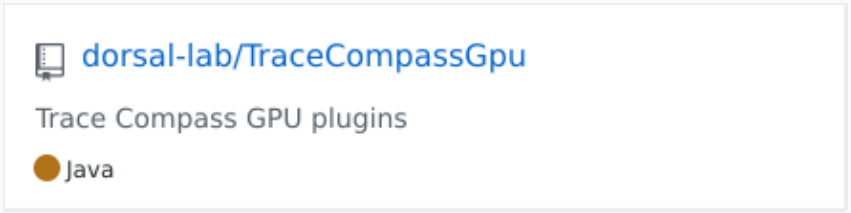
// TODO

- Identify trends among kernels to recommend best tracing method
  - Expected number of events
  - Register usage
  - Launch grid size
  - Kernel run time
  - ...
- Better instrumentation / compiler integration
  - Static / dynamic instrumentation tools
  - Backend implementation
- Explore async. event collection on APUs (MI300 ?)

Conclusion and future work

- Encouraging results and feedback
- Runtime event collector implemented & working
  - Currently running experiments on different kernels
- Next step is working on better instrumentation
- Available freely on Github, feedback and/or use cases are more than welcome





Q&A

References

[1] D. Shen, S. L. Song, A. Li, and X. Liu, **"Cudaadvisor: Llvm-based runtime profiling for modern gpus,"** in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018.

[2] Y. Arafa, A.-H. Badawy, A. ElWazir, *et al.*, **"Hybrid, scalable, trace-driven performance modeling of gpgpus,"** in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[3] S. Darche and M. R. Dagenais, **"Low-overhead trace collection and profiling on gpu compute kernels,"** *ACM Transactions on Parallel Computing*, vol. 11, no. 2, pp. 1–24, 2024.

[4] S. Che, M. Boyer, J. Meng, *et al.*, **"Rodinia: A benchmark suite for heterogeneous computing,"** in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.