Low-overhead trace collection and profiling on GPU compute kernels

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

June 1st, 2023

Dorsal - Polytechnique Montréal

Introduction

- GPUs have become ubiquitous in many fields, notably HPC and machine learning
- Multiple programming models have been developped, both low and high level
  - CUDA, HIP, OpenCL
  - SYCL, OpenMP, OpenACC
- GPU programming remains a difficult task

Motivation

- Tooling is maturing, mostly for profiling from the host point of view
  - ROC-profiler
  - Intel VTune
  - HPCToolkit[1], ...
- Most tools rely on hardware performance counters and/or PC sampling
- Current work on device instrumentation
- Little consideration for instrumentation noise (runtime overhead, register pressure, . . .)

[1] K. Zhou, L. Adhianto, J. Anderson, *et al.*, "Measurement and analysis of gpu-accelerated applications with hpctoolkit," *Parallel Computing*, vol. 108, p. 102 837, 2021.

Shortcomings of current work

- CUDAAdvisor[2] proposes LLVM-based instrumentation of compute kernels. PPT-GPU[3] is similar, with dynamic instrumentation.
  - little consideration for overhead (costly kernel-wide atomic operations)
  - Overhead ranging from $\sim 10\times$ to $120\times$
- CUDA Flux[4] introduces CFG instrumentation combined with static analysis
  - only one thread is instrumented, does not support divergence
  - Overhead ranging from $\sim 1\times$ to $151\times$ (avg. $13.2\times$)

---

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

[3] 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.

[4] L. Braun and H. Fröning, "Cuda flux: A lightweight instruction profiler for cuda applications," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, 2019.

Method

We propose a method for instrumenting kernel execution on the GPU with a minimal runtime overhead.

- Relies on a set of LLVM passes for the host and device IR
- Multi-stage performance analysis
  - CFG counters to retrieve the control flow of the program
  - Event collection for precise analysis
  - Optionally, original kernel for timing data
- Knowledge of the control flow allows for pre-allocation of the buffers
- Deterministic execution is ensured by reverting memory

Results

- Instrumentation tested against the Rodinia benchmark[5]

	Mean overhead	Median overhead
Counters instr. (kernel)	2.3×	1.32×
Tracing instr. (kernel)	3.23×	2.34×
Program execution time	4.19×	1.68×

- Good improvements over state of the art
- Significant outliers (large kernels are challenging!)

[5] 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.

What's new

- Mostly focused on comprehensive exam (offically a PhD candidate!)
- Strong connection with partners (AMD, ANL, LLNL). Expressed interest in precise timing information on kernels.
- Article on first research track 80% ready, needs rework
- Further reducing overhead through scalar instructions, fairly reliable method

Scalar instructions

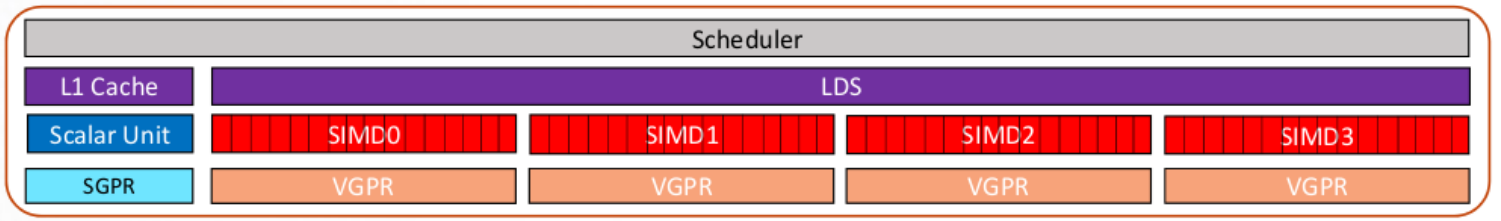


Figure 1: AMD GCN Compute unit[6]

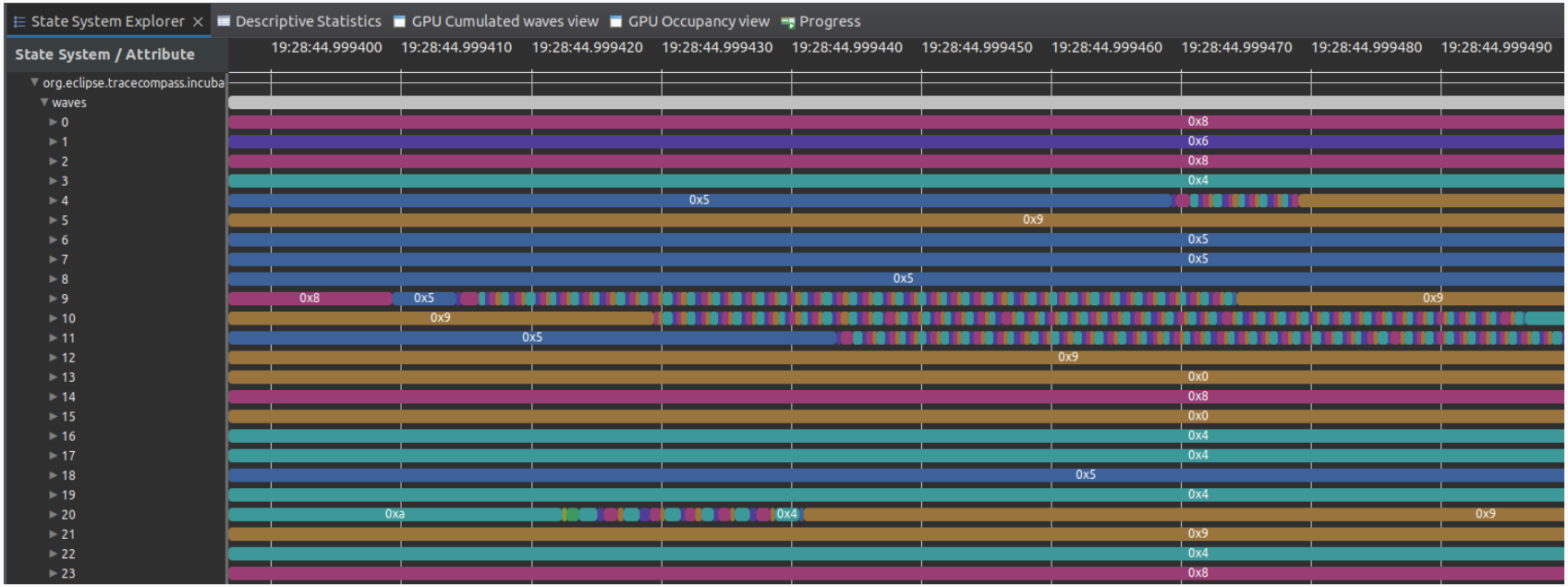
- A special set of instructions and registers are shared amongst all threads in a wavefront (SALU, SGPRs)
- Most tracepoints are at wavefront-scope and thus could benefit from scalar insts. instead of a vector mask
- Requires handwritten assembly routines, not "LLVM IR friendly"

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

Quick example

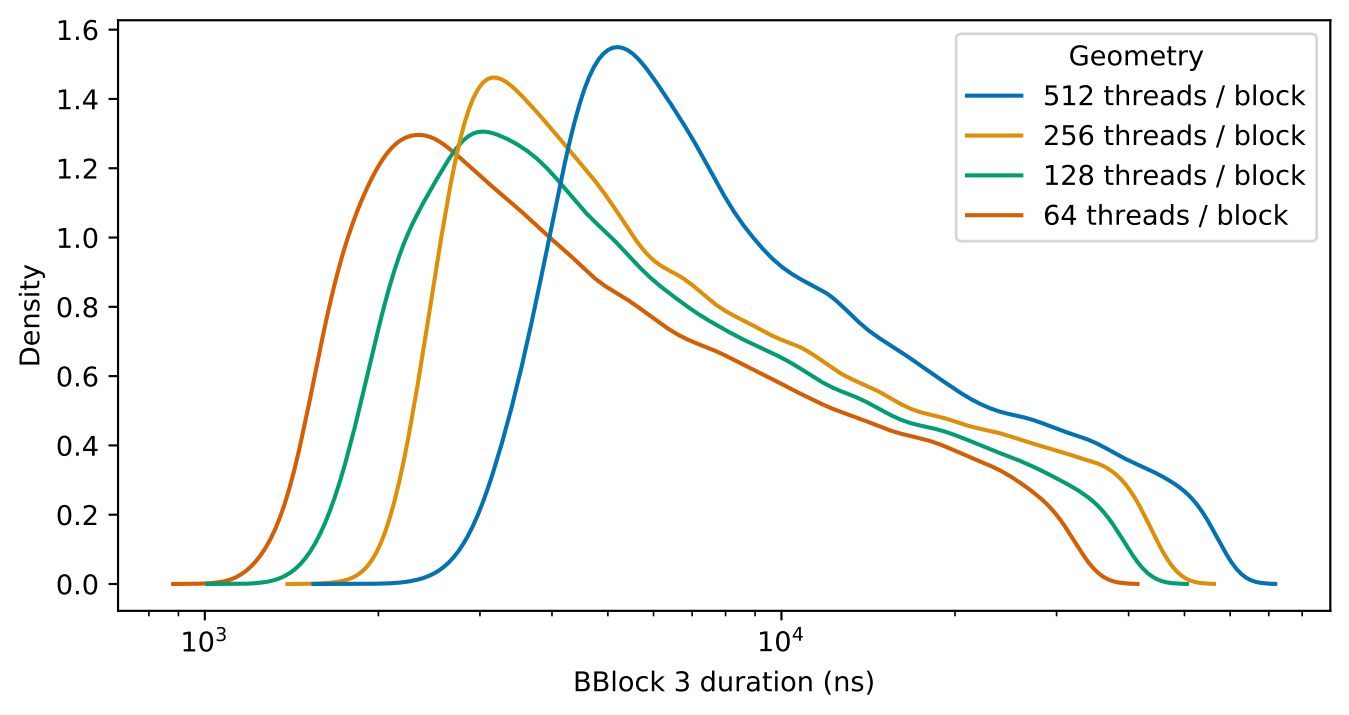
- CFG counters can generate the total number of FLOPs
- Original run allows us to compute the Arithmetic Intensity (FLOPs/s)
  - A quick roofline plot shows we're below theoretical maximum performance
- We decide to collect more data for analysis with the event collection pass
  - Precise thread divergence
  - If needed, obtain accessed addresses for locality analysis

State system analysis



Which basic block each wavefront is executing. Kernel performs a lookup on an open-addressing hashmap.

Precise timing information



Identify timing information in a "hotspot" of the code. How long the lookup takes, as a function of block geometry.

Conclusion and future work

- Encouraging results and feedback
- Runtime event collector on the GPU is on the way
  - would eliminate the need for the first CFG run
  - particularily challenging to implement!
- Available freely on Github, feedback and/or use cases are more than welcome