Low-overhead trace collection and profiling on GPU compute kernels

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

December 7th, 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*, t. 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 Control-Flow Graph (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, p. 1-15.
4. L. Braun et 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 Intermediate Representation (IR)
- Multi-stage performance analysis
    - Control-flow 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

What's new

- Deep dive into GPU architecture, new instrumentation written directly in assembly
- Vast improvements over previous versions, especially for large kernels
- Revised article submitted – awaiting reviews
- Exploring runtime trace collection on GPU

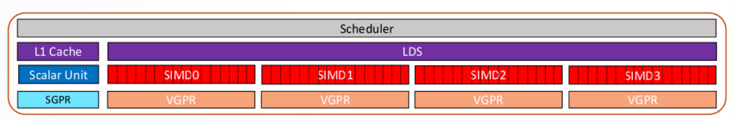


Figure 1 – AMD GCN Compute unit [5]

5. Reproduced from *AMD GPU Hardware Basics*, 2019 Frontier Application Readiness Kick-off Workshop

Results

- Instrumentation tested on the Rodinia[6] benchmark

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

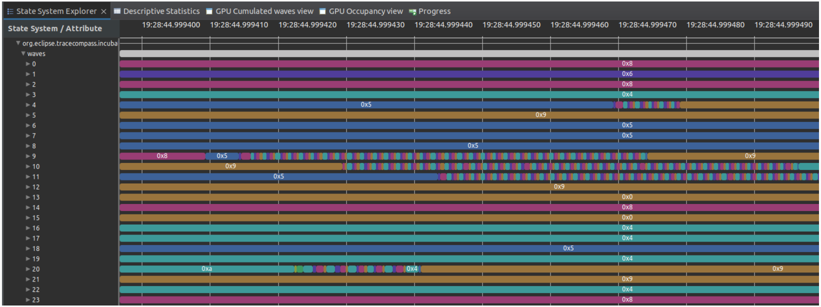
- Good improvements over state of the art
- Correlation between kernel complexity and overhead

6. 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, p. 44-54.

Runtime trace collection

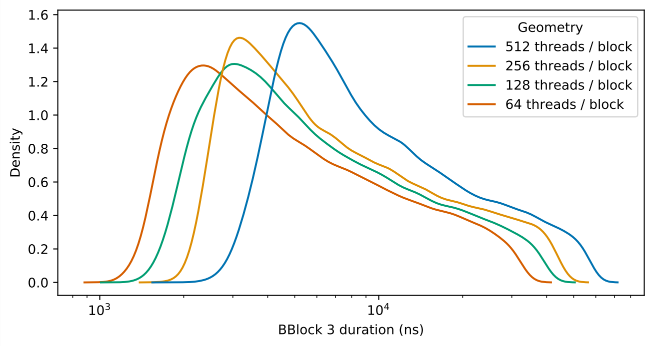
- Implemented a baseline, "naive", trace collection scheme
  - Single shared buffer in global device memory – no resizing possible
  - Enqueuing relies on heavy use of atomics
- Requires specific tuning for the hardware
  - Memory locality
  - Allocation granularity
- Many GPU allocation algorithms to explore !

State system analysis



Which basic block is executed by each wavefront. 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.

Future tracks

- Hardware optimized tracing and improved host–device interactions for memory management
- Better compiler integration through intrinsics
- Improved static analysis to reduce instrumentation

Conclusion and future work

- Encouraging results and feedback
- Exploring improvements on the method through memory management on the device
- Exciting new tracks and partnerships
- Available freely on Github, feedback and/or use cases are more than welcome

dorsal-lab/hip-analyzer

Compiler plugin for performance analysis of HIP applications

C++ 2 1

dorsal-lab/TraceCompassGpu

Trace Compass GPU plugins

Java