POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Low-overhead trace collection and profiling on GPU compute kernels

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

May 12th, 2025

Dorsal - Polytechnique Montréal

Introduction

- GPUs have become ubiquitous in many fields, notably HPC and machine learning
- Multiple programming models have been developed, 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 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. $13.2\times$)

Baseline 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
- Article published in the ACM Transactions on Parallel Computing[2]

[2] S. Darche and M. R. Dagenais, **"Low-overhead trace collection and profiling on gpu compute kernels,"** *ACM Trans. Parallel Comput.*, vol. 11, no. 2, Jun. 2024, ISSN: 2329-4949. DOI: 10.1145/3649510. [Online]. Available: https://doi.org/10.1145/3649510.

Baseline Results

- Instrumentation tested on the Rodinia[3] 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

[3] 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.

Runtime Trace Collection

- First approach works well, but is unweildy in many ways
  - Two kernel runs require saving context & input data
  - Limited by non-deterministic kernels (parallelism?)
- "Regular" tracing is possible but has its own set of challenges
- Requires specific tuning for the hardware
  - Memory locality
  - Allocation granularity
- Second article covers a few methods (*cf* last progress report meeting)

Results

- Instrumentation tested on the HeCBench[4] benchmark. Overhead is reported as the slowdown factor between the traced kernel execution time and the original, uninstrumented kernel.

	mean	median
`hip-trace`	2.07×	1.50×
4 × padded `hip-trace`	2.18×	1.58×
`hip-global-mem`	3.73×	1.96×
`hip-cu-mem`	2.47×	1.60×
`hip-chunk-allocator`	1.79×	1.33×
`hip-cu-chunk-allocator`	1.77×	1.32×

[4] Z. Jin and J. S. Vetter, **"A benchmark suite for improving performance portability of the sycl programming model,"** in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2023, pp. 325–327.

Reducing the number of tracepoints

- For most kernels, the number of tracepoints could be reduced
  - Reduction in trace size
  - Reduction in run time overhead
- Intuitively, if half of the threads go through an *if* statement, we can deduce the other half goes to the *else* statement
- Can be generalized to switch statements and more complex control flow (more than two outgoing edges)

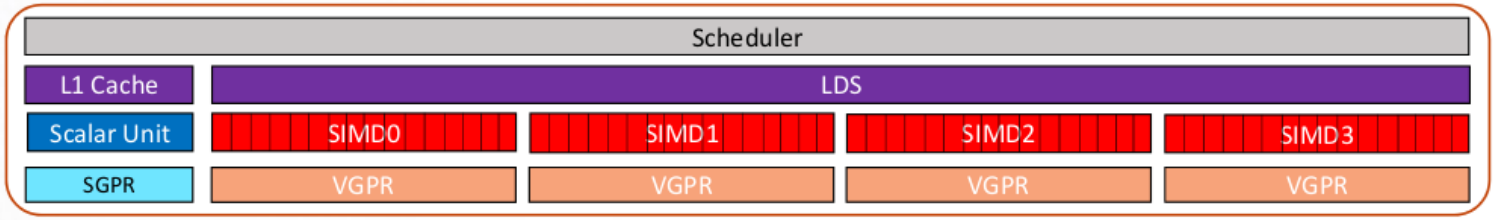


Figure 1: AMD GCN Compute unit [5]

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

Static analysis

- In an acyclic CFG (simple case), the control flow can be completely computed by instrumenting $n - 1$ outgoing edges
- Vertices in the CFG are processed using a variant of Kahn's algorithm

$$\bigvee_{e_i \in \text{incoming}} T(e_i) = \bigvee_{e_i \in \text{outgoing}} T(e_i) \tag{1}$$

- The algorithm does not terminate for CFGs containing a cycle
- We identify back edges using a depth-first search (DFS), and run the algorithm on the CFG stripped of its cycles. Back edges must be instrumented.

Static analysis

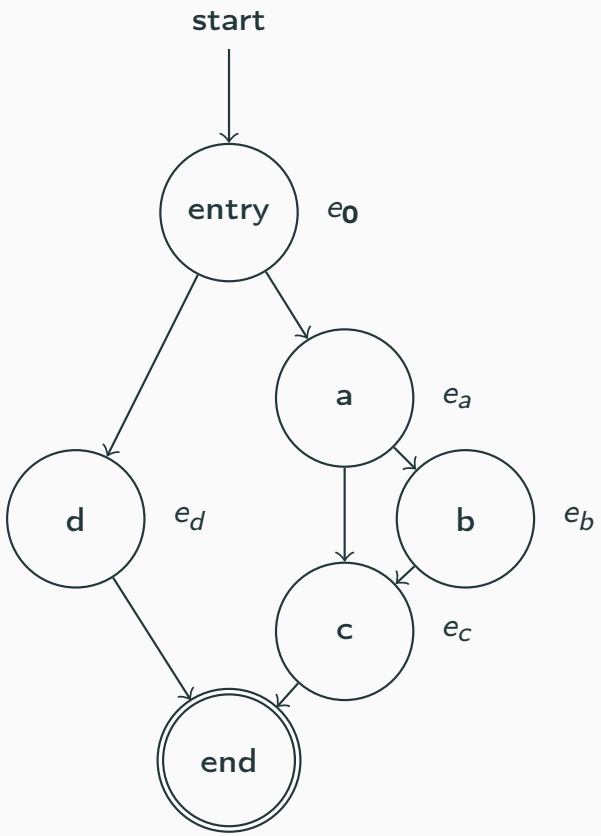
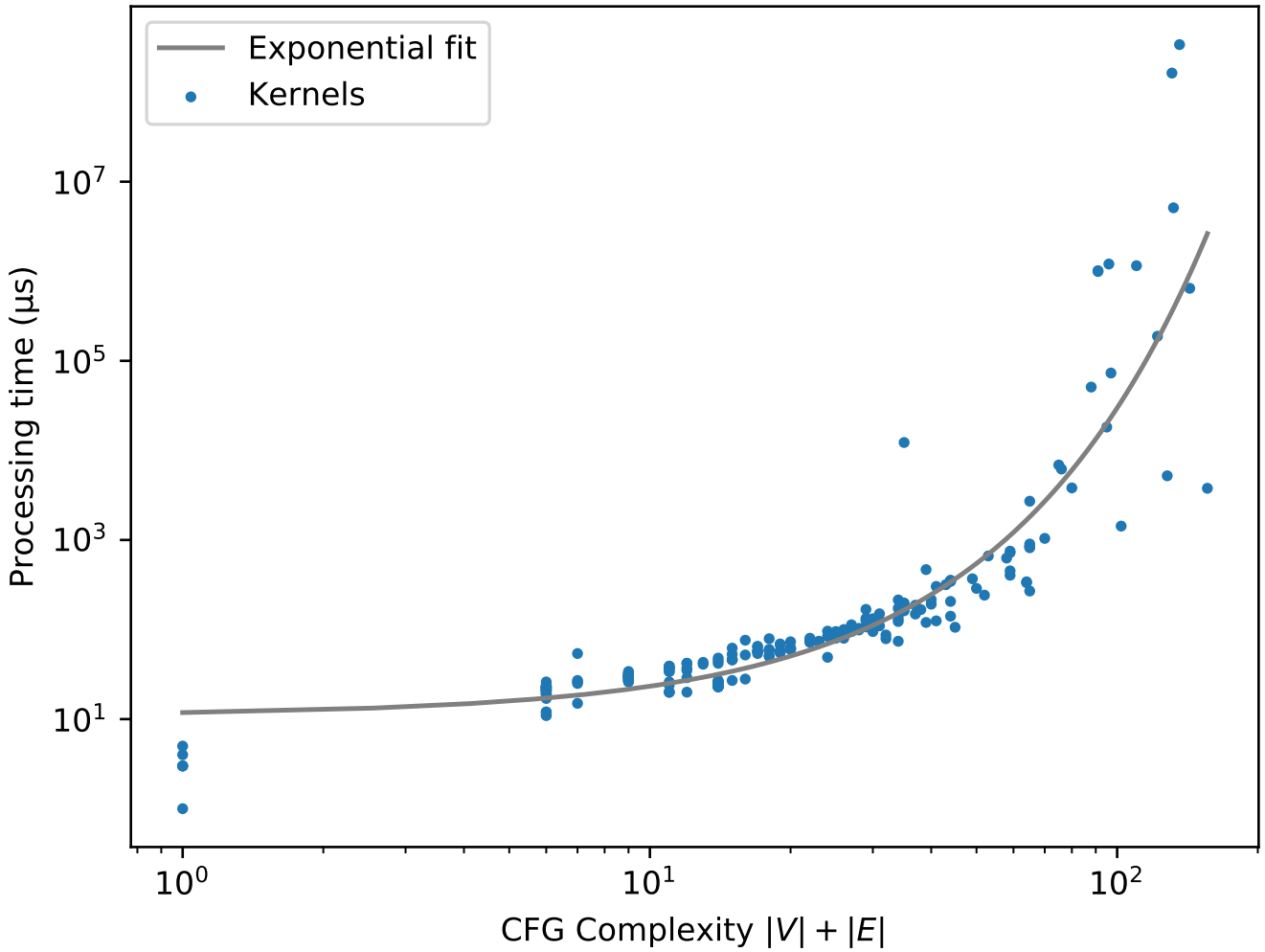


Figure 2: Thread-centric CFG Example

- $e_a = \overline{e_d} \cdot e_0$
- $e_c = e_a + \overline{e_b} \cdot e_a = e_a$
- $e_{end} = e_d + e_c = e_d + e_a = e_d + \overline{e_d} \cdot e_0 = e_0$

Analysis run time complexity

- DFS is expensive, exponential complexity



Trace size reduction

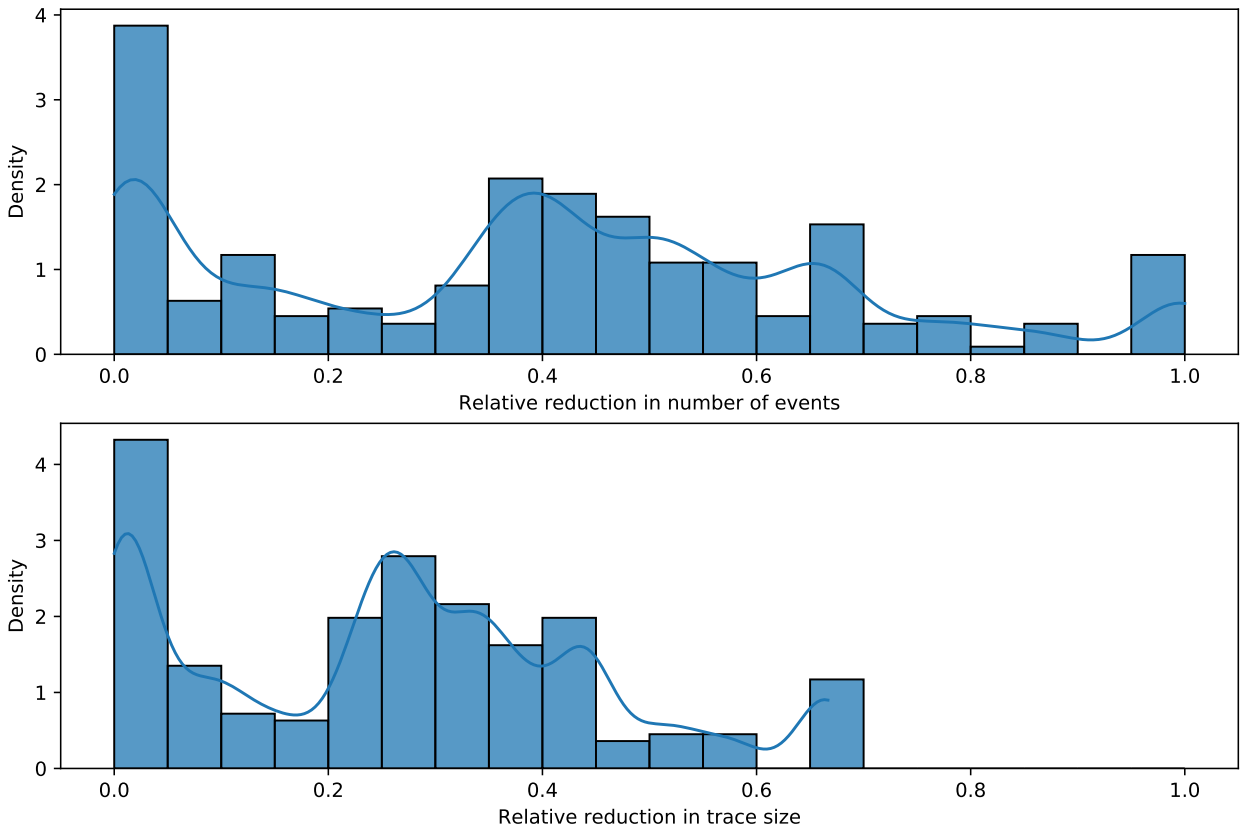


Figure 4: Relative reduction in number of events and total trace size

Run time overhead

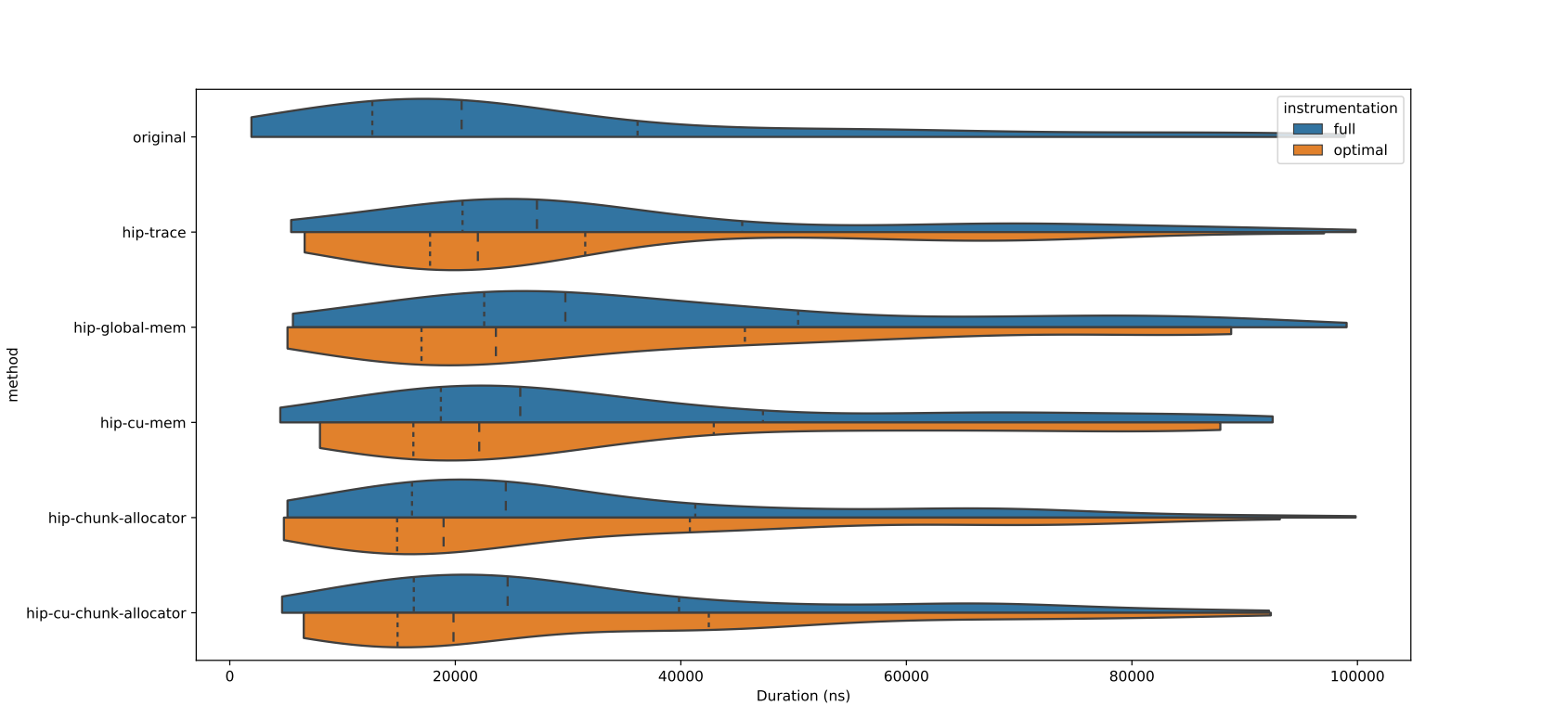
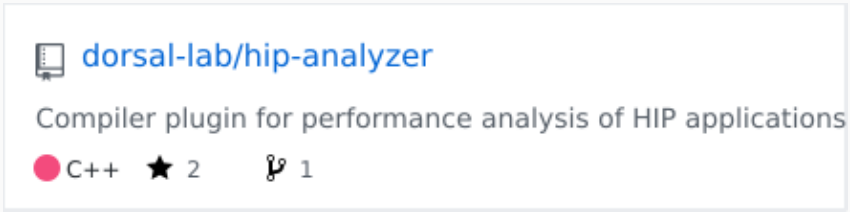
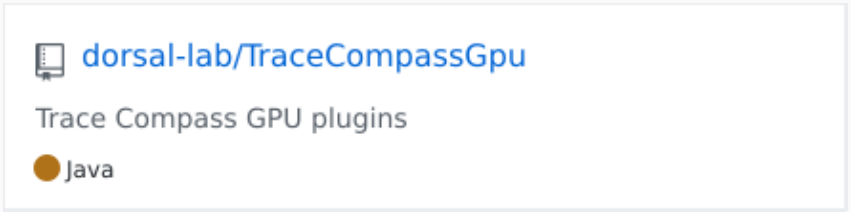


Figure 5: Distribution of kernel run time as a function of collection method and instrumentation

Conclusion and future work

- PhD project is nearing its end
- Explored instrumentation methods for tracing compute kernels
- Studied the performance impact of data structures for online tracing
- Improved baseline results by reducing the number of tracepoints
- Interest for the project from partners
- Available freely on Github, feedback and/or use cases are more than welcome





References

[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.

[2] 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.

[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.