Heterogeneous HPC Performance Analysis with Trace Compass

Low-overhead Trace Collection on GPU

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

June 19th, 2023

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 (ongoing work)
  - ...
- 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
- GPU Tracing is unweildy : clumsy memory management, massive parallelism (concurrency control, high throughput)
- Why not separate buffer allocation and event collection ?

Similar work

- CUDAAdvisor[1] proposes LLVM-based instrumentation of compute kernels. PPT-GPU[2] 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[3] 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. $13.2\times$)

---

[1] 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.

[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] 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

- Relies on a set of LLVM passes for the host and device IR
- Multi-stage instrumentation
  - 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 buffer (and producer offset!)
- Deterministic execution is ensured by reverting memory

Results

- Instrumentation tested against the Rodinia benchmark[4]

	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!)

[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.

Scalar instructions

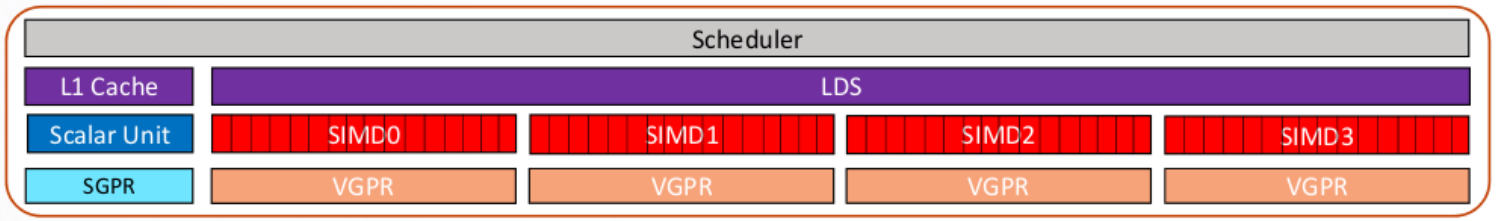


Figure 1: AMD GCN Compute unit[5]

- 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"

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

Scalar instructions - LLVM IR

- IR is still thread-centric, vectorization is done in the `amdgpu` backend
- `alloca`-ted values (locals) are only stored in VPGRs, SALU asm crashes the compiler
- Solution : Chain of PHI nodes to pass global wavefront state throughout the kernel

Scalar instructions - LLVM IR

```
define i32 @func(ptr %0, i32 %1) {
    ; ...

    br label %cond

  cond:                         ; preds = %entry, %body


    ; ...
    br i1 %compare, label %body, label %end

  body:                         ; preds = %cond


    ; ...
    br label %cond

  end:                          ; preds = %cond

    ; ...
    ret %value
}
```

Scalar instructions - LLVM IR

```
define i32 @func(ptr %0, i32 %1) {
    ; ...
    %ctr_entry = call i32 asm "...", "..."()
    br label %cond

  cond:                         ; preds = %entry, %body
    %ctr_cond = phi i32 [ %ctr_entry, %entry], [ %ctr_body_inc, %body ]
    %ctr_cond_inc = call i32 asm "s_add_u32 $0, $0, 1", "=s,s"(i32 %ctr_cond)
    ; ...
    br i1 %compare, label %body, label %end

  body:                         ; preds = %cond
    %ctr_body = phi i32 [ %ctr_cond_inc, %cond ]
    %ctr_body_inc = call i32 asm "s_add_u32 $0, $0, 1", "=s,s"(i32 %ctr_body)
    ; ...
    br label %cond

  end:                          ; preds = %cond
    %ctr_end = phi i32 [ %ctr_cond_inc, %cond ]
    ; ...
    ret %value
}
```

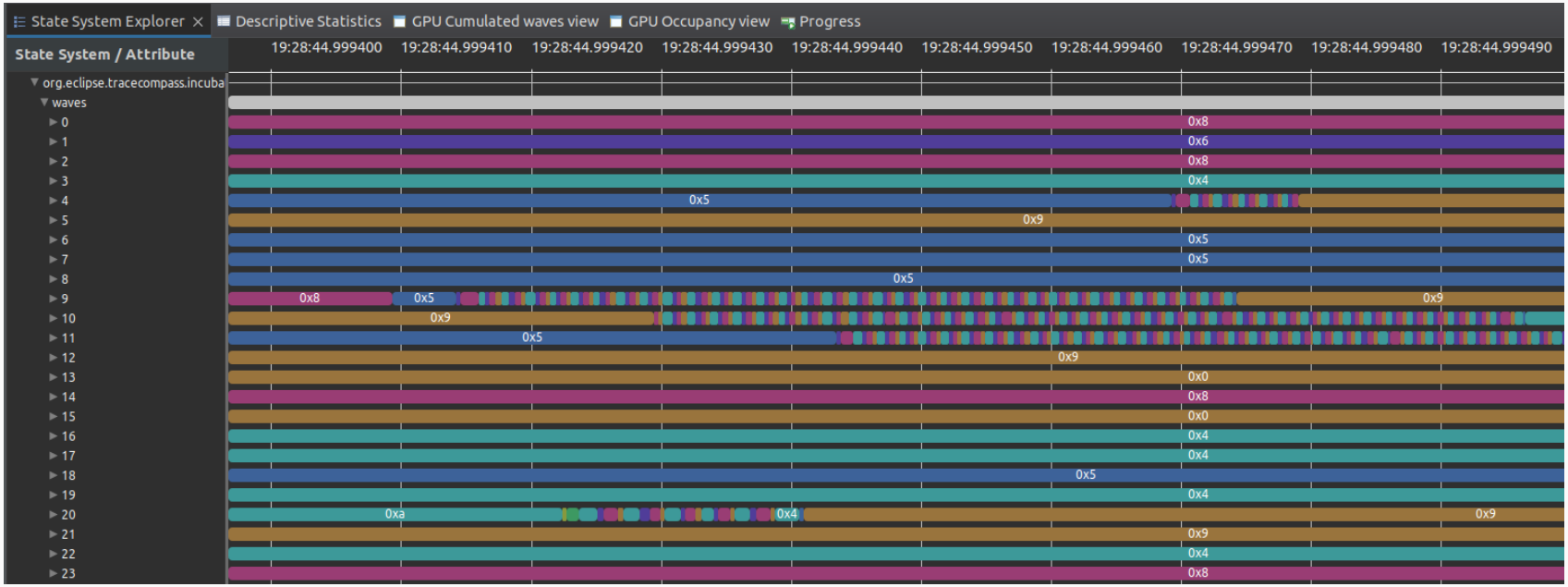
Wavefront status

- What can we gain from instrumenting the wavefronts ?
  - Precise timestamps : timing analysis
  - Trace active threads (`EXEC` mask)
  - Microarchitecture-specific info (`HW_ID` register)
- Per-CU performance counters ?

Quick example

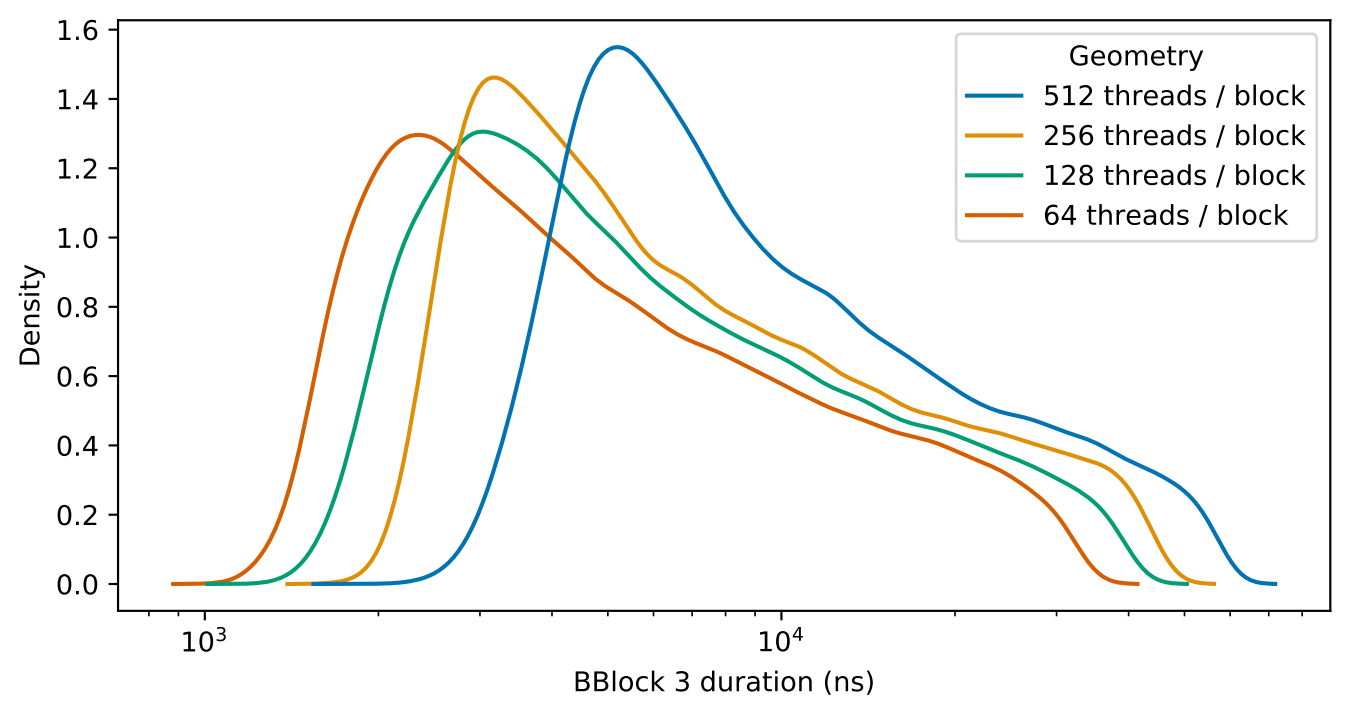
- CFG counters can generate the total number of FLOPs
- Original run allows us to compute the Arithmetic Intensity (FLOPs/s)
  - A quick roofline plot shows we're below theoretical maximum performance
- We decide to collect more data for analysis with the event collection pass
  - Precise thread divergence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

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.

Challenges

- IR instrumentation, go deeper ?
- Trace size
  - Throughput
  - Analysis time
- Scalability with large kernels

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