MadGraph4GPU

Kernel Profiling

A. Thete & C. Vuosalo

University of Wisconsin-Madison

11.06.2023

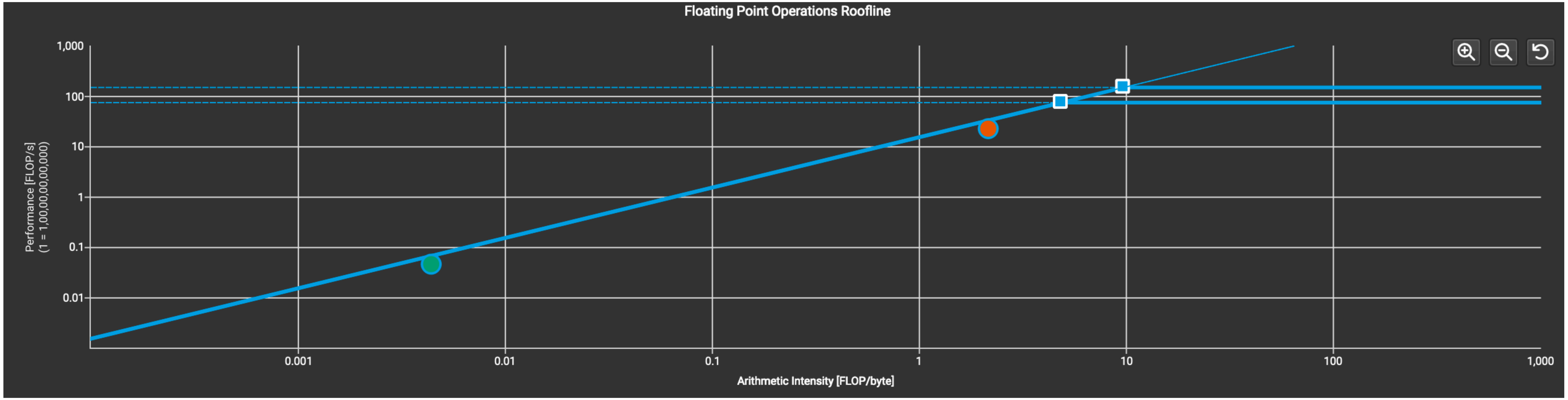
Launch Configuration

- Jobs run on a shared node with a single NVIDIA A100 GPU (Compute Capability 8.0) with a 40GB DRAM
- `nvcc` 12.1.105, kernel launch configuration: <<<216 blocks, 256 threads/block>>>
- More relevant A100 stats:
    - 128 SMs
    - 64 fp32 units/SM, 32 fp64 units/SM
    - 64 warps/SM
    - 1555 GB/s memory bandwidth
    - 256 KB register file/SM; upto 255 registers/thread
    - Upto 164 KB Shared Memory/SM

Overview

- Profiling `mg5amcGpu::sigmaKin` for the process gg -> ttgg
- Speed-of-light Metrics
    - GPU wall time: 67.28 ms
    - High memory throughput (63.9%); cf compute (39.08%) — kernel is memory-bound
    - Achieved fp64 performance is 10% lower than the fp64 pipeline utilization — indicates inefficient fp64 operations under the hood
- Normally, branches are largest source of latency
    - 100% branch efficiency, 0 divergent branches — all threads in a warp execute same sequence of instructions

Roofline Plot



- Red point is our achieved double precision (2.18 FLOP/byte, 2.1 TFLOPs/s); theoretical maximum (4.86 FLOP/byte, 7.5 TFLOPs/s)
- *Very* close to the roofline, so the kernel is pretty optimal as it is; can’t expect a very drastic improvement.
- Position on the plot indicates a memory-bound kernel, performance can be improved by increasing memory bandwidth and moving point “up”.

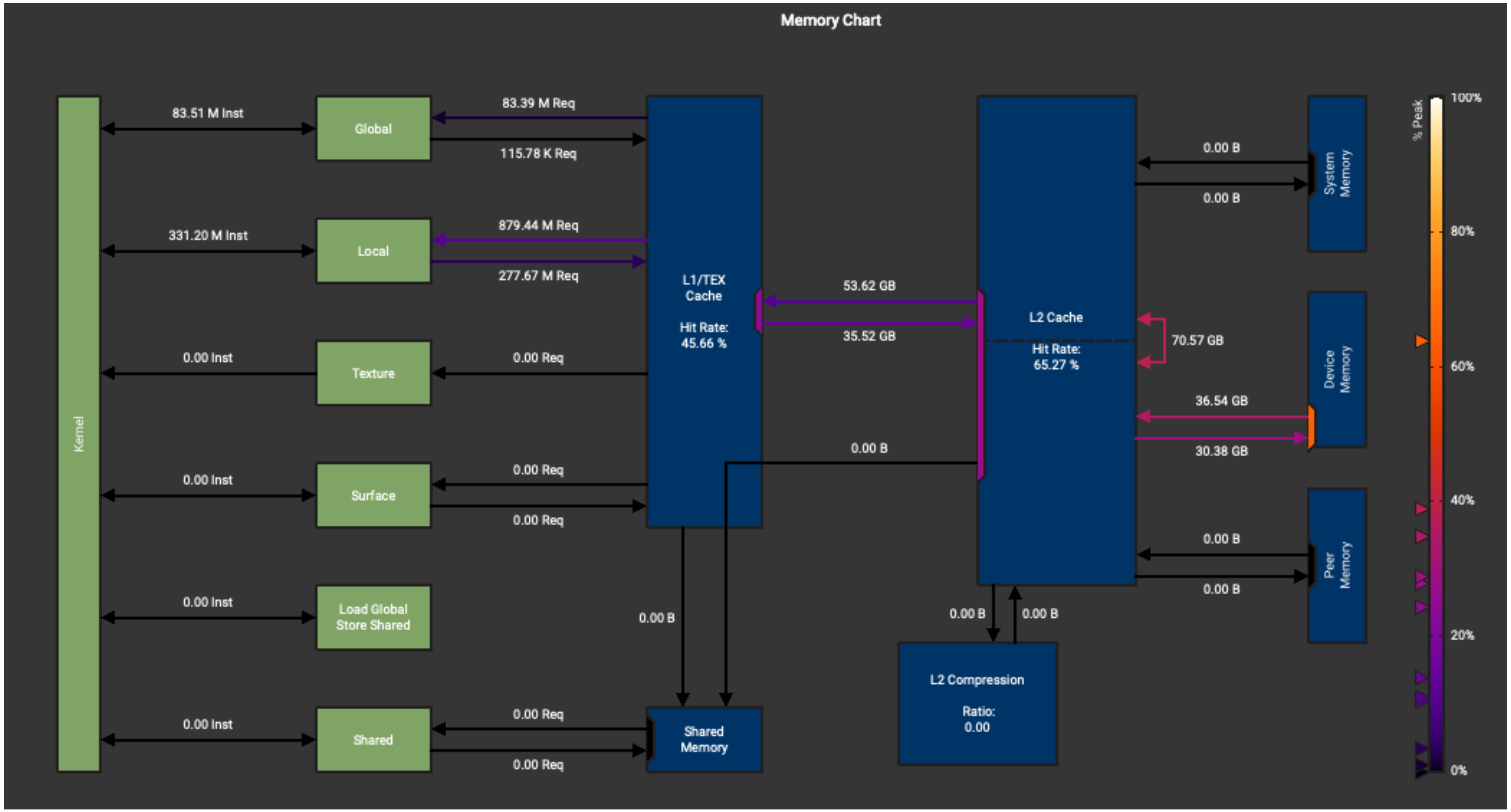
Memory Analysis

- Throughput of internal memory activity (cache/DRAM) is only 29.53%, but each warp spends **7.6 cycles being stalled** waiting for a L1 cache dependency.
    - Each warp is resident for 16.7 cycles — 45% of warp time spent stalled!
- Two memory bottlenecks in the source code:
    - CUDA Thrust libraries (can't do anything about this).
    - Line 1107 in `FFV1_0` of HelAmps_sm.h to compute output amplitude vertex from input 3 wavefunctions

```
  const cxtype_sv TMP9 = ( F1[2] * ( F2[4] * ( V3[2] + V3[5] ) + F2[5] * ( V3[3] + cI * V3[4] ) )
+ . . .
```

- HelAmps_sm.h is a madgraph-generated file for the process.
    - Bottleneck can be eased by either increasing hit rates or moving more frequently used data to shared memory
        - Hit rates at ~46% and ~66% for L1 and L2; no shared memory utilization at all (intentional?).
- Memory accesses appear to be well-coalesced, built into the code too via the AOSOA representation format used.

Memory Analysis



Compute Analysis

- Memory bottleneck leaking into compute performance too
- Kernel allocates ~2 warps/scheduler (cf. theoretical max of 16). Occupancy is being limited by the number of registers available to each thread
    - High occupancy not always an indicator of better performance, but still needs to be investigated.
    - Launch: 255 regs/thread; maximum utilized: 248 by the bottleneck
    - Out of the two active warps, each cycle only 0.17 eligible for next instruction (others are stalled)
- More parallelism can be exposed by efficiently utilizing our fp64 pipeline.
    - 1.6B non-fused fp64 instructions (cf 1.4B fused); by converting pairs of non-fused instructions to their fused counterparts, achieved fp64 performance could be increased by up to 27%.
    - nvcc enables this by default (compiler flag `—-fmad=true`), but still got this warning so don't know what to make of it.