CUDA Profiling and Tuning

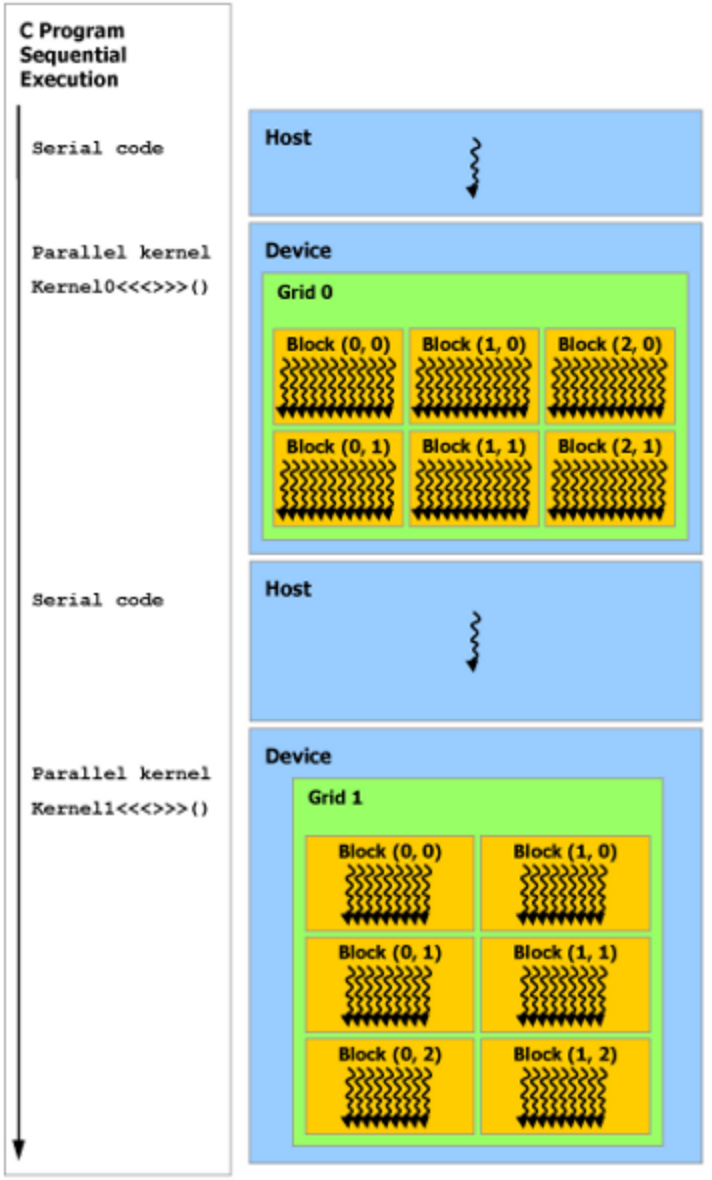
Fei Mao
HPC Technical Consultant
SHARCNET

Outlines:

- CUDA Review
- GPU Profiling and Optimization
- Nsight and NVidia Visual Profiler (NVVP) on SHARCNET
- Performance Limiter
- Warp and Occupancy
- Instruction Latency and Throughput (hardware background)
- Play with Shuffle instructions and Instruction Level Parallelism

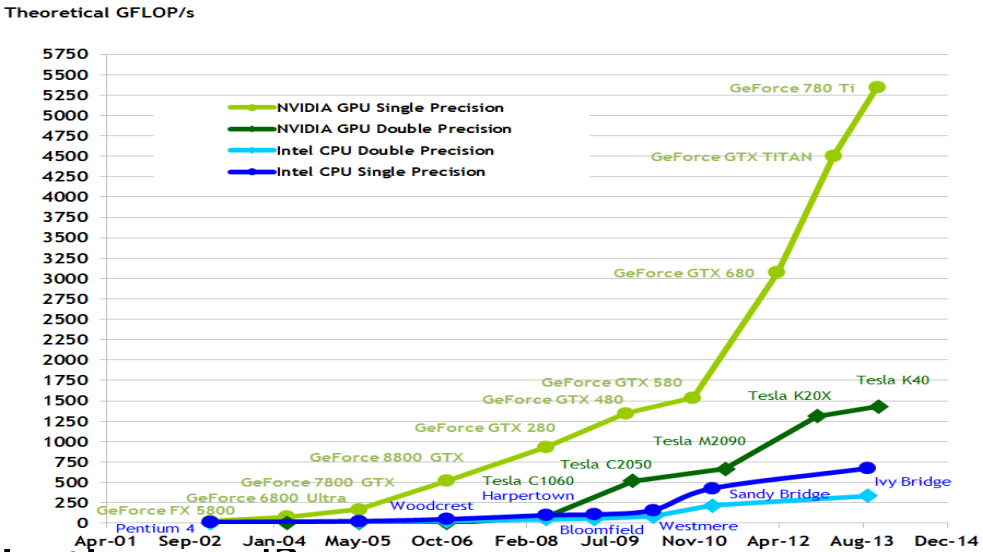
CUDA Review

- Programming model
  - Threads
  - Blocks
- Memory hierarchy
  - Global mem
  - Shared mem
  - Register
- Code flow
  - Heterogeneous Programming



GPU Profiling and Optimization

- Why do I need to profile my code (again)?



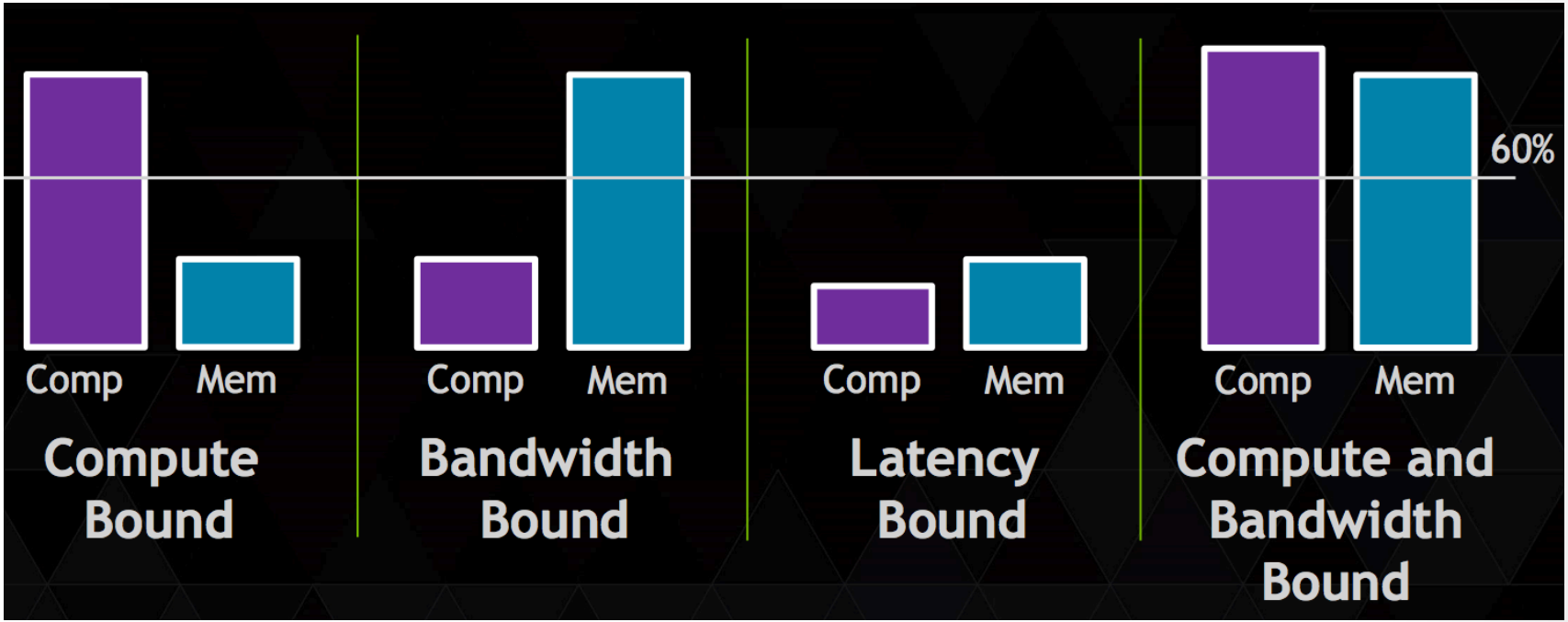
- Optimization goal?
  - Fully utilize all hardware resource

Nsight and NVVP

- Nsight: A full-featured IDE that provides an all-in-one integrated environment to edit, build, debug and profile CUDA-C applications
- NVVP: A cross-platform performance profiling tool that delivers developers vital feedback for optimizing CUDA C/C++ applications
- NVVP is built in Nsight
- Using Nsight on SHARCNET
  - Development nodes on Monk (Fermi based GPUs, be careful if GPU is being used by others)
  - Submitting interactive job to Angel (Maxwell based GPUs)

Performance Limiter

- How to tell if the code is compute or memory bound?
  - (From hardware side, not from the nature of algorithm)



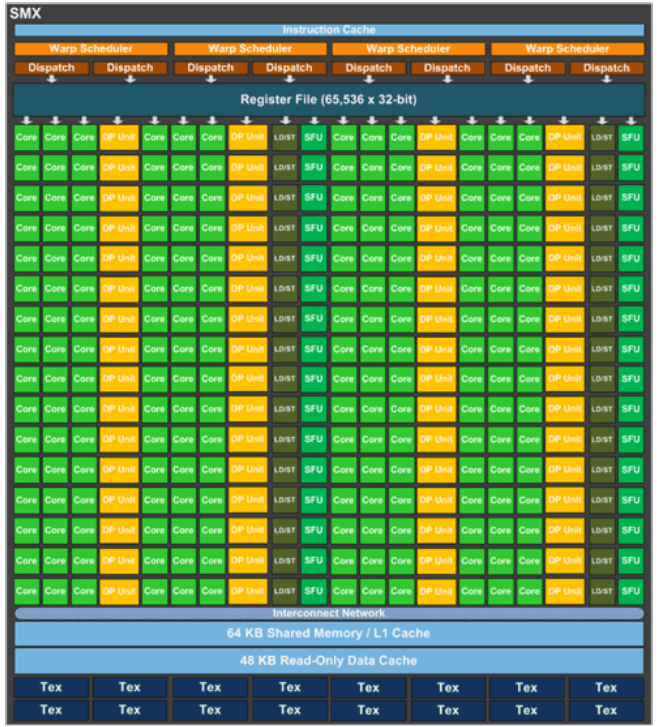
Warp and Occupancy

- Warp = 32 Threads with consecutive thread indexes executed physically in parallel (SIMD) on a multiprocessor
- Occupancy = Active Warps/ Maximum Active Warps
- Occupancy Limiter
  - Register usage
  - Shared Memory usage
  - Block size
- 100% occupancy isn't needed to reach maximum performance, Once the "needed" occupancy is reached, further increases won't improve performance
  - Many find 66% is enough
  - More independent work per thread -> less occupancy is needed
  - Memory-bound codes tend to need more occupancy
    - Higher latency than for arithmetic, need more work to hide it

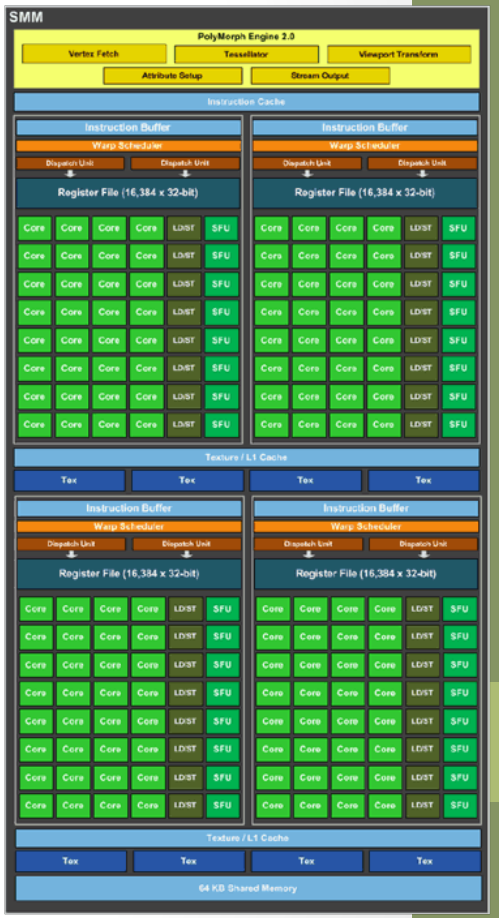
Instruction Latency and Throughput

- SM (stream multiprocessor)
  - Core
  - LD/ST unit
  - DP unit
  - SFU
- 4 schedulers
  - “dual issue” instructions to pipes
- Pipelines
  - LD/ST, Arithmetic, Control-Flow, Texture

NVIDIA Kepler SM

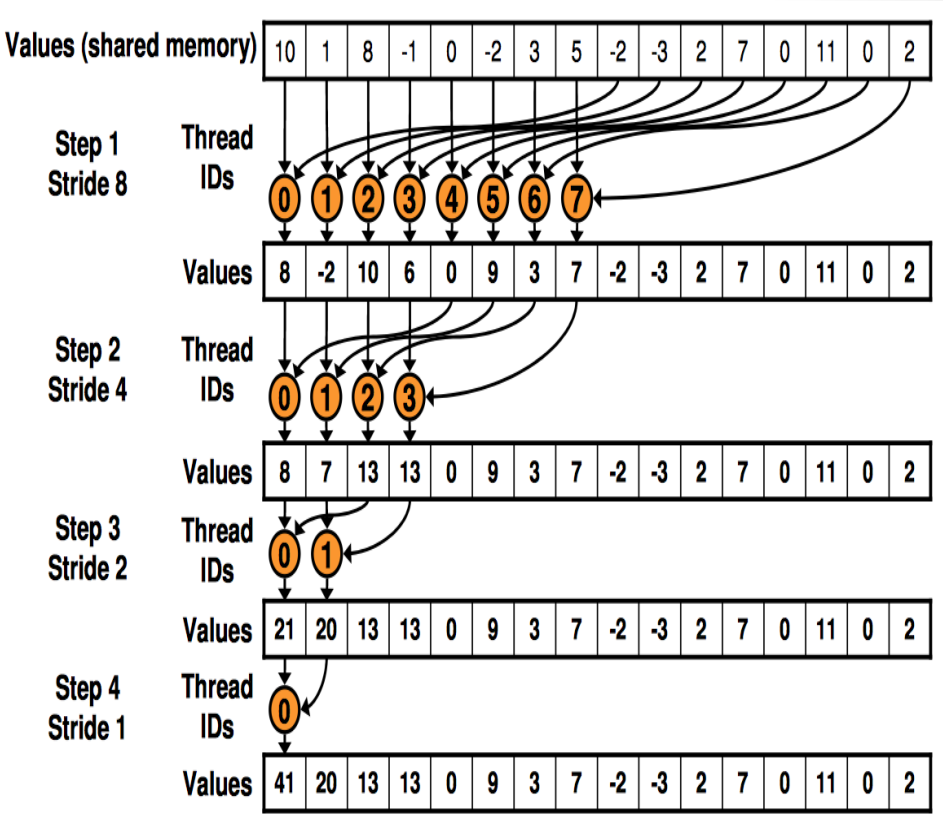


NVIDIA Maxwell SM



Example code (binary reduction)

- Basic code flow:
  - Loading data to shared memory
  - Synchronizing all warps in a block
  - Reducing to half sized data
  - Synchronizing
  - Reducing to half sized data
  - …
  - Reducing to 1 element

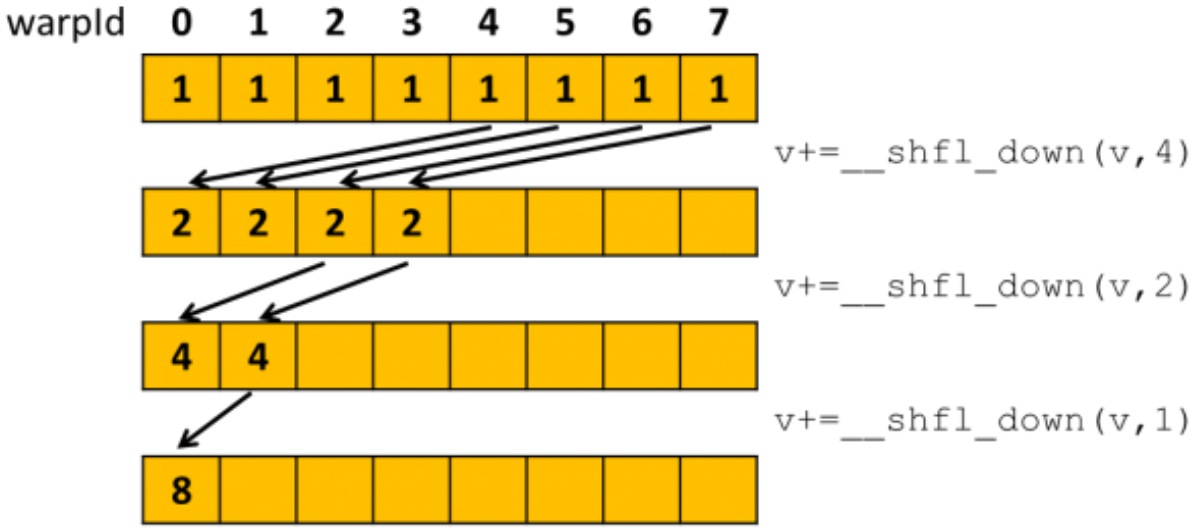


Example code (binary reduction)

- Instruction latencies
  - Execution dependency:
    - An input required by the instruction is not yet available
  - Synchronization
    - The warp is blocked at a __syncthreads() call
- Reducing synchronization latency
  - “Shuffle” instructions instead of shared memory
- Reducing execution dependency
  - Instruction-level parallelism

Shuffle instructions

- Shuffle instruction (SHFL) enables a thread to directly read a register from another thread in the same warp (32 threads)
- Only supported on Compute Capability 3.0 (Kepler) and higher
- Shuffle Warp Reduce



Instruction Level Parallelism

- Thread-level parallelism ---> Concurrent threads
- Instruction-level parallelism ---> Concurrent instructions
  - Overlapping instructions
- Instruction latencies:
  - Arithmetic: ~20 cycles
  - Reading shared memory(or L1 cache): 20-40 cycles
  - Reading global memory: 400-600 cycles
  - GPU switch warp to hide latency, issuing independent instructions
- For reduction problem:
  - Hidden independent ADD operation in LOAD operation ---> need more ADD
  - Number of instructions that can be scheduled is limited by number of active warps and number of instructions in each warp

Instruction Level Parallelism

- For reduction problem:
  - Increase number of instructions within a thread (warp)
  - Each thread adds multiple elements before binary reduction, each pair of elements is independent to others
    - Loading a pair of two elements, adding them and storing the temporal result into a register
    - Adding temporal results together and then using shuffle instructions to sum between threads

Benchmark results

- Reduction of 4K elements on K20 (theoretical memory bandwidth 208GB/s)

	Time(us)	Speed-Up	Mem bandwidth (GB/s)
Original	479	1x	46
Shuffle	318	1.5x	67
+2 ILP	155	3.1x	135
+4 ILP	135	3.5x	143
+8 ILP	116	4.1x	172
+16 ILP	114	4.2x	177

Q & A