

# CUDA: Synchronization and Scheduling

Mark Greenstreet

CpSc 418 – Mar. 9, 2016

- Summary Grids, blocks, threads, and warps.
- Synchronization
- Examples

# Grids, blocks, threads, and warps

An nVidia glossary

- When a kernel is launched:
  - ▶ The code running on the CPU calls a function that dispatches code to execute on the GPU.
  - ▶ Execution on the GPU is performed by an array of threads called a grid.
- A grid is organized as a 1D or 2D array of **blocks**.
  - ▶ Blocks managed by software.
  - ▶ Lots of blocks, more switching overhead.
- Each block is a 1D, 2D, or 3D array of **threads**.
  - ▶ Threads managed by hardware.
  - ▶ Limited (only a thousand or so) threads.
  - ▶ Essentially zero thread-switch cost.

## Reduce (from Mar. 7)

```
for(int stride = n/2; stride > 0; stride >>= 1) {  
    if(my_idx < stride)  
        data[my_idx] += data[my_idx] + stride;  
    __syncthreads();  
}
```

- Consider `n == 1024`.
- In the first iteration, there are 16 active warps – all threads in each warp are busy.
- In the second iteration, there are 8 active warps – all threads in each active warp are busy.
- Similarly, for the 3<sup>rd</sup> through 5<sup>th</sup> iterations:
  - ▶ The number of active warps decreases.
  - ▶ All threads in each active warp are busy.
- **What does `__syncthreads()` do?**

# Synchronization

- The reduce example used `__syncthreads()`: all the threads **in the block** must execute this statement before any continue beyond it.
  - ▶ Be **very** careful about thread divergence.
  - ▶ All threads in the block must meet at the barrier.
  - ▶ They must all meet at the **same** barrier.
    - ★ Note about loops: that means the **same** iteration.

# Synchronization Across Blocks

- **Q:** What if you need to synchronize more threads that fit in a block?
- **A:** Launch multiple kernels.
  - ▶ Each kernel completion-followed-by-launch acts like a barrier.
  - ▶ I need to measure the timings.
    - ★ Synchronization by kernel launch is certainly slower than `__syncthreads()`
    - ★ How much slower?
- No `__syncthreads()` needed if all the threads are in the same warp.
  - ▶ Again, I need to measure the timings.

# Instruction Scheduling

- A GPU has multiple SMs (SM = streaming multiprocessor)
- Each SM can be assigned multiple thread blocks.
- The hardware treats each thread-block as groups of 32 consecutive threads called warps.
  - ▶ For example, If a thread has `threadIdx.x = 0`, that means it is thread 0 in warp 0. If a thread has `threadIdx.x = 83`, that means it is thread 19 in warp 3.
- The number of blocks and threads that can be running at any time is determined by:
  - ▶ The number of SMs.
  - ▶ The number of blocks supported per SM.
  - ▶ The number of threads supported per block.
  - ▶ The number of registers per SP, and the number needed by each thread.
  - ▶ And other constraints described in Chapter 6.

# Instruction Execution

- Each SM tracks which warps of its blocks are ready to execute.
  - ▶ Earlier GPUs used a scoreboard.
  - ▶ More recent generations have the compiler handle this: instruction latencies are fixed.
    - ★ What about cache misses?
- On each clock-cycle, the SM chooses an instruction from a runnable warp and issues it.
  - ▶ Figure 4.10 (page 72 Kirk & Hwu) shows an SM with eight SPs, and issuing an instruction would take 4 consecutive cycles.
  - ▶ This makes sense: broadcasting the instruction to the SPs and using it 4 times further amortizes the overhead of managing instructions.
  - ▶ Not shown is that some GPUs can execute two, independent instructions per cycle – i.e. a floating point operation and a load or store.
  - ▶ These details are supposed to be largely invisible to the programmer.

# Lots of Blocks

- A kernel may consist of more blocks than can execute at the same time on the GPU.
- In this case, it runs as many blocks as it can.
- When a block completes, the GPU then starts a new block to replace it.
- I believe that each block runs to completion before a new block starts:
  - ▶ I couldn't find anything that suggested otherwise.
- The order in which blocks are executed is unspecified.



# Some examples

See [examples.cu.](#)

See [examples.erl.](#)

# Preview

## **March 11: The GPU Memory Model 1**

Reading: Kirk & Hwu, Chapter 5.

## **March 14: The GPU Memory Model 2**

Reading: Kirk & Hwu, Chapter 5.

## **March 16: GPU Performance 1**

Reading: Kirk & Hwu, Chapter 6.

## **March 18: GPU Performance 2**

Reading: Kirk & Hwu, Chapter 6.

## **March 21: Parallel Sorting**

Reading: TBD.

But of course, we'll adjust this as we go.