Objective

• To learn more about CUDA memory coalescing
• To learn more about barrier synchronization

Memory Coalescing

• After row-major linearization, adjacent threads in each warp access adjacent memory locations
  – Accesses for threads in the warp consolidated into one (or two) DRAM access (burst)
• The point is efficiency in using DRAM bursts
  – Current DRAM burst size is 64-128 bytes
  – Any distance between locations accessed by adjacent threads in a warp reduces DRAM efficiency
• When adjacent threads in a warp access identical locations, accesses for threads in the warp consolidated into one DRAM access
Four Important Access Patterns

Neighboring Columns (Coalesced)

Neighboring Threads access

Neighboring Rows (Not Coalesced)

Access direction in kernel code

Same Row (Coalesced, not efficient)

Access direction in kernel code

Coalesced since only one access will be done for all threads

Inefficient since most of the data returned will be wasted.
Same Column
(Coalesced, not efficient)

Access direction in kernel code

Load iteration 0
T₀ T₁ T₂ T₃

Load iteration 1
T₀ T₁ T₂ T₃

Coalesced since only one access will be done for all threads
Inefficient since half of the data returned will be wasted.

Enabling Coalescing with Shared Memory

Memory Coalescing - Summary

- After row-major linearization, adjacent threads in each warp access adjacent memory locations
  - Accesses for threads in the warp consolidated into one (or two) DRAM access (burst)
- The point is efficiency in using DRAM bursts
  - Current DRAM burst size is 64-128 bytes
  - Any distance between locations accessed by adjacent threads in a warp reduces DRAM efficiency
- When adjacent threads in a warp access identical locations, accesses for threads in the warp consolidated into one DRAM access

Stride Access Example (Stride = 1)

Adjacent threads in a warp access elements that are one way from each other.
Inefficient since half of the data returned will be wasted.
Barrier Synchronization

• An API function call in CUDA
  – __syncthreads()

• All threads in the same block must reach the __syncthreads() before any can move on

• Best used to coordinate tiled algorithms
  – To ensure that all elements of a tile are loaded
  – To ensure that all elements of a tile are consumed

The Reduction Steps

```c
// Stride is distance to the next value being accumulated into the threads mapped position
// in the partialSum[] array
for (unsigned int stride = 1; stride <= blockDim.x; stride *= 2)
{
__syncthreads();
if (t % stride == 0)
__syncthreads();  // Why is this a bad idea?
partialSum[2*t] += partialSum[2*t+stride];
}
```

READ CHAPTER 5