ECE408/CS483/CSE408 Spring 2019

Applied Parallel Programming

Lecture 22: Performance Considerations

1

Objective

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

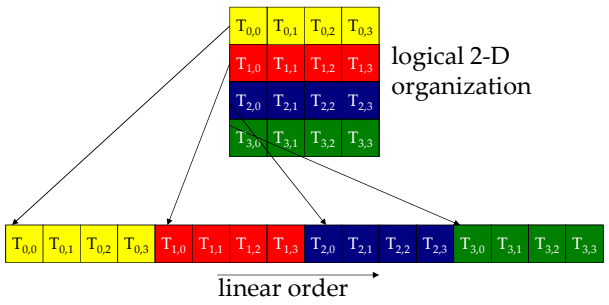
2

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

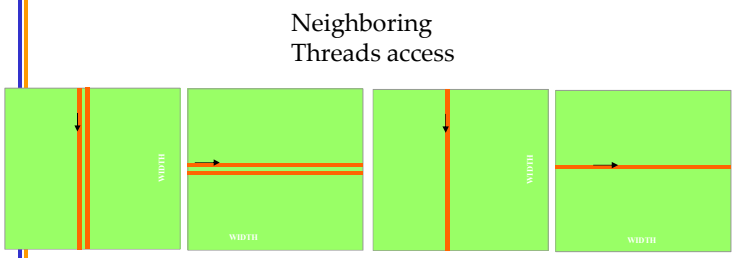
3

2D Threads in Linear Order (Row Major Layout)



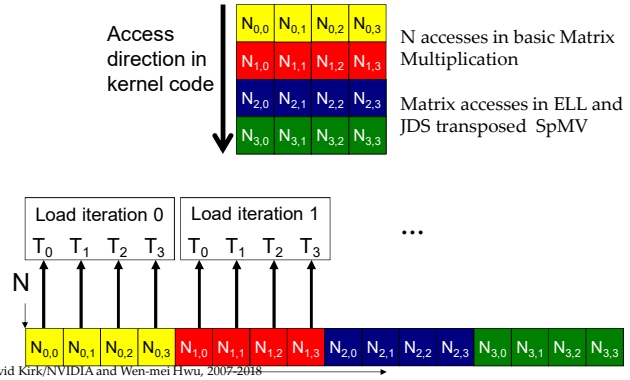
4

Four Important Access Patterns



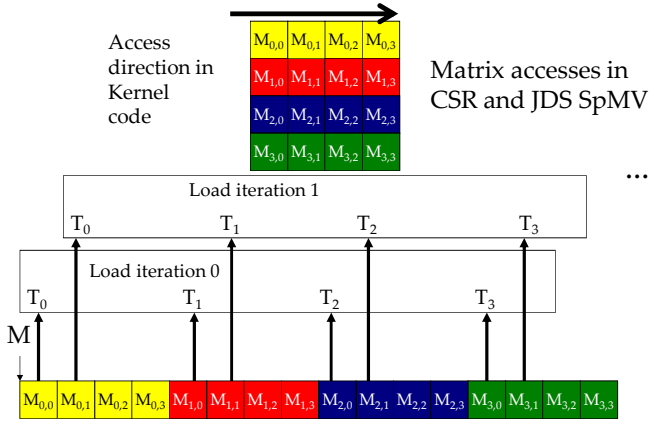
5

Neighboring Columns (Coalesced)



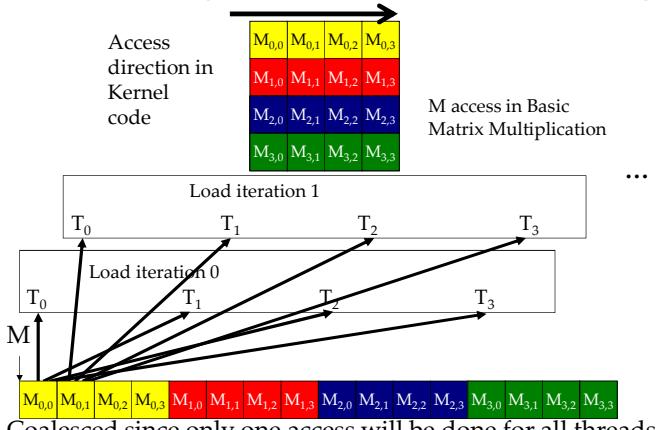
6

Neighboring Rows (Not Coalesced)



7

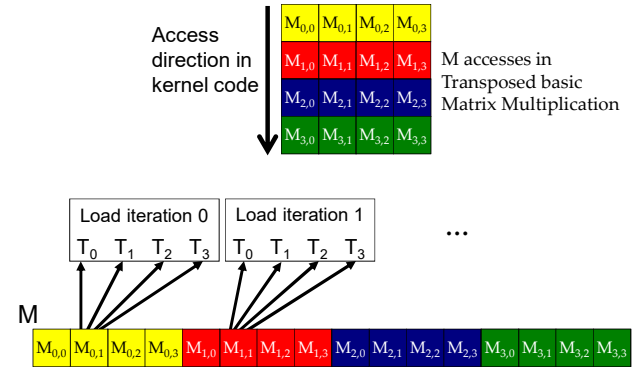
Same Row (Coalesced, not efficient)



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

8

Same Column (Coalesced, not efficient)

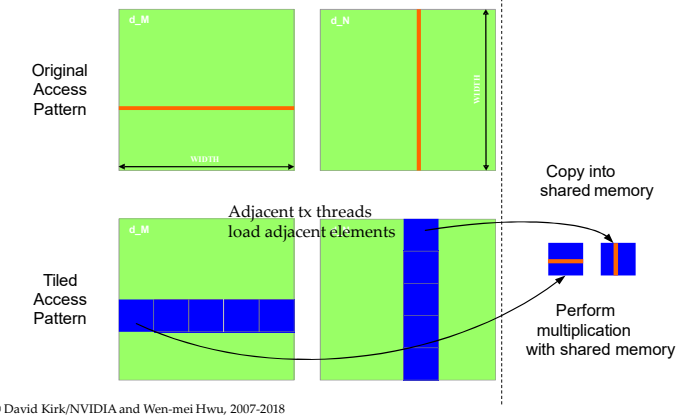


Coalesced since only one access will be done for all threads

Inefficient since most of the data returned will be wasted.

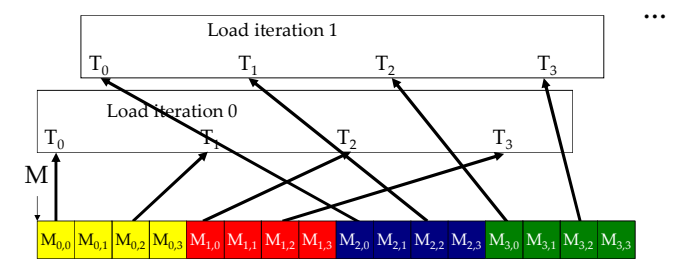
9

Enabling Coalescing with Shared Memory



10

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.

11

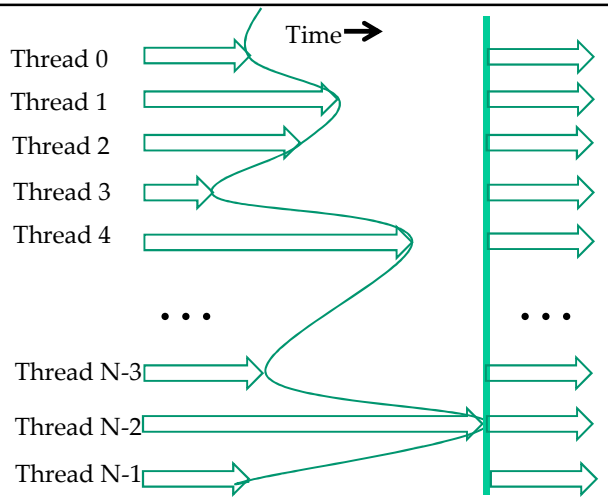
Memory Coalescing - Summary

- 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

12

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

```
// Stride is distance to the next value being
// accumulated into the threads mapped position
// in the partialSum[] aray
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