CUDA Shared Memory

Class: CS315 – Distributed Scalable Computing
Whitworth University, Instructor: Scott Griffith

Last Modified: 11/1/2018

Part 1: Matrix Multiply (100 pts)

For this learning module you are going to implement at least two kernels. One will be a naive, first pass at solving a matrix multiply using a traditional parallel approach. The other will utilize your GPUs shared memory space to see if there is a speed up in execution.

Both of these kernels will operate on double precision floating point numbers. **(+5 extra credit if you implement complex numbers, I could suggest cuComplex.h)**

First implementation just has to get the answer. There are a number of ways to get this working. A good starting point would be to make every thread map to one output cell, then grab the required data from the input matrices. This will be inherently non-optimal. Make sure you fully verify your answer.

Next you are going to implement the same algorithm but using shared memory and thread synchronization. For synchronization: you will utilize __syncthreads(); inside your kernel. For shared memory you will utilize a __shared__ tag before a memory initialization inside your kernel.

You will need to implement the same performance metrics as the previous LM using CUDA Events. Again a reference: https://devblogs.nvidia.com/how-implement-performance-metrics-cuda-cc/

(20 pts) Report your results of running these two kernels. Run them at different sizes and see if there is a difference (there should be! Shared memory is more efficient!).