Nikhil Gosike - ng1449
Parallel Computing
Professor Mohamed Zahran
May 3rd 2019

Lab 3 Report

Introduction

This is a program that uses CUDA to parallelize and find the max value in an array of randomly generated unsigned integers. This given input is X in the arguments to produce an array of X unsigned integers and fills the array with integers between 1 and X.

Compile Instructions

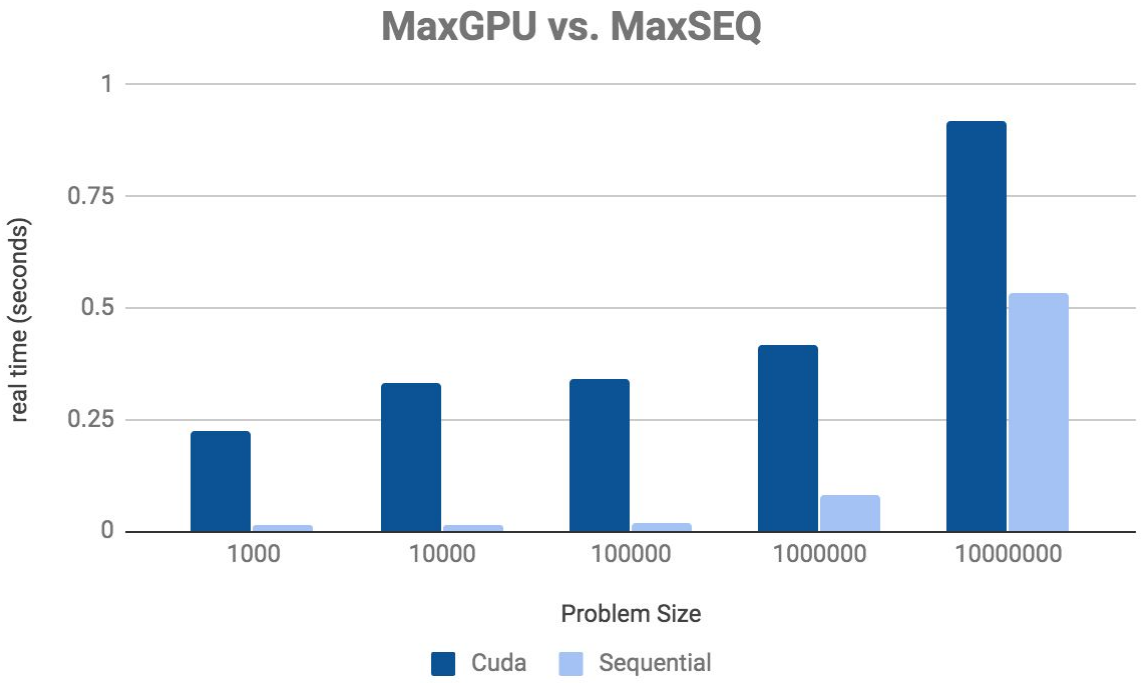
In order to compile this program, ssh to one of the cuda machines and run nvcc -arch=compute_35 -code=sm_35 maxgpu.cu in commandline.

Choices for Block and Grid size/dimensions

For the thread per block size, I chose 1024, as it was the maximum allowed by the 3.x architecture. In addition, it is a multiple of the warp size of 32 on these machines. For the grid size, i decided to calculate it as (size + threads per block - 1)/threads per block. This helped account for any extra items in the array that did not fit within the block, especially since the size was not evenly divisible by the threads per block.

Graph

Below is a graph to show the time taken between the sequential and Cuda code for X equals 1000, 10000, 100000, 1000000 (one million), and 10000000 (ten million).



Conclusion

From the graph, two trends can be clearly noticed. Both the Seq and GPU increase in time as the problem size increases, but the relative difference greatly decreases as we move from a problem size of a million to ten million. Of course, this points to the fact that a speedup may not be seen unless we have a greatly large problem size. The most outright in overhead that led to slower times on the GPU was the use of CudaMalloc and, even more importantly, cudaMemcpy. These are extremely expensive operations, so the parallel code is affected due the copying and allocation of memory from the host to the device. This is even more affected with the copying of the results back to the Host. There is also probably a contention of resources on the servers which are not observable.