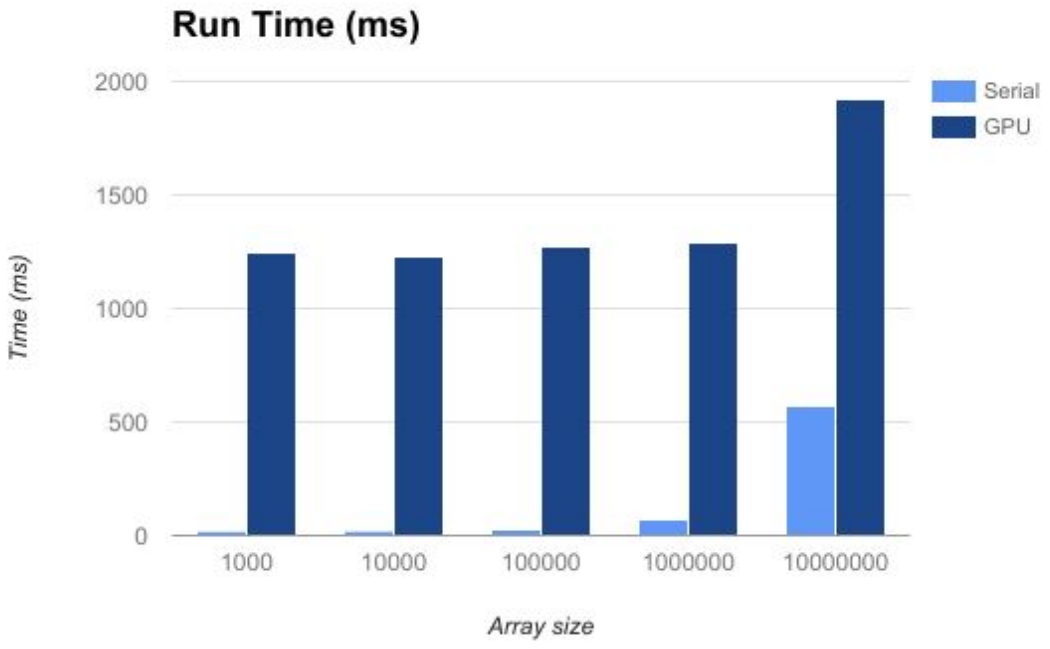
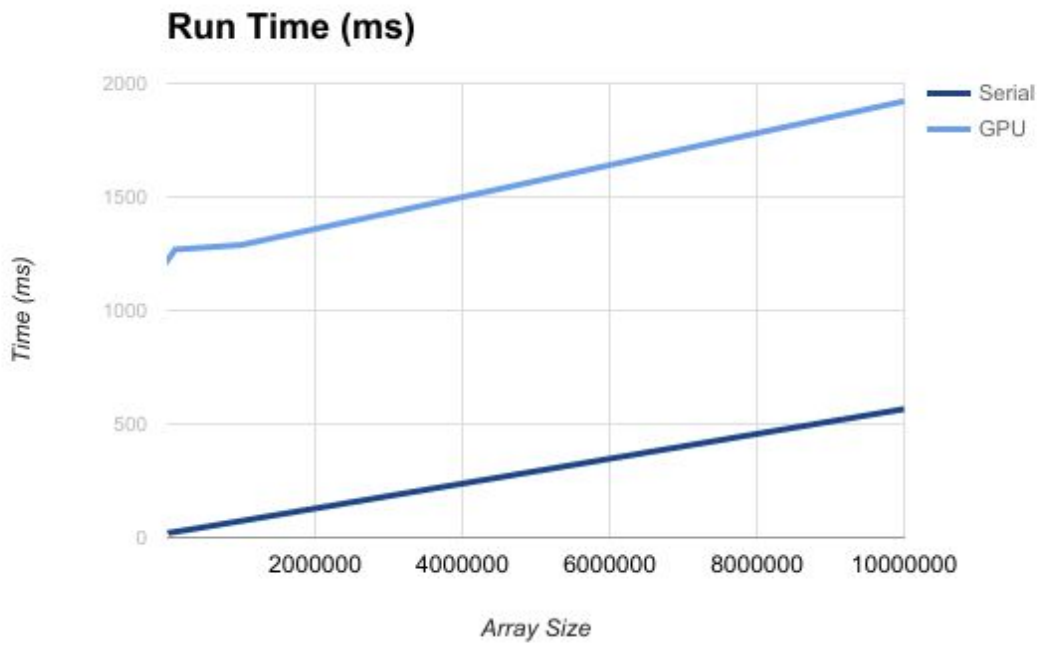
Michael Laucella
Lab3
Parallel processing

1.) The dimensions used were block sizes of 1024 threads which was the max threads per block on my GPU as well as cuda5. The blocks were one dimensional as the access patterns are of a single vector/array and there is no reason to add more dimensionality to the block. The grid consists of ceil(size/blocksize) blocks, this way as problem sizes increase so do the number of blocks used in the grid. The grid was also one dimensional as the whole problem deals with a single dimension vector and therefore does not require a 2d approach.

2.) nvcc -o maxgpu maxgpu.cu

3.)





4.) the graph shows the run times of both the serial and the gpu max array search. At first glance the GPU performed much slower than the CPU but the important thing to note here is the percent increase in the run time from 1,000 to 10,000,000 for GPU there was a 29% decrease in speed whereas for the GPU a 1.5% speed decrease is observed. Given a large enough problem size the sequential version will undoubtedly surpass the GPU run time. Many reasons for slow downs in the GPU also include the basic overhead in allocations, instruction fetching, moving memory and synchronization (the atomic max for result operates once per block, the more blocks the more atomic swaps that occur). Another interesting point of slowdown is at 10,000,000 prior to all the slow downs that occur are less than 1%, some possible reasons are scheduling the GPU run on was a GeForce GT 650M which contains 2 sms as the size increases so do the number of blocks, this means more scheduling and possibly having to wait for blocks to finish before more can run in this case there are approx 9,765 blocks.