Name: Jingfei Shao

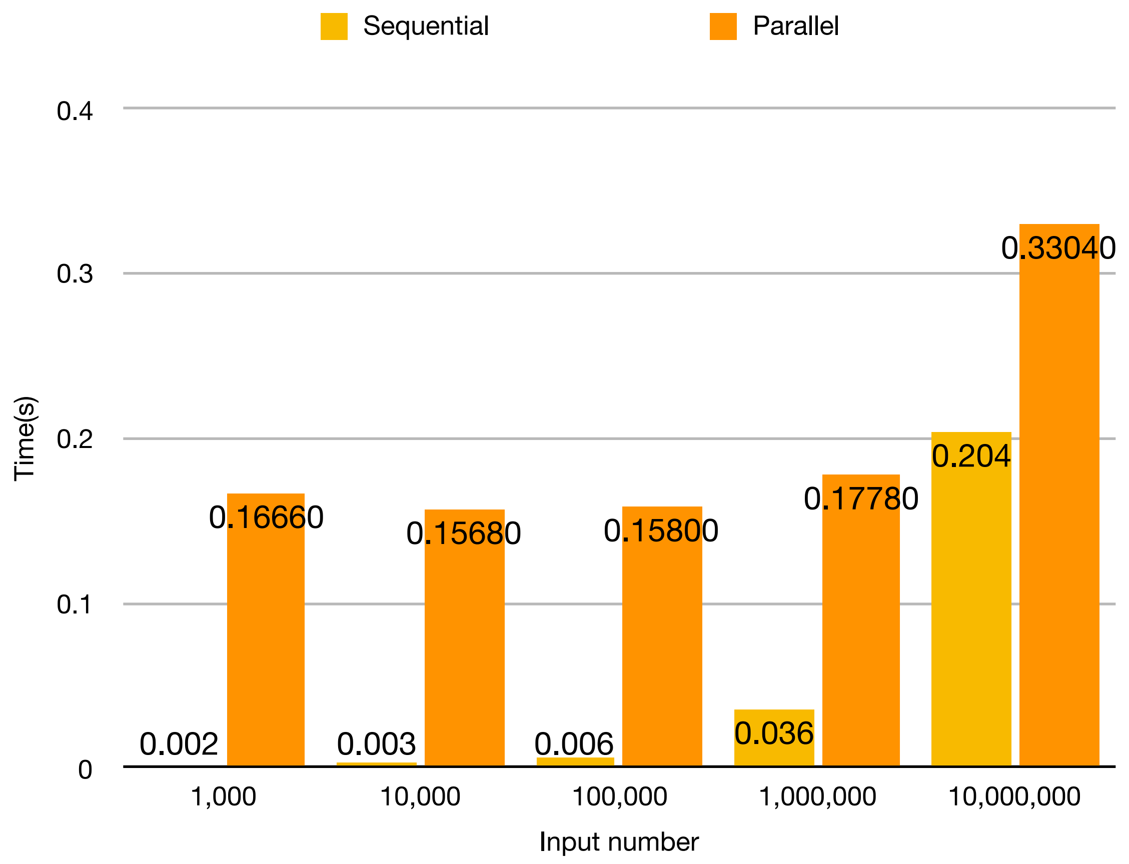
NetID: js9950

I used CUDA1 to run the program.

1. I used 1024 threads per block because it is the upper limit of the number of threads per block for 3.x architectures, and at the same time I want to maximize the number of threads that are running. So for the block size, it is 1024. And for the grid size, I make it be the ceil(number input)/1024. Also, both the grid and block are 1-d. The work is evenly assigned to each block. For the last block, the number of threads might be smaller than 1024, which is the input % 1024.

2. Compile with: nvcc -o maxgpu maxgpu.cu

3.



4.

I implement the reduction algorithm for this assignment. So basically it works this way: let each block finds their max value, then store this value in the array at a certain position, then finally find the final max value of all the max values found by different blocks.

The runtime of parallel code is affected negatively by copying the arrays from host memory to the GPU device. In my program it takes 0.152s on average. Yet, for the sequential program, the time taken is far less than this. In other words, no matter what algorithm we use, when the problem is not big enough, the parallel program can never be faster than the sequential program. In addition, there is a synchronization point in the for loop, which harms the performance as well. Yet, this synchronization is necessary because of racing condition. Otherwise, the program will proceed even though the current for loop is not finished for certain threads. In addition, the branch condition will also harm the performance. Due to the fact that GPU program will not proceed because all the threads need to do the same computation. Also, since I make use of shared memory for each block, the time of accessing global memory is reduced. So in a word, it is nearly impossible for the parallel program to beat the sequential one due to the limitation of problem size.