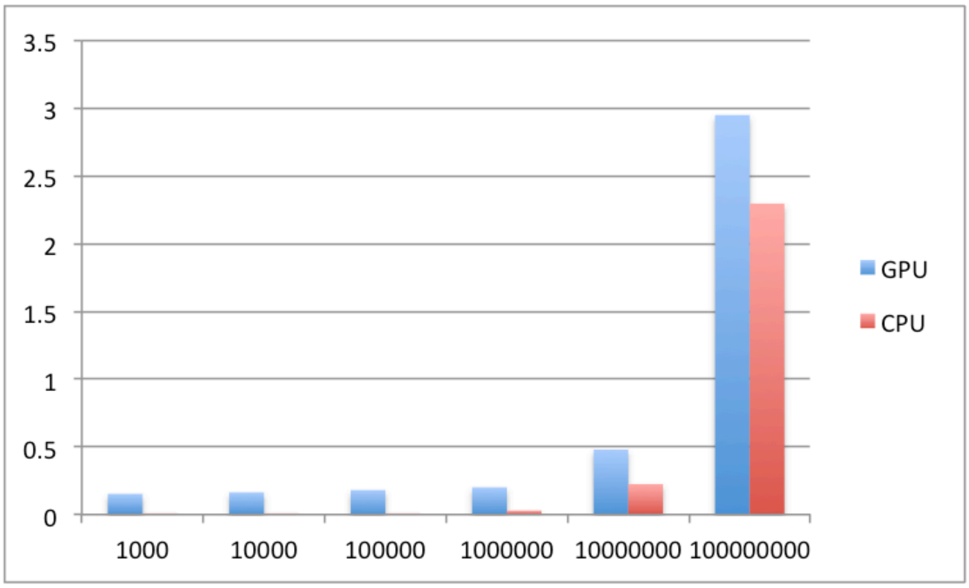
Parallel Computing Lab 3 Report
Name: Ran (Elaine) Ang   NetID: ra1695

1. Block size: 1 dimensional block, 1024 threads per block
   Grid size: 1 dimensional grid of size: (int) ceil((float) sz / 1024), where "sz" is the input size, and ceil is the C ceiling function.
   Justify: I run my code on CUDA 1 and CUDA 5, where the maximum block size is (1024, 1024, 64), maximum grid size is (2147483647,65535,65535), and maximum #threads per block is 1024 (getting from cudaGetDeviceProperties). Since we are dealing with one dimension array, a natural thing to do is to have also one dimension block and grid, so that indexing the array would be simpler. Grid size is computed at run time depending on the input (array size).

2. Compile with: nvcc maxseq.cu –o maxseq –arch=sm_35

3.



4. In the above graph, the obvious overall trend is that the execution time increases with an increase of input size, because we have more data to process. However, it's worth noticing that although CPU performs better than GPU on all the testing input, the difference of their performance gets smaller when the input size grows. Namely, if we measure the relative performance increase of GPU with regard to CPU by (time(CPU) – time(GPU))/time(GPU), we could see that this number, though being negative, increases with the increase of input size. This happens because executing on GPU needs to copy data from main memory to GPU's global memory. This takes relatively a long time, and the performance decrease brought by such time overhead cannot be eliminated with the current amount of data we test.

Another reason that we do not see a performance increase may be that I used atomic operations when updating the largest data, and all threads within a single block are actually executed sequentially during finding the local_max_num. Although I think such sequentially execution here has approximately the same effect of using atomicMax. Since I kept a current largest number and keep updating it while preventing potential race condition, the algorithms here are actually not very parallel friendly.

We could give a rough prediction that if we keep increasing the input size, we will have better performance with using GPU than CPU although we may need to be more careful with memory management in that situation.