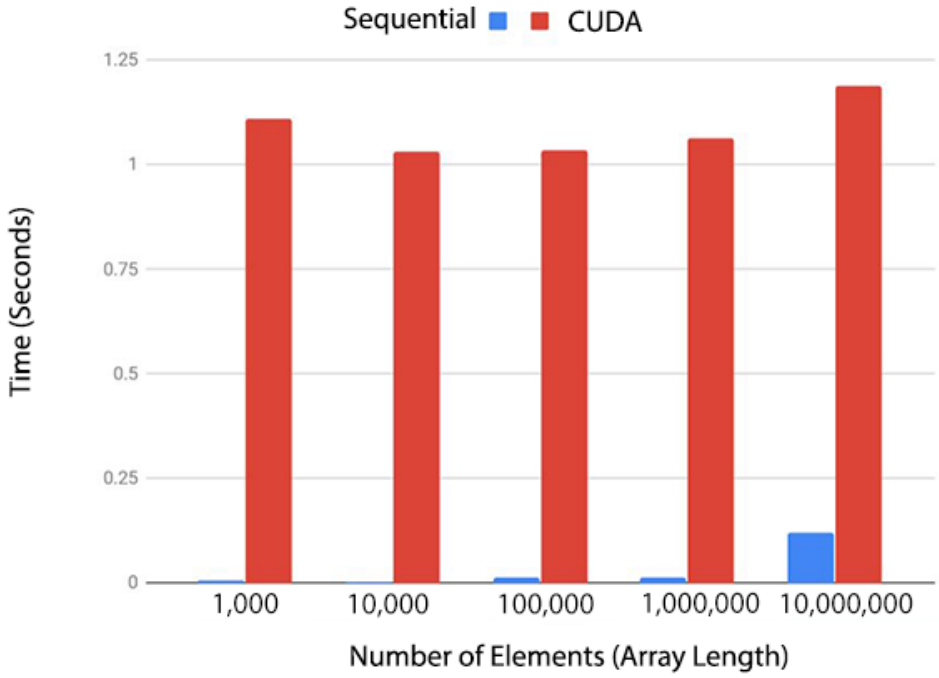
Samira Mantri

Lab 3

cuda4

1. For the block size/dimension I used 1024 because according to the gpu I utilized, 1024 is the maximum threads that can be used in each block. For the grid size/dimension, I used the following equation to determine it: size/1024. The size represents the number of elements that need to be checked in the array divided by the maximum number of threads that can be used in each block.
2. nvcc maxgpu.cu
3. 

4. As you can see, the sequential code performed far better than the cuda code. However, one noticeable difference is how the sequential code loses performance at a steady rate whereas the cuda tends to fluctuate more. One reason the sequential likely did better is because data did not have to

be copied from the cpu to the gpu and back. Its steady loss of performance can be attributed to the increase in problem size. With a larger data set, the time taken to iterate through it will increase. With regards to the cuda code, CudaMemcpy takes a lot of time because the memory has to travel along the bus from the host to the device. With the sequential code, the needed variables are already located within the cpu so it can execute faster. The data size may also have been too small to determine if the cuda code could outperform the sequential. One reason there is a dip in the cuda code is because when the initial number of blocks is larger than 1 (meaning the data size is larger), I rerun the kernel with the block maxes. I continue to do this until I reduce the number of blocks to 1 since data in different blocks cannot synchronize. Every time I gather new maxes I have to reallocate and copy memory to hold them which might be why the performance improves before getting worse. Basically the more the problem size is reduced by the kernel calls, the more work that has to be done.