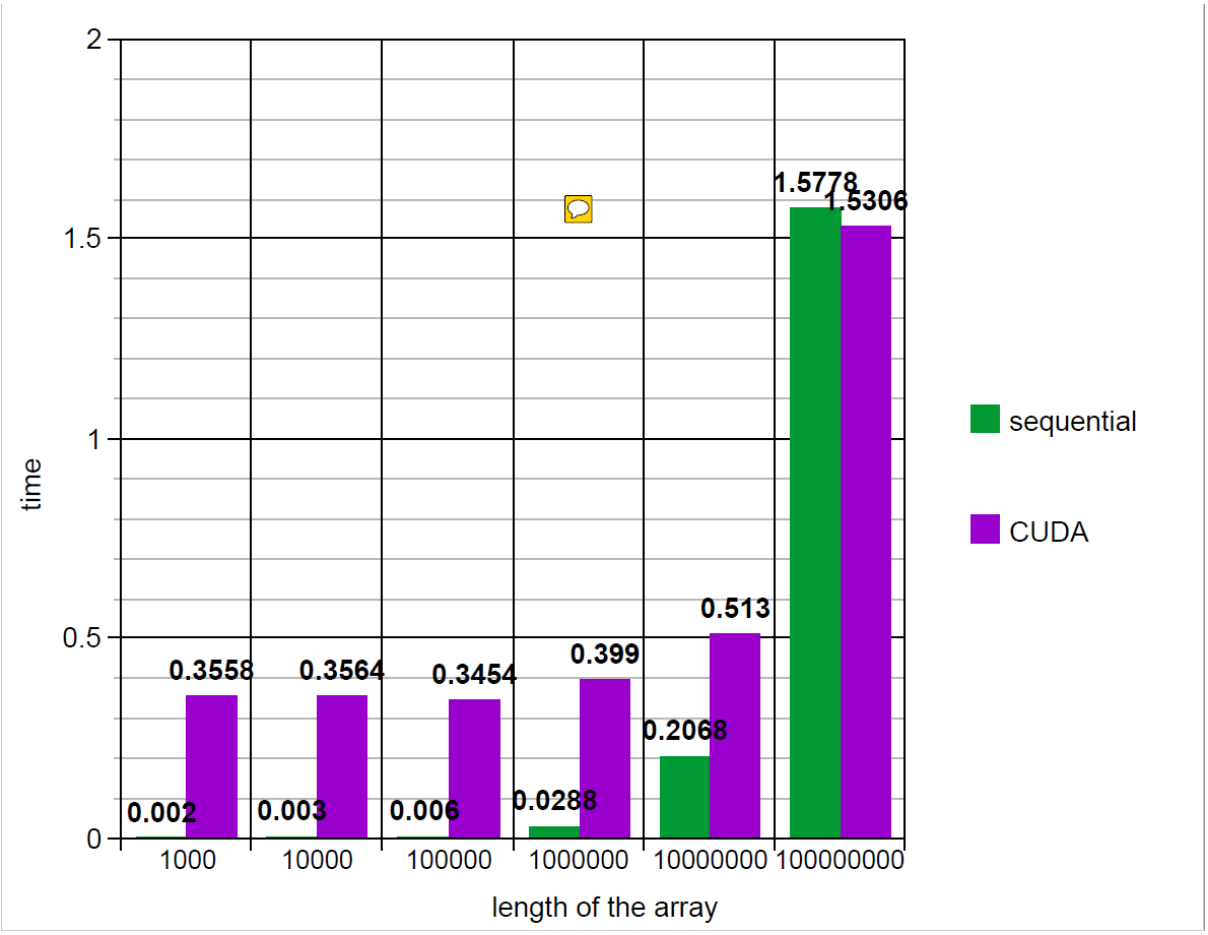
CUDA Machine Used: Cuda1

1. The size for the blocks and grids I chose is 512 threads per block and 1block per grid. The reason I chose 512 threads per block is because that is the maximum amount of thread allowed per block on the machine and there are no restrictions on the number of registers based on the number of elements of the array given. By making maximum use of threads, my code can get the maximum parallelism. The waste of resource from idle threads created from the last block with the remainder is small compared to the performance gained from utilizing all the threads. Also 512 is divisible by 32 which means it is an efficient number in the sense of warp. The reason why I didn't pick (number of element/ 512) blocks per grid to maximize the amount of resource used while not having excess idle blocks waste resource by switching in and out is because I tried my code with both (number of element/ 512) blocks and just 1 block and not much performance was gained from increasing the number of blocks and sometimes had worse performance. I think this is happening because my code is embarrassingly parallel and cannot use the increased number of blocks and only make the extra block waste resource.

2. nvcc maxgpu.cu ("I know about using nvcc maxgpu.cu –arch=compute_35 -code=compute_35,sm_35 for number of elements too big for the graphic card memory such as 100000000 but I couldn't test it out because CUDA machine was not working).




4. The graph shows CUDA having worse performance compared to the sequential code from 1000 to 1000000. This is because the data size is not large enough to create a performance gain enough to cover the performance lost created from the overhead (kernel execution, memory copy from gpu to cpu and memory copy from cpu to gpu). This fact can also be seen from the fact that the data sets from 1000 to 1000000 has not much of a time increase because unlike sequential codes that takes much more time with the computation (shown by the increase of time by x250 from 1000 to 1000000) computation for CUDA takes the least time; Since everything except from the amount of computation from 1000 to 1000000 has not changed the time change from increased amount of computation is minimal. The change in CUDA code comes from 10000000. The graphic card's memory restricts my code and makes it have to call multiple kernel functions. This change in the code makes a big spike on the time taken compared to the previous time increases. CUDA performance finally catches up with the sequential because the increased amount of computation count is less than the increased amount of kernel call and the increase in execution time due to computation is more than the increase in execution time from the overhead. As we increase the data size this gap will increase more and more (CUDA with better execution time).