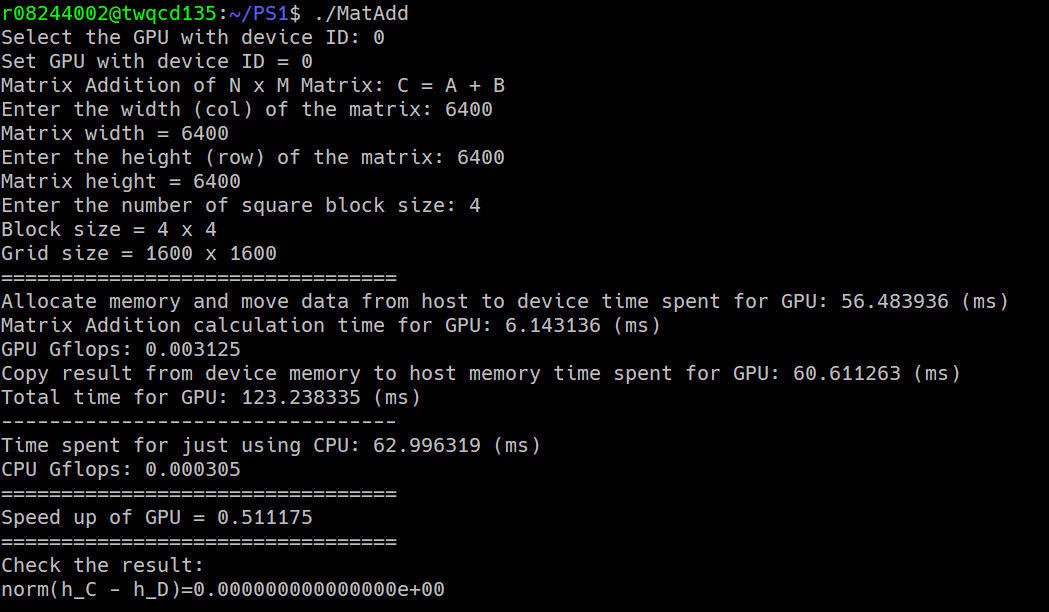
**CUDA Parallel Programming Problem Set 1**

R08244002 Shin-Rong, Tsai

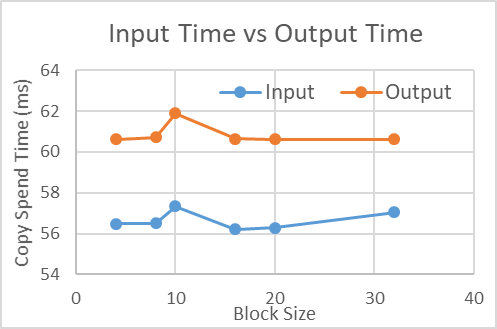
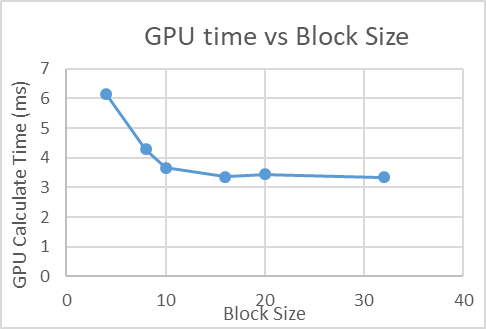
* **Matrix Addition**

1. **Results**

Slightly change the code *vecAdd.cu* given in course, and saved it in file *MatAdd.cu*. The CUDA Runtime API I used here to copy memory between host and device is cudaMemcpy, since I view this 2D matrix as 1D. Compiled and run on the node, we get:

Execute and input the demand block size individually, and collect all the data, we get:

1. **Discussion**
2. The optimal block size would be around 16, since it reaches a saturation after the block size more than 16. It is probably because that the streaming multiprocessor number that GTX 970 has is 13. Which means dividing the matrix with small block size leads to a bigger grid size and it somewhat needs more serial time to finish compute all the small blocks. It is inefficient!
3. Copying data from host to device is faster than copying data from device to host. And the performance of the host + device in runtime affects the most (which is obvious), since both curves look alike.



1. If we consider the compute time of CPU and GPU only, GPU is way much faster than CPU. It’s about 10 to 20 times faster than CPU. But when considering the input and output of data, GPU does not speed up. It is because that matrix addition is not that intense, so it is not worth it to do the computation on GPU.

Otherwise, if CPU is dealing with other stuff which slows down the speed, it might be a good idea to move the computation to GPU.

