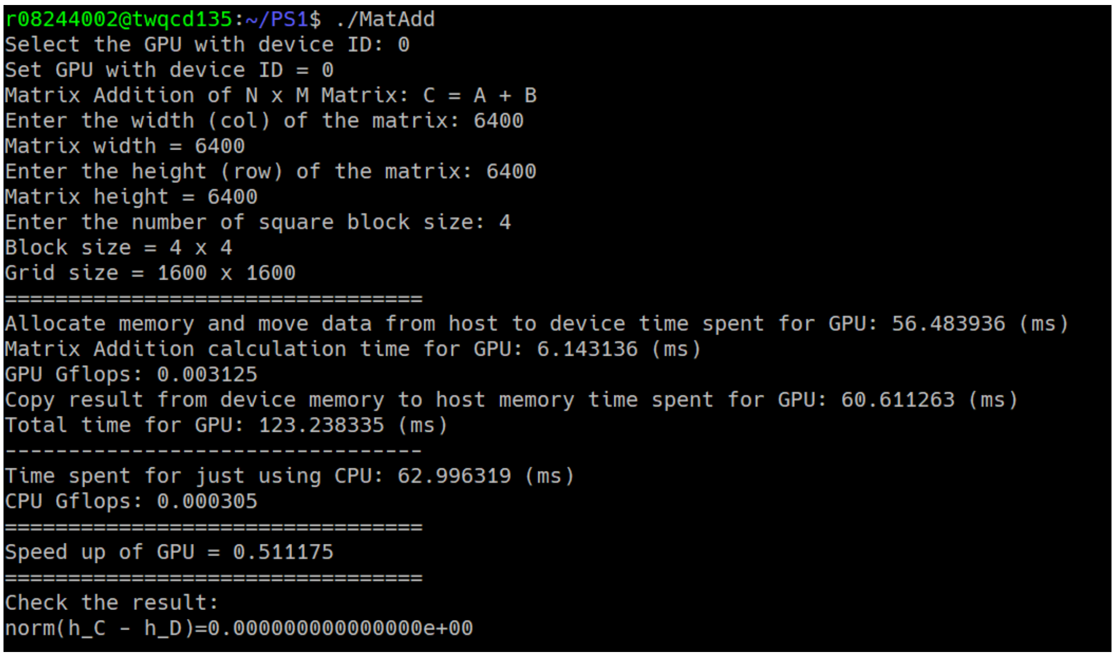
CUDA Parallel Programming Problem Set 1

R08244002 Shin-Rong, Tsai

• Matrix Addition

I. 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:



```
r08244002@twqcd135:~/PS1$ ./MatAdd
Select the GPU with device ID: 0
Set GPU with device ID = 0
Matrix Addition of N x M Matrix: C = A + B
Enter the width (col) of the matrix: 6400
Matrix width = 6400
Enter the height (row) of the matrix: 6400
Matrix height = 6400
Enter the number of square block size: 4
Block size = 4 x 4
Grid size = 1600 x 1600
=================================
Allocate memory and move data from host to device time spent for GPU: 56.483936 (ms)
Matrix Addition calculation time for GPU: 6.143136 (ms)
GPU Gflops: 0.003125
Copy result from device memory to host memory time spent for GPU: 60.611263 (ms)
Total time for GPU: 123.238335 (ms)
---------------------------------
Time spent for just using CPU: 62.996319 (ms)
CPU Gflops: 0.000305
=================================
Speed up of GPU = 0.511175
=================================
Check the result:
norm(h_C - h_D)=0.000000000000000e+00
```

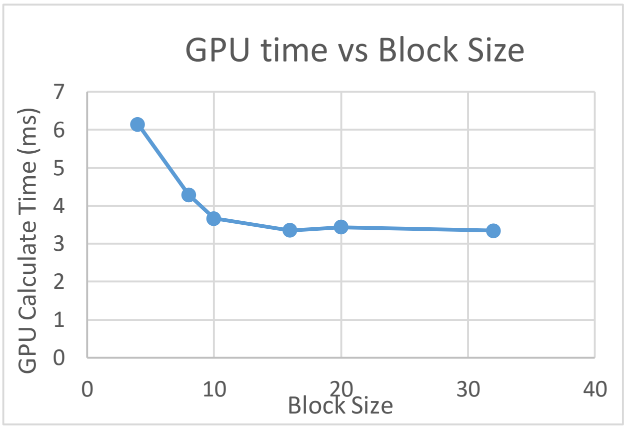
Execute and input the demand block size individually, and collect all the data, we get:

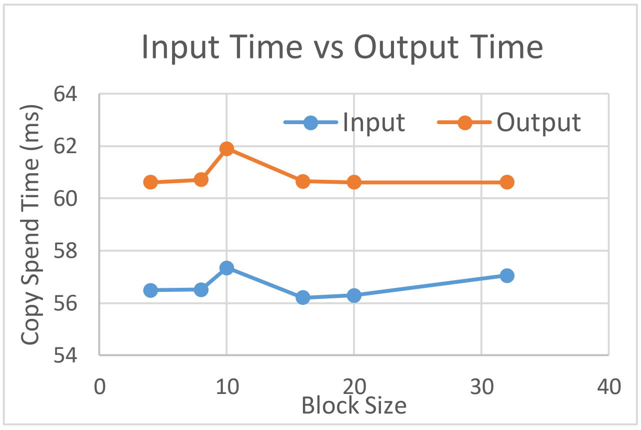
Block Size	Grid Size	Input	GPU	GPU gflops	Output	GPU total	CPU	CPU gflops	SpeedUP
4	1,600	56.483936	6.143136	0.003125	60.611263	123.238335	62.996319	0.000305	0.511175
8	800	56.504192	4.289472	0.004476	60.711552	121.505219	62.688511	0.000306	0.515933
10	640	57.348927	3.669248	0.005233	61.900127	122.918304	63.131554	0.000304	0.513606
16	400	56.207359	3.353792	0.005725	60.64592	120.207069	62.822754	0.000306	0.522621
20	320	56.289536	3.436896	0.005586	60.607712	120.334145	63.274689	0.000303	0.525825
32	200	57.048286	3.345888	0.005738	60.615425	121.009598	63.175102	0.000304	0.522067

II. Discussion

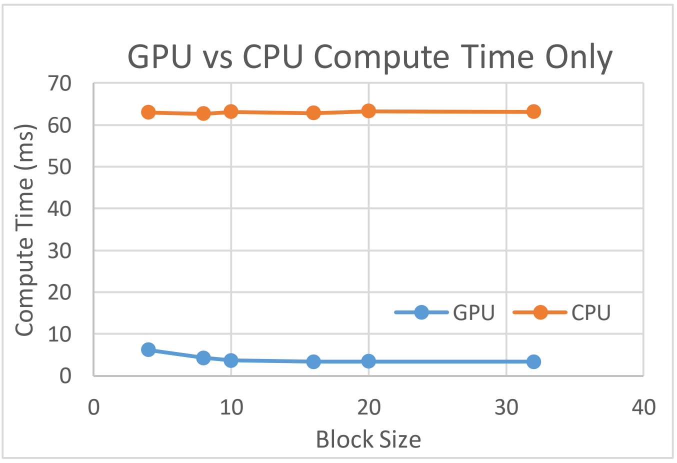
1. The optimal block size would be around 16, since it reaches a saturation after the block size more than 16. It is probably because that the streaming multiprocessor number that GTX 970 has is 13. Which means dividing the matrix with small block size leads to a bigger grid size and it somewhat needs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

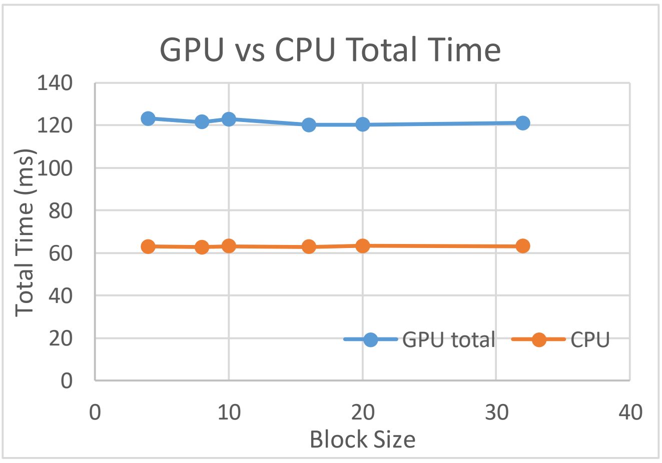
affects the most (which is obvious), since both curves look alike.





3. 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.


GPU vs CPU Compute Time Only
Compute Time (ms)
Block Size
GPU
CPU


GPU vs CPU Total Time
Total Time (ms)
Block Size
GPU total
CPU