BLG562E – Parallel Computing for GPUs using CUDA

Homework #2

Mustafa İzzet Muştu - 504211564

Development and Test Environment

Operating system on the host machine is Windows 11 22H2 and nvcc version is V12.1.66. Hardware information as follows: AMD Ryzen 7 5800X, 32 GB 3400 MHz, MSI RTX 3060 12 GB.

1. Performance of Vector Addition on CPU (sequential code)

Important Note: When I set N=2048, run time always was 0.00 seconds so I set N=2048*2048 for the test.
As it can be seen from the table, average run time is 0.01 seconds.

Trial/Run Time (s)	CPU
1	0.01
2	0.01
3	0.01
4	0.01
5	0.01

Table 1

2. Performance of Vector Addition on GPU

a) In this part of step 2, implement the vector addition kernel and the host code to run the GPU kernel with only a single block. Compile your code with nvcc compiler and run to report the execution time. Again you need to make several runs and report on the average execution time. Your code should also include a function to verify the results generated by GPU. You can compare the results generated on the CPU and the results generated on the GPU.
Maximum number of threads per block supported by CUDA is 1024 so I set the thread number as 1024 and validated the results by filling the validate function and got 2048*2048-1024 mismatch between the CPU result and the GPU result. This is because we could calculate only a small portion of addition with 1 block. Several run times of the GPU code can be seen in the Table 2.

Trial/Run Time (s)	GPU
1	0.01
2	0.01
3	0.01
4	0.01
5	0.01

Table 2

b) In this part of step 2, implement the vector addition kernel and the host code to run the GPU kernel with blocks having a single thread. Compile your code and run to report the execution time.

I set the number of blocks as N which is equal to 2048*2048. I did not get any mismatch error between the CPU result and the GPU result. Run times are given in Table 3.

Trial/Run Time (s)	GPU
1	0.01
2	0.01
3	0.01
4	0.01
5	0.01

Table 3

c) In this part of step 2, change the vector size to 2096; implement the vector addition kernel and the host code to run the GPU kernel with blocks having 128 threads. Compile your code and run to report the execution time. Does your code generate correct results?

I changed N=2096*2096 got correct results because 2096*2096/128=34322 which is an integer. If I set N=2096, I would have got 48 mismatch errors because 2048=16*128 and there is 48 remainder in 2096/128. Run times reduced and we can say that using multiple blocks with multiple threads per block is better than using many number of blocks with 1 thread per block. Results can be seen in Table 4.

Trial/Run Time (s)	GPU
1	0.00
2	0.00
3	0.00
4	0.00
5	0.00

Table 4

d) Now, modify your code to run with a specified block size without generating incorrect results. Compile your code and run to report the execution time. Make sure your GPU kernel generates correct results.

I changed the number of threads per block as 512 and got result below. For this specific task, I could not observe any difference. I assume that run time would decrease for more complex tasks.

Trial/Run Time (s)	GPU
1	0.00
2	0.00
3	0.00
4	0.00
5	0.00

Table 5

3. Performance Study of Matrix—Matrix Addition

I used arrays to represent matrix and treated the certain number of elements in the array as rows. I set N as 2048, block dimension as 32x32x1 and grid dimension as 64x64x1. Results can be seen in Table 6.

Trial/Run Time (s)	CPU	GPU
1	0.01	0.00
2	0.01	0.00
3	0.01	0.00
4	0.01	0.00
5	0.01	0.00

Table 6

4. Scalability Study (Optional, Extra Points)

Run times of CPU and GPU for different N values with the same code of Question 3 are shown in Figure 1. Multiple of 16384 requires data type change that's why I stopped at this value. As we can see from the Figure 1, when we increase the matrix dimension, the speedup increases much more. For the N=16384, speedup is **53**.

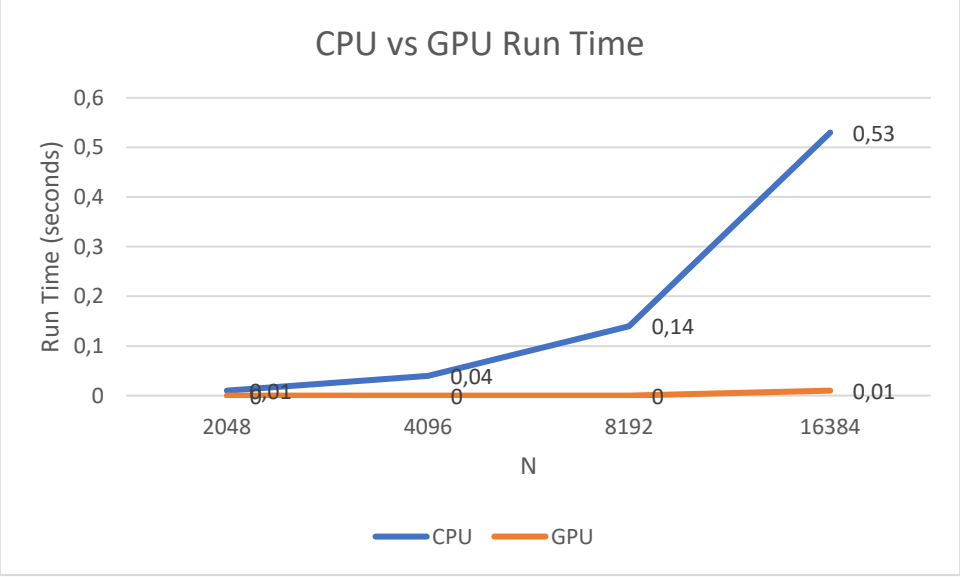

Figure 1