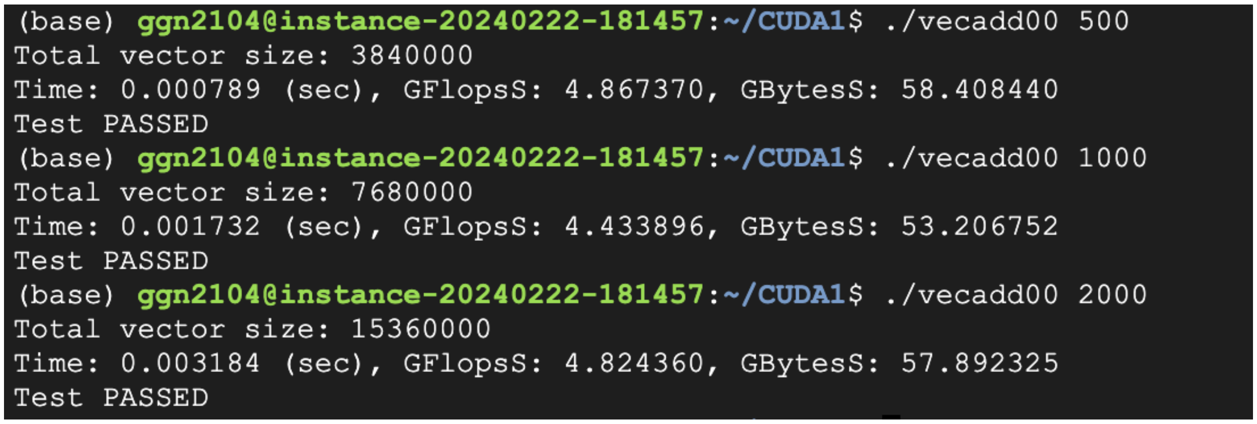
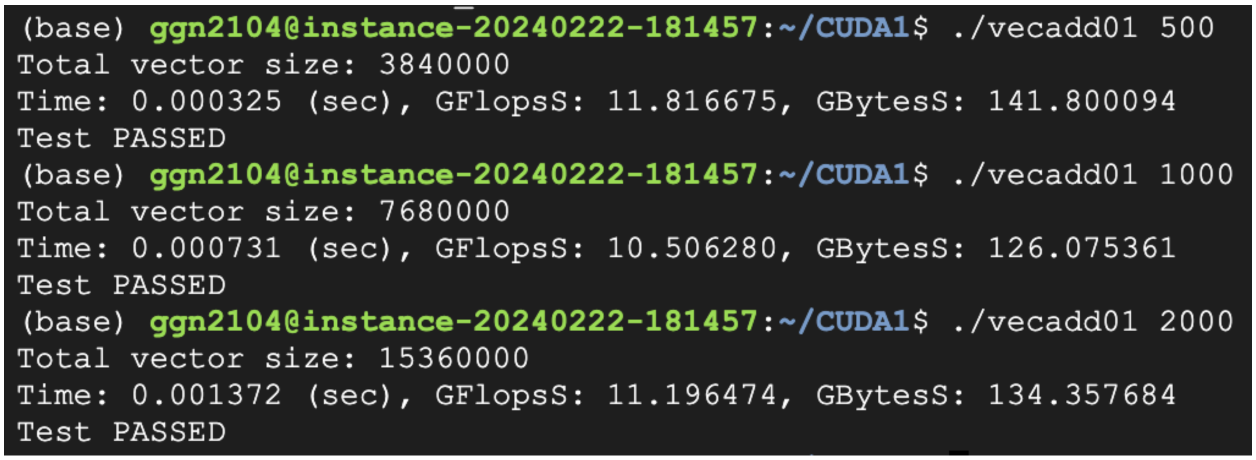
Part A

Q1)



```
(base) ggn2104@instance-20240222-181457:~/CUDA1$ ./vecadd00 500
Total vector size: 3840000
Time: 0.000789 (sec), GFlopsS: 4.867370, GBytesS: 58.408440
Test PASSED
(base) ggn2104@instance-20240222-181457:~/CUDA1$ ./vecadd00 1000
Total vector size: 7680000
Time: 0.001732 (sec), GFlopsS: 4.433896, GBytesS: 53.206752
Test PASSED
(base) ggn2104@instance-20240222-181457:~/CUDA1$ ./vecadd00 2000
Total vector size: 15360000
Time: 0.003184 (sec), GFlopsS: 4.824360, GBytesS: 57.892325
Test PASSED
```

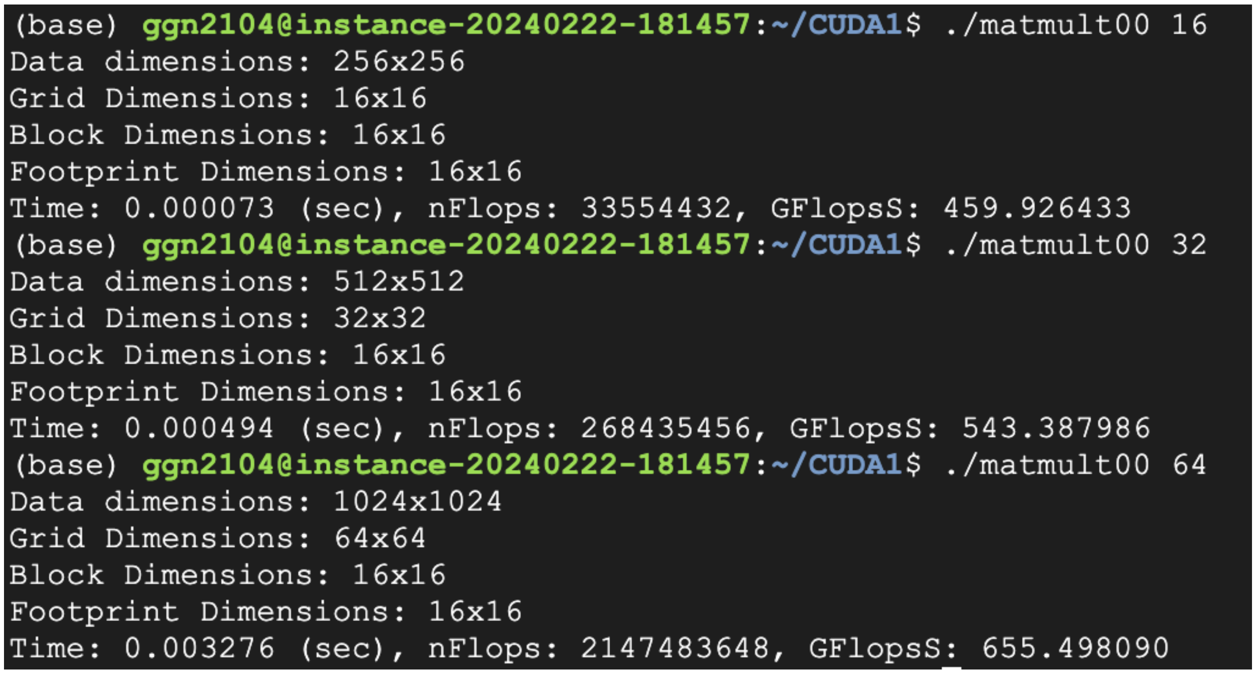
Q2)



```
(base) ggn2104@instance-20240222-181457:~/CUDA1$ ./vecadd01 500
Total vector size: 3840000
Time: 0.000325 (sec), GFlopsS: 11.816675, GBytesS: 141.800094
Test PASSED
(base) ggn2104@instance-20240222-181457:~/CUDA1$ ./vecadd01 1000
Total vector size: 7680000
Time: 0.000731 (sec), GFlopsS: 10.506280, GBytesS: 126.075361
Test PASSED
(base) ggn2104@instance-20240222-181457:~/CUDA1$ ./vecadd01 2000
Total vector size: 15360000
Time: 0.001372 (sec), GFlopsS: 11.196474, GBytesS: 134.357684
Test PASSED
```

Observing the results of running vector addition with coalesced memory accesses, we see that it is significantly faster than running vector addition with non-coalesced memory accesses. Specifically, there are over 2x improvements with the latency (sec) and throughput (GFlopsS, GBytesS).

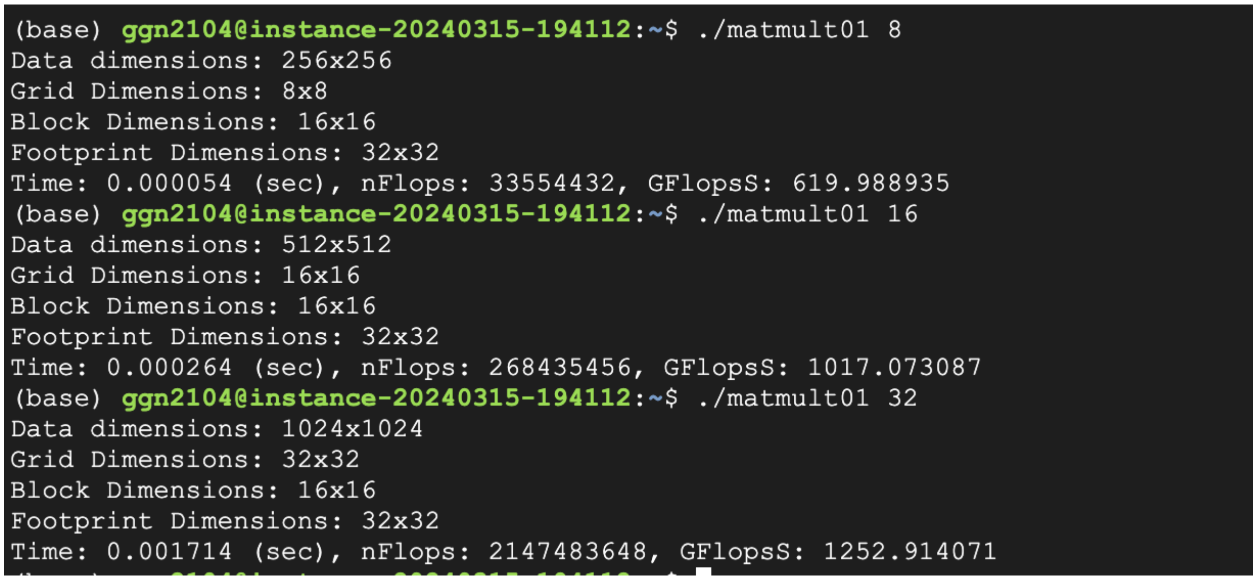
Q3)



```
(base) ggn2104@instance-20240222-181457:~/CUDA1$ ./matmult00 16
Data dimensions: 256x256
Grid Dimensions: 16x16
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.000073 (sec), nFlops: 33554432, GFlopsS: 459.926433
(base) ggn2104@instance-20240222-181457:~/CUDA1$ ./matmult00 32
Data dimensions: 512x512
Grid Dimensions: 32x32
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.000494 (sec), nFlops: 268435456, GFlopsS: 543.387986
(base) ggn2104@instance-20240222-181457:~/CUDA1$ ./matmult00 64
Data dimensions: 1024x1024
Grid Dimensions: 64x64
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.003276 (sec), nFlops: 2147483648, GFlopsS: 655.498090
```

As the size of the matrix (data dimensions) increases along with the number of blocks in the grid (grid dimensions), we see that the throughput (GFlopsS) increases as well. This shows that as we increase the grid size and have more threads in the grid, the GPU is taking advantage of thread-level parallelism. Therefore, more threads are available to perform matrix multiplication and execute in parallel, thereby improving throughput and GPU utilization.

Q4)



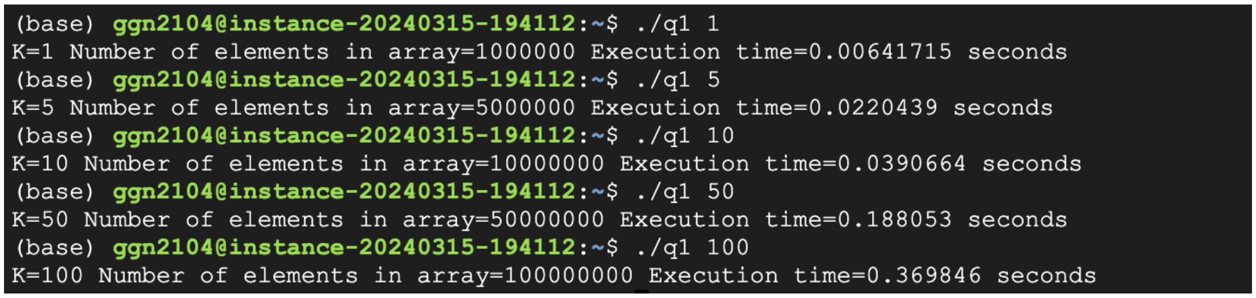
```
(base) ggn2104@instance-20240315-194112:~$ ./matmult01 8
Data dimensions: 256x256
Grid Dimensions: 8x8
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.000054 (sec), nFlops: 33554432, GFlopsS: 619.988935
(base) ggn2104@instance-20240315-194112:~$ ./matmult01 16
Data dimensions: 512x512
Grid Dimensions: 16x16
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.000264 (sec), nFlops: 268435456, GFlopsS: 1017.073087
(base) ggn2104@instance-20240315-194112:~$ ./matmult01 32
Data dimensions: 1024x1024
Grid Dimensions: 32x32
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.001714 (sec), nFlops: 2147483648, GFlopsS: 1252.914071
```

As the size of the matrix (data dimensions) increases along with the number of blocks in the grid (grid dimensions), we see that the throughput (GFlopsS) increases as well. This shows that as we increase the grid size and have more threads in the grid, the GPU is taking advantage of thread-level parallelism. Additionally, when each thread computes 4 values in the resulting C block rather than 1 value the performance increases. Specifically, there is approximately 1.22 speedup when completing multiplication with 256x256 matrices, 1.8 speedup when completing multiplication with 512x512 matrices, and 1.86 speedup when completing multiplication with 1024x1024 matrices.

Q5) When trying to achieve good performance in CUDA, two important concepts to follow are memory coalescing and tiling. In memory coalescing, all threads in a warp access consecutive global memory locations. As a result, memory accesses will be coalesced into a single request for consecutive DRAM locations, allowing the DRAM to deliver data as a burst thereby improving performance. Since the shared memory is smaller but faster than the global memory, we can partition the data into subsets known as tiles and load the tiles into shared memory. As a result, we access data from the shared memory to complete the kernel operation before loading another tile into shared memory and repeating the computation. This can also increase performance.

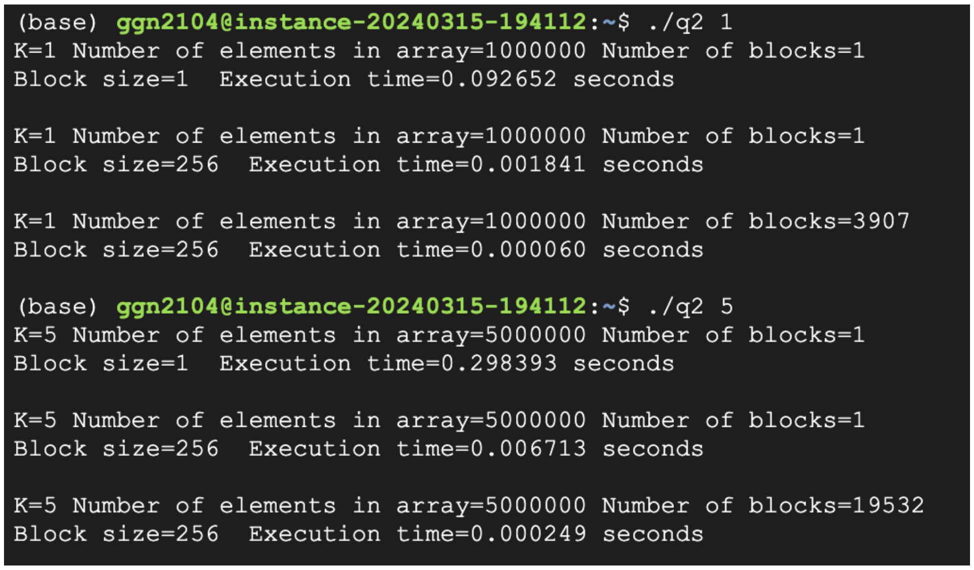
Part B

Q1)

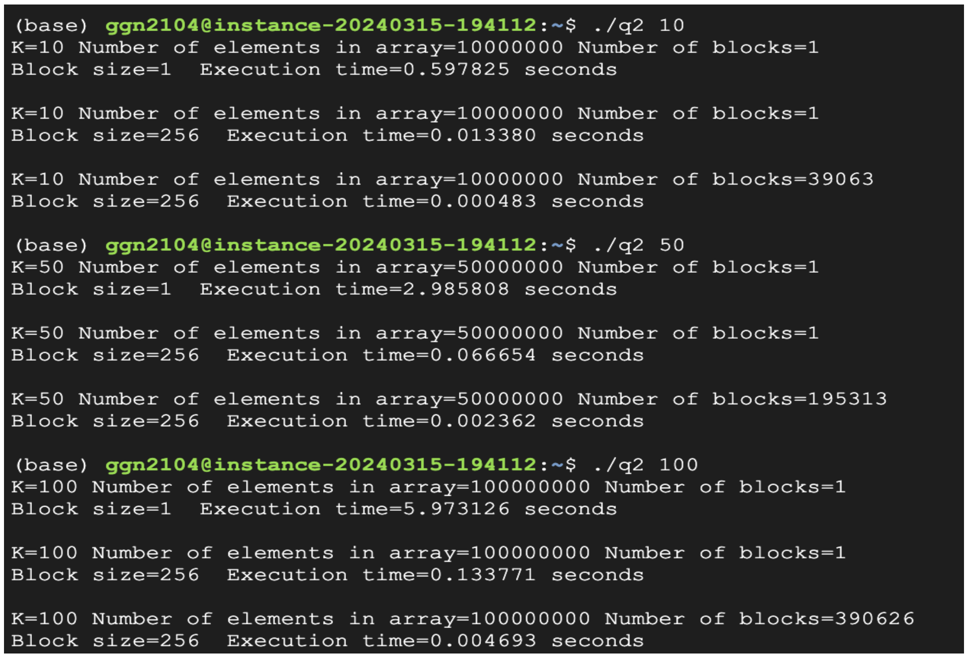


```
(base) ggn2104@instance-20240315-194112:~$ ./q1 1
K=1 Number of elements in array=1000000 Execution time=0.00641715 seconds
(base) ggn2104@instance-20240315-194112:~$ ./q1 5
K=5 Number of elements in array=5000000 Execution time=0.0220439 seconds
(base) ggn2104@instance-20240315-194112:~$ ./q1 10
K=10 Number of elements in array=10000000 Execution time=0.0390664 seconds
(base) ggn2104@instance-20240315-194112:~$ ./q1 50
K=50 Number of elements in array=50000000 Execution time=0.188053 seconds
(base) ggn2104@instance-20240315-194112:~$ ./q1 100
K=100 Number of elements in array=100000000 Execution time=0.369846 seconds
```

Q2)

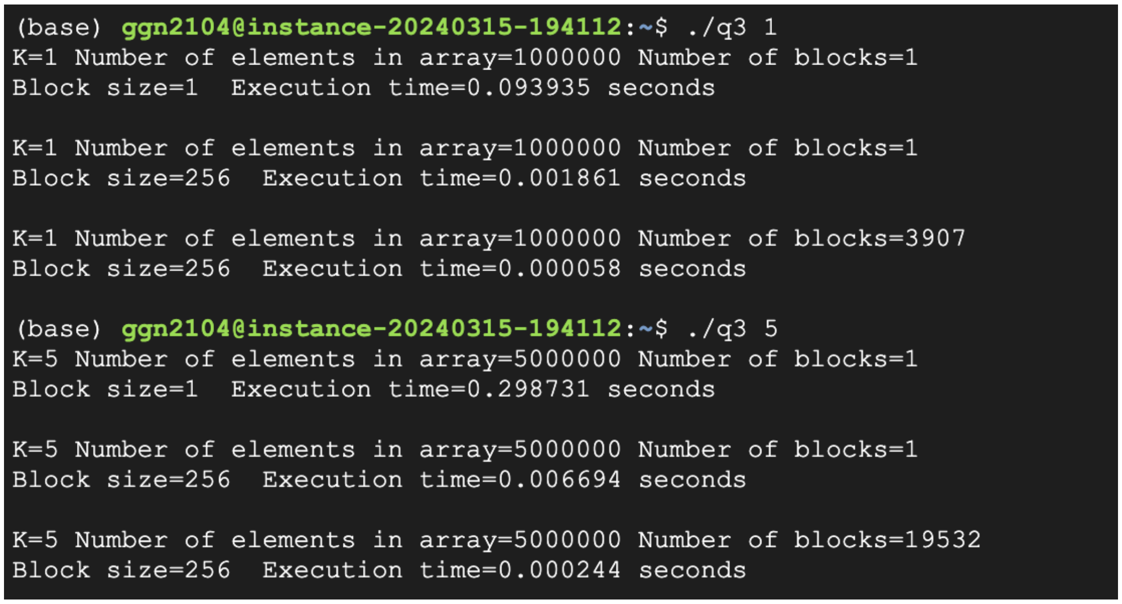


```
(base) ggn2104@instance-20240315-194112:~$ ./q2 1
K=1 Number of elements in array=1000000 Number of blocks=1
Block size=1  Execution time=0.092652 seconds

K=1 Number of elements in array=1000000 Number of blocks=1
Block size=256  Execution time=0.001841 seconds

K=1 Number of elements in array=1000000 Number of blocks=3907
Block size=256  Execution time=0.000060 seconds

(base) ggn2104@instance-20240315-194112:~$ ./q2 5
K=5 Number of elements in array=5000000 Number of blocks=1
Block size=1  Execution time=0.298393 seconds

K=5 Number of elements in array=5000000 Number of blocks=1
Block size=256  Execution time=0.006713 seconds

K=5 Number of elements in array=5000000 Number of blocks=19532
Block size=256  Execution time=0.000249 seconds
```

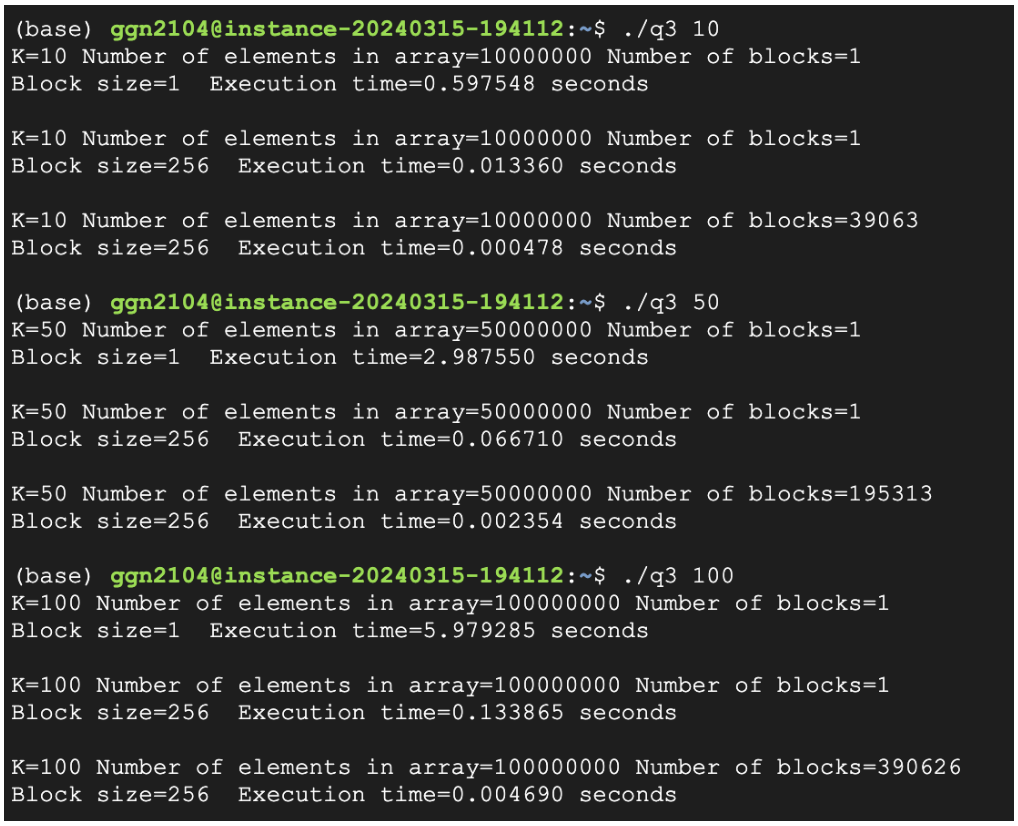


```
(base) ggn2104@instance-20240315-194112:~$ ./q2 10
K=10 Number of elements in array=10000000 Number of blocks=1
Block size=1  Execution time=0.597825 seconds

K=10 Number of elements in array=10000000 Number of blocks=1
Block size=256  Execution time=0.013380 seconds

K=10 Number of elements in array=10000000 Number of blocks=39063
Block size=256  Execution time=0.000483 seconds

(base) ggn2104@instance-20240315-194112:~$ ./q2 50
K=50 Number of elements in array=50000000 Number of blocks=1
Block size=1  Execution time=2.985808 seconds

K=50 Number of elements in array=50000000 Number of blocks=1
Block size=256  Execution time=0.066654 seconds

K=50 Number of elements in array=50000000 Number of blocks=195313
Block size=256  Execution time=0.002362 seconds

(base) ggn2104@instance-20240315-194112:~$ ./q2 100
K=100 Number of elements in array=100000000 Number of blocks=1
Block size=1  Execution time=5.973126 seconds

K=100 Number of elements in array=100000000 Number of blocks=1
Block size=256  Execution time=0.133771 seconds

K=100 Number of elements in array=100000000 Number of blocks=390626
Block size=256  Execution time=0.004693 seconds
```

Q3)

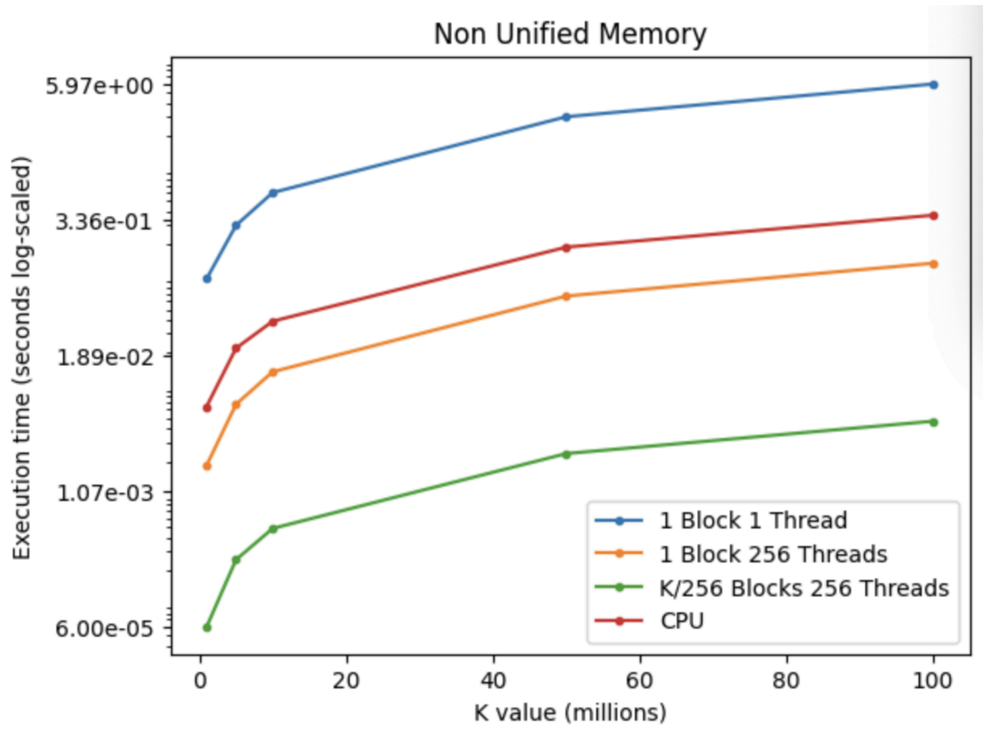


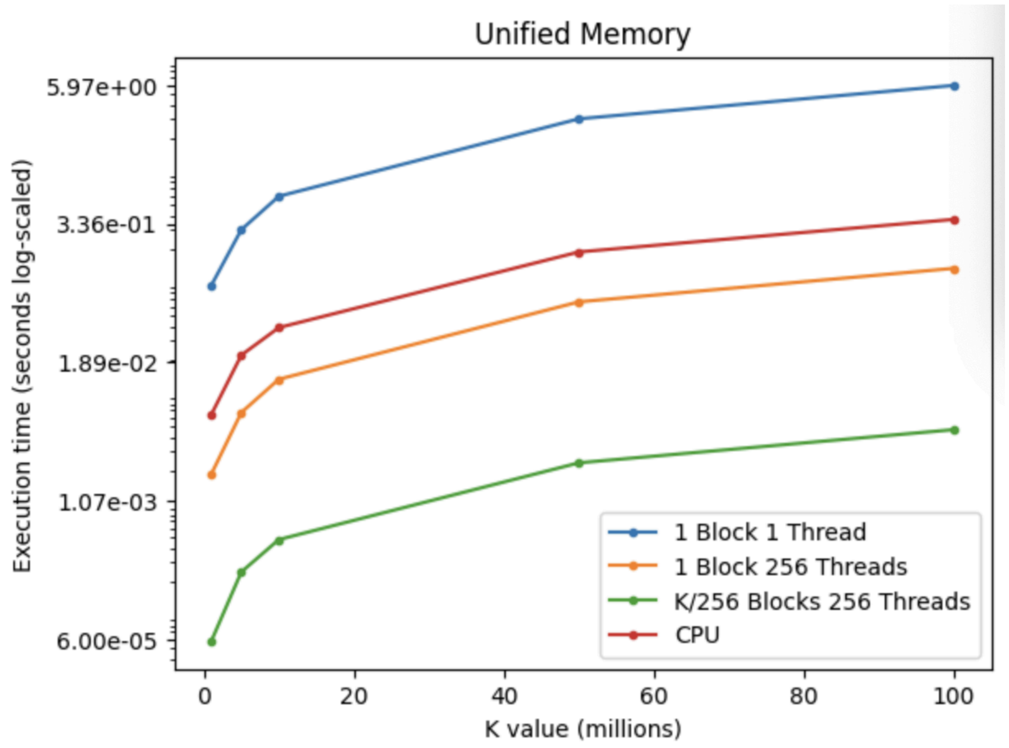
```
(base) ggn2104@instance-20240315-194112:~$ ./q3 1
K=1 Number of elements in array=1000000 Number of blocks=1
Block size=1  Execution time=0.093935 seconds

K=1 Number of elements in array=1000000 Number of blocks=1
Block size=256  Execution time=0.001861 seconds

K=1 Number of elements in array=1000000 Number of blocks=3907
Block size=256  Execution time=0.000058 seconds

(base) ggn2104@instance-20240315-194112:~$ ./q3 5
K=5 Number of elements in array=5000000 Number of blocks=1
Block size=1  Execution time=0.298731 seconds

K=5 Number of elements in array=5000000 Number of blocks=1
Block size=256  Execution time=0.006694 seconds

K=5 Number of elements in array=5000000 Number of blocks=19532
Block size=256  Execution time=0.000244 seconds
```



```
(base) ggn2104@instance-20240315-194112:~$ ./q3 10
K=10 Number of elements in array=10000000 Number of blocks=1
Block size=1  Execution time=0.597548 seconds

K=10 Number of elements in array=10000000 Number of blocks=1
Block size=256  Execution time=0.013360 seconds

K=10 Number of elements in array=10000000 Number of blocks=39063
Block size=256  Execution time=0.000478 seconds

(base) ggn2104@instance-20240315-194112:~$ ./q3 50
K=50 Number of elements in array=50000000 Number of blocks=1
Block size=1  Execution time=2.987550 seconds

K=50 Number of elements in array=50000000 Number of blocks=1
Block size=256  Execution time=0.066710 seconds

K=50 Number of elements in array=50000000 Number of blocks=195313
Block size=256  Execution time=0.002354 seconds

(base) ggn2104@instance-20240315-194112:~$ ./q3 100
K=100 Number of elements in array=100000000 Number of blocks=1
Block size=1  Execution time=5.979285 seconds

K=100 Number of elements in array=100000000 Number of blocks=1
Block size=256  Execution time=0.133865 seconds

K=100 Number of elements in array=100000000 Number of blocks=390626
Block size=256  Execution time=0.004690 seconds
```

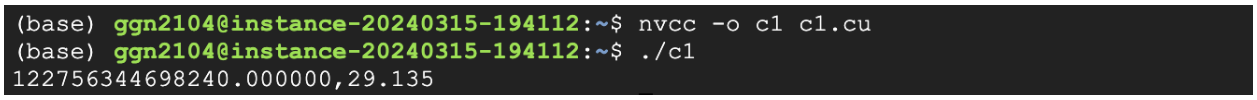
Q4)


Non Unified Memory
5.97e+00
3.36e-01
1.89e-02
1.07e-03
6.00e-05
Execution time (seconds log-scaled)
0
20
40
60
80
100
K value (millions)
1 Block 1 Thread
1 Block 256 Threads
K/256 Blocks 256 Threads
CPU


Unified Memory
5.97e+00
3.36e-01
1.89e-02
1.07e-03
6.00e-05
Execution time (seconds log-scaled)
0
20
40
60
80
100
K value (millions)
1 Block 1 Thread
1 Block 256 Threads
K/256 Blocks 256 Threads
CPU

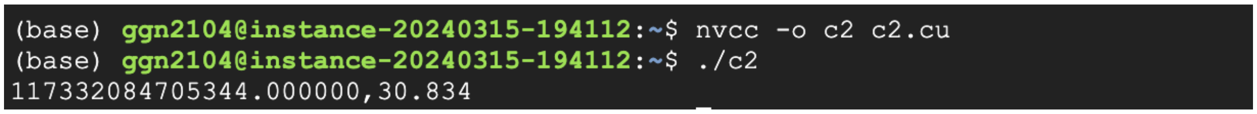
Part C

C1)



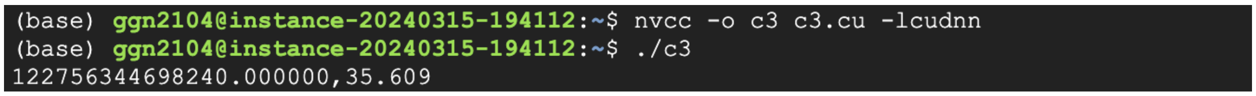
```
(base) ggn2104@instance-20240315-194112:~$ nvcc -o c1 c1.cu
(base) ggn2104@instance-20240315-194112:~$ ./c1
122756344698240.000000,29.135
```

C2)



```
(base) ggn2104@instance-20240315-194112:~$ nvcc -o c2 c2.cu
(base) ggn2104@instance-20240315-194112:~$ ./c2
117332084705344.000000,30.834
```

C3)



```
(base) ggn2104@instance-20240315-194112:~$ nvcc -o c3 c3.cu -lcudnn
(base) ggn2104@instance-20240315-194112:~$ ./c3
122756344698240.000000,35.609
```