High Performance Machine Learning
Lab 4

Ankit Rajvanshi
ar7996

Part – A

Problem-1:

Vector addition without memory coalescing.

Execution time for various size of vector:

Values per thread	Vector size	Time (sec)
500	3840000	0.000372
1000	7680000	0.000729
2000	15360000	0.001468

We can see that with increasing number of values per thread, execution time for addition also increases.

Vector addition with memory coalescing.

Execution time for various size of vector:

Values per thread	Vector size	Time (sec)
500	3840000	0.000254
1000	7680000	0.000503
2000	15360000	0.001001

We can see that with increasing number of values per thread, execution time for addition also increases but for each value, execution time is less than the time in non-memory coalescing version.

Values per thread	SpeedUp
500	1.46
1000	1.45
2000	1.47

Problem-2: Shared CUDA Matrix Multiply

Q3: Profiling of original kernel

Parameter	Size of Matrix	Time (sec)	GFlopsS
256	4096x4096	0.094794	1449.869219
512	8192x8192	0.689928	1593.661281
1024	16384x16384	5.583812	1575.284595

We can see that with increasing size of the matrix, execution time increase but overall throughput remains same i.e. there is not much effect on Gflops/sec.

While keeping parameter mixed i.e. 256 and if we change block size, we can see that as block size increases, execution time increases but throughput also increases.

```
[ar7996@gr031 PartA]$ ./matmult00 256
Data dimensions: 2048x2048
Grid Dimensions: 256x256
Block Dimensions: 8x8
Footprint Dimensions: 8x8
Time: 0.019169 (sec), nFlops: 17179869184, GFlopsS: 896.238732
```

```
[ar7996@gr031 PartA]$ ./matmult00 256
Data dimensions: 4096x4096
Grid Dimensions: 256x256
Block Dimensions: 16x16
Footprint Dimensions: 16x16
Time: 0.081955 (sec), nFlops: 137438953472, GFlopsS: 1677.006005
```

```
[ar7996@gr031 PartA]$ ./matmult00 256
Data dimensions: 8192x8192
Grid Dimensions: 256x256
Block Dimensions: 32x32
Footprint Dimensions: 32x32
Time: 0.554462 (sec), nFlops: 1099511627776, GFlopsS: 1983.024472
```

Q4: Profiling of modified kernel

Parameter	Size of Matrix	Time (sec)	GFlopsS
256	8192x8192	0.304622	3609.430254
512	16384x16384	2.430987	3618.321880
1024	32768x32768	19.492514	3610.039452

Again, we can see that with increasing size of the matrix, execution time increase but overall throughput remains same i.e. there is not much effect on Gflops/sec.

Again, while keeping parameter mixed i.e. 256 and if we change block size, we can see that as block size increases, execution time increases but throughput also increases.

```
[ar7996@gr031 PartA]$ ./matmult01 256
Data dimensions: 4096x4096
Grid Dimensions: 256x256
Block Dimensions: 8x8
Footprint Dimensions: 16x16
Time: 0.073992 (sec), nFlops: 137438953472, GFlopsS: 1857.483614
```

```
[ar7996@gr031 PartA]$ ./matmult01 256
Data dimensions: 8192x8192
Grid Dimensions: 256x256
Block Dimensions: 16x16
Footprint Dimensions: 32x32
Time: 0.306574 (sec), nFlops: 1099511627776, GFlopsS: 3586.446492
```

But when we compare original and modified kernel, we can significant improvement in execution time and throughput for same size of matrix.

Size of Matrix	SpeedUp
8192x8192	2.26
16384x16384	2.30

Q5: Can you formulate any rules of thumb for obtaining good performance when using CUDA?
From above analysis, we can say that,

1. Favorable memory access pattern is most important. It helps in increasing throughput of the program.
2. If favorable memory access pattern is not possible, always use shared memory concept (like tiling) to decrease memory access time.
3. Increase block size until throughput no longer increases.
4. Manual unrolling the loop also increases the performance.

Part – B: CUDA Unified Memory

Q1: Profiling of C++ program (CPU)

K (in million)	Time (sec)
1	0.00558615
5	0.020762
10	0.0404131
50	0.20291
100	0.408763

Q2: Profiling of Non-Unified Memory program

K (in million)	Blocks	Threads	Time (sec)
1	1	1	0.0745721
5	1	1	0.289676
10	1	1	0.579509
50	1	1	2.90074
100	1	1	5.79799

K (in million)	Blocks	Threads	Time (sec)
1	1	256	0.00125813
5	1	256	0.00621796
10	1	256	0.0125179
50	1	256	0.0621629
100	1	256	0.124147

K (in million)	Blocks	Threads	Time (sec)
1	Var	256	0.0000288486
5	Var	256	0.000124216
10	Var	256	0.000242949
50	Var	256	0.00118899
100	Var	256	0.00233603

Q3: Profiling of Unified Memory program

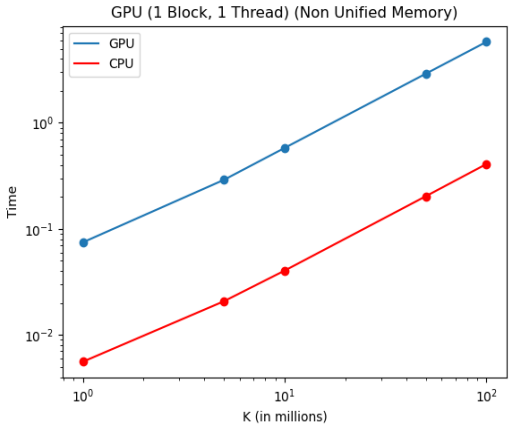
K (in million)	Blocks	Threads	Time (sec)
1	1	1	0.0739801
5	1	1	0.289798
10	1	1	0.579805
50	1	1	2.89889
100	1	1	5.79769

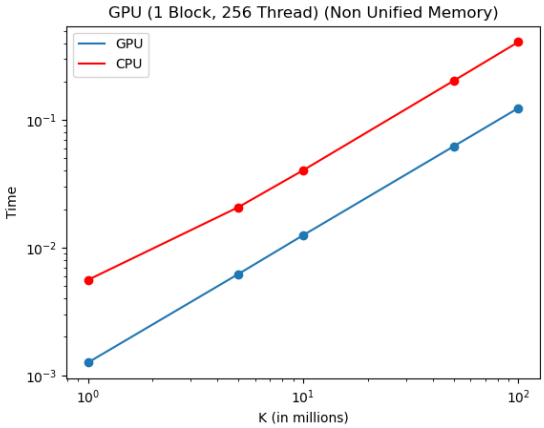
K (in million)	Blocks	Threads	Time (sec)
1	1	256	0.00125504
5	1	256	0.00622296
10	1	256	0.012517
50	1	256	0.0620739
100	1	256	0.124149

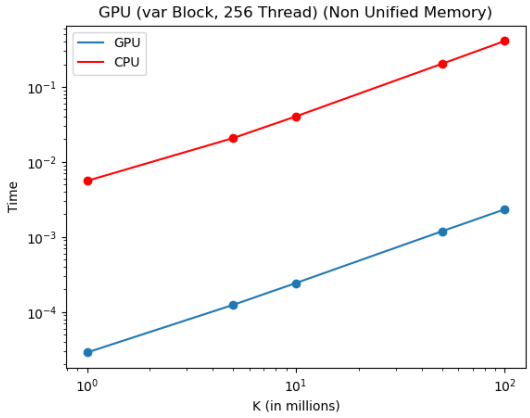
K (in million)	Blocks	Threads	Time (sec)
1	Var	256	0.0000319481
5	Var	256	0.000128984
10	Var	256	0.000249147
50	Var	256	0.00119495
100	Var	256	0.00234222

Q4: Plots

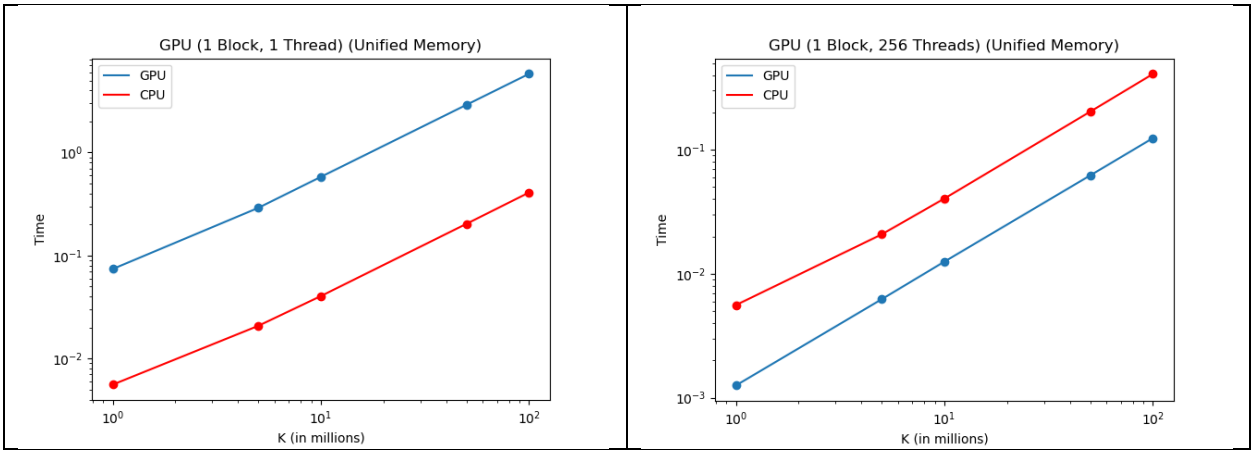
Non-Unified Memory Plots

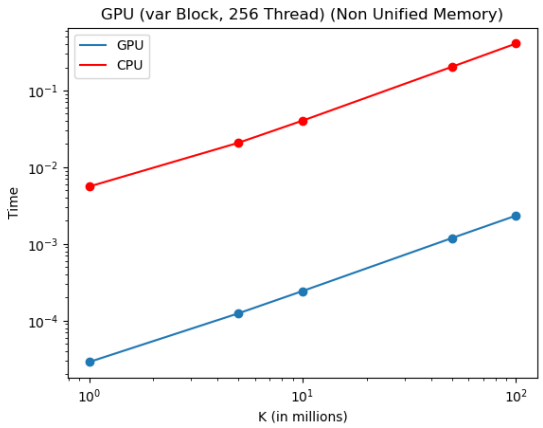






Unified Memory Plots





We can see that GPU performance beats CPU except in the case of 1 block and 1 thread.

But there is not much difference in unified vs non-unified.

Part – C: Convolution in CUDA

Checksum, time (in milli sec)

```
122756344698240.000000,10.764
122756344698240.000000,14.192
122756344698240.000000,0.086
```

Checksum in all the three cases is same.