Parallel Computing

lab 2

Name	ID	SEC	BN
Norhan Reda Abdelwahed Ahmed	9203639	2	31
Hoda Gamal Hamouda Ismail	9203673	2	33

Result for a 1000*1000 matrix:

kernel1 (Element Kernel)

```
==1179== Profiling application: ./K1 Q1-sample-testcases-template-file.txt Q1-sample-output-file.txt
==1179== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   54.17%  1.7719ms         1  1.7719ms  1.7719ms  1.7719ms  [CUDA memcpy DtoH]
                   43.91%  1.4362ms         2  718.11us  699.90us  736.31us  [CUDA memcpy HtoD]
                    1.92%  62.687us         1  62.687us  62.687us  62.687us  MatAdd(float*, float*, float*, int, int)
      API calls:   96.52%  171.34ms         3  57.112ms  73.300us  171.18ms  cudaMalloc
                    2.85%  5.0650ms         3  1.6883ms  924.16us  3.1785ms  cudaMemcpy
                    0.36%  647.18us         3  215.73us  208.11us  229.62us  cudaFree
                    0.17%  302.22us         1  302.22us  302.22us  302.22us  cudaLaunchKernel
                    0.08%  140.19us       114  1.2290us     142ns  56.225us  cuDeviceGetAttribute
                    0.01%  11.786us         1  11.786us  11.786us  11.786us  cuDeviceGetName
                    0.00%  5.7770us         1  5.7770us  5.7770us  5.7770us  cuDeviceGetPCIBusId
                    0.00%  4.5990us         1  4.5990us  4.5990us  4.5990us  cuDeviceTotalMem
                    0.00%  2.0840us         3     694ns     242ns  1.5960us  cuDeviceGetCount
                    0.00%  1.1040us         2     552ns     191ns     913ns  cuDeviceGet
                    0.00%     548ns         1     548ns     548ns     548ns  cuModuleGetLoadingMode
                    0.00%     228ns         1     228ns     228ns     228ns  cuDeviceGetUuid
```

Pros:

Element kernel is faster than both row and column kernels because each thread is working in parallel to compute the value of each element in the result array.

Cons:

The element kernel accesses memory locations randomly or non-consecutive which can result in increased memory latency.

Kernel2 (Row Kernel)

```
==1417== Profiling application: ./K2 Q1-sample-testcases-template-file.txt Q1-sample-output2-file.txt
==1417== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   74.41%  8.9843ms         1  8.9843ms  8.9843ms  8.9843ms  MatAdd(float*, float*, float*, int, int)
                   13.81%  1.6674ms         1  1.6674ms  1.6674ms  1.6674ms  [CUDA memcpy DtoH]
                   11.78%  1.4217ms         2  710.84us  680.35us  741.34us  [CUDA memcpy HtoD]
      API calls:   91.81%  168.83ms         3  56.275ms  71.664us  168.68ms  cudaMalloc
                    7.53%  13.840ms         3  4.6132ms  911.75us  11.949ms  cudaMemcpy
                    0.40%  729.76us         3  243.25us  226.98us  274.91us  cudaFree
                    0.18%  329.46us         1  329.46us  329.46us  329.46us  cudaLaunchKernel
                    0.07%  135.09us       114  1.1850us     138ns  53.826us  cuDeviceGetAttribute
                    0.01%  11.809us         1  11.809us  11.809us  11.809us  cuDeviceGetName
                    0.00%  5.7120us         1  5.7120us  5.7120us  5.7120us  cuDeviceGetPCIBusId
                    0.00%  4.9610us         1  4.9610us  4.9610us  4.9610us  cuDeviceTotalMem
                    0.00%  1.3930us         3     464ns     173ns     957ns  cuDeviceGetCount
                    0.00%     834ns         2     417ns     246ns     588ns  cuDeviceGet
                    0.00%     665ns         1     665ns     665ns     665ns  cuModuleGetLoadingMode
                    0.00%     225ns         1     225ns     225ns     225ns  cuDeviceGetUuid
```

Pros:
Each thread accesses consecutive memory locations within a row, resulting in better memory coalescing and improved memory access efficiency.

Cons:
Much slower than kernel1, as it computes the value of each row in the result array.

Kernel3 (Column Kernel)

```
==1635== Profiling application: ./K3 Q1-sample-testcases-template-file.txt Q1-sample-output3-file.txt
==1635== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   82.13%  14.893ms         1  14.893ms  14.893ms  14.893ms  MatAdd(float*, float*, float*, int, int)
                    9.94%  1.8018ms         1  1.8018ms  1.8018ms  1.8018ms  [CUDA memcpy DtoH]
                    7.93%  1.4386ms         2  719.30us  708.54us  730.07us  [CUDA memcpy HtoD]
      API calls:   88.99%  169.97ms         3  56.656ms  72.206us  169.82ms  cudaMalloc
                   10.44%  19.948ms         3  6.6494ms  942.61us  18.017ms  cudaMemcpy
                    0.32%  616.31us         3  205.44us  203.22us  209.13us  cudaFree
                    0.15%  295.68us         1  295.68us  295.68us  295.68us  cudaLaunchKernel
                    0.07%  132.27us       114  1.1600us     141ns  52.731us  cuDeviceGetAttribute
                    0.01%  10.961us         1  10.961us  10.961us  10.961us  cuDeviceGetName
                    0.00%  7.8810us         1  7.8810us  7.8810us  7.8810us  cuDeviceGetPCIBusId
                    0.00%  4.0990us         1  4.0990us  4.0990us  4.0990us  cuDeviceTotalMem
                    0.00%  2.4980us         3     832ns     233ns  1.9480us  cuDeviceGetCount
                    0.00%  1.0480us         2     524ns     177ns     871ns  cuDeviceGet
                    0.00%     770ns         1     770ns     770ns     770ns  cuModuleGetLoadingMode
                    0.00%     237ns         1     237ns     237ns     237ns  cuDeviceGetUuid
```

Pros:
It is much better for memory access than the element kernel as it accesses memory locations separated by matrix width, while the element kernel accesses memory locations randomly.

Cons:
Much slower than kernel1, as it computes the value of each column in the result array.

A little bit slower than the row kernel, as the row kernel accesses consecutive memory locations.

Result analysis for different array sizes:

For matrix 1000*1000

	Time
kernal1	62.687us
kernel2	8.9843ms
kernel3	14.893ms

For matrix 1000*900

	Time
kernal1	53.856us
kernel2	7.3323ms
kernel3	13.690ms

For matrix 900*1000

	Time
kernal1	56.672us
kernel2	8.1581ms
kernel3	12.295ms

For matrix 10*1000

	Time
kernal1	`3.6470us`
kernel2	`154.21us`
kernel3	`8.4480us`

For matrix 1000*10

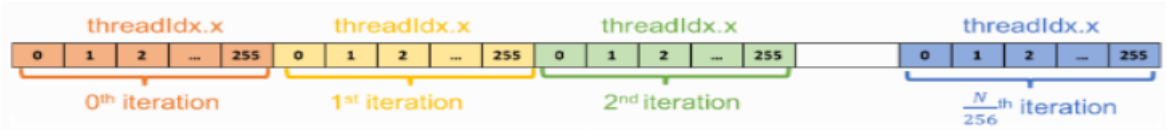
	Time
kernal1	`3.7440us`
kernel2	`7.7760us`
kernel3	`450.62us`

Conclusion:

- **The element kernel** is the fastest one with the least execution time for the kernel, while it can increase memory latency as it accesses non-consecutive memory locations.

- **Both the row and the column kernels** have much more execution time for the kernel than the element kernel, while they have better memory efficiency.

- **The row kernel** has a little bit better memory efficiency as it accesses consecutive memory locations, while the column kernel accesses memory locations separated by matrix width.



- **When there is no significant difference** between rows and columns dimensions, the row kernel is faster than the column kernel.

- **But when there is significant difference** between them, we notice the following:

  For a matrix with a number of columns is much more than the number of rows (10*1000), the row kernel is slower than the column kernel, because each row iterates over a large number of columns.

  And on the opposite side, For a matrix with a number of rows is much more than number of columns (1000*10), the column kernel is slower than the row kernel, because each column iterates over a large number of rows.