Lab (2)

Name	Sec	BN
Basma Hatem Elhoseny	1	16
Sarah Mohamed Hossam Hassan	1	29

Requirement(1) [Matrix Addition]:

Kernel (1) Each Thread Produces one output matrix element:

$$GridSize = (ceil(\frac{No.Cols - 1}{16}), ceil(\frac{No.rows - 1}{16}))$$

*Case(1) 3*4 Matrix*

With configuration that kernel size 16*16 and one thread per element so:

```
3-4
Total no of blocks 1
Total no of threads 256
```

```
==18272== Profiling application: out_1.exe ./tests/test_3_4.txt out.txt
==18272== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   93.78%  40.516us         1  40.516us  40.516us  40.516us  add_matrix(float*, float*, float*, int, int)
                    3.85%  1.6640us         2     832ns     352ns  1.3120us  [CUDA memcpy HtoD]
                    2.37%  1.0240us         1  1.0240us  1.0240us  1.0240us  [CUDA memcpy DtoH]
      API calls:   90.78%  473.60ms         3  157.87ms  3.5000us  473.59ms  cudaMalloc
                    5.72%  29.824ms         1  29.824ms  29.824ms  29.824ms  cuDevicePrimaryCtxRelease
                    3.19%  16.653ms         3  5.5509ms  48.400us  16.337ms  cudaMemcpy
                    0.12%  627.40us         3  209.13us  5.6000us  559.70us  cudaFree
                    0.08%  437.10us         1  437.10us  437.10us  437.10us  cuLibraryUnload
                    0.08%  415.80us         1  415.80us  415.80us  415.80us  cuLibraryLoadData
                    0.02%  99.600us         1  99.600us  99.600us  99.600us  cudaLaunchKernel
                    0.00%  25.700us       114     225ns     100ns  2.2000us  cuDeviceGetAttribute
                    0.00%  9.2000us         3  3.0660us     300ns  8.4000us  cuDeviceGetCount
                    0.00%  4.5000us         2  2.2500us     200ns  4.3000us  cuDeviceGet
                    0.00%  3.2000us         1  3.2000us  3.2000us  3.2000us  cuModuleGetLoadingMode
                    0.00%     800ns         1     800ns     800ns     800ns  cuDeviceGetName
                    0.00%     300ns         1     300ns     300ns     300ns  cuDeviceTotalMem
                    0.00%     300ns         1     300ns     300ns     300ns  cuDeviceGetLuid
                    0.00%     200ns         1     200ns     200ns     200ns  cuDeviceGetUuid

D:\Parallel Computing Labs\Lab2\requirement>
```

Kernel(2) Each Thread Produces one output matrix row:

$$GridSize = ceil(\frac{No.Rows - 1}{16})$$

*Case(1) 3*4 Matrix*

With configuration that kernel size 256 and one thread per rows so:

```
rows:3-cols:4
Total no of blocks 1
Total no of threads 256
```

```
==21016== Profiling application: out_2.exe ./tests/test_3_4.txt out.txt
==21016== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   98.47%  167.09us         1  167.09us  167.09us  167.09us  add_matrix(float*, float*, float*, int, int)
                    0.98%  1.6640us         2     832ns     352ns  1.3120us  [CUDA memcpy HtoD]
                    0.55%     928ns         1     928ns     928ns     928ns  [CUDA memcpy DtoH]
      API calls:   67.85%  142.81ms         3  47.604ms  3.8000us  142.80ms  cudaMalloc
                   18.65%  39.247ms         1  39.247ms  39.247ms  39.247ms  cuDevicePrimaryCtxRelease
                   12.70%  26.727ms         3  8.9090ms  41.100us  26.400ms  cudaMemcpy
                    0.28%  586.70us         3  195.57us  6.1000us  533.70us  cudaFree
                    0.21%  438.20us         1  438.20us  438.20us  438.20us  cuLibraryLoadData
                    0.20%  415.20us         1  415.20us  415.20us  415.20us  cuLibraryUnload
                    0.10%  216.40us         1  216.40us  216.40us  216.40us  cudaLaunchKernel
                    0.01%  26.800us       114     235ns     100ns  2.2000us  cuDeviceGetAttribute
                    0.00%  6.6000us         2  3.3000us     300ns  6.3000us  cuDeviceGet
                    0.00%  5.9000us         3  1.9660us     300ns  5.0000us  cuDeviceGetCount
                    0.00%  3.0000us         1  3.0000us  3.0000us  3.0000us  cuModuleGetLoadingMode
                    0.00%     800ns         1     800ns     800ns     800ns  cuDeviceGetName
                    0.00%     400ns         1     400ns     400ns     400ns  cuDeviceTotalMem
                    0.00%     400ns         1     400ns     400ns     400ns  cuDeviceGetLuid
                    0.00%     200ns         1     200ns     200ns     200ns  cuDeviceGetUuid

D:\Parallel Computing Labs\Lab2\requirement>
```

Kernel(3) Each Thread Produces one output matrix column:

$$GridSize = ceil(\frac{No.Rows - 1}{16})$$

*Case(1) 3*4 Matrix*

With configuration that kernel size 256 and one thread per rows so:

```
Total no of blocks 1
Total no of threads 256
```

```
==4172== Profiling application: out_3.exe ./tests/test_3_4.txt out.txt
==4172== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   97.54%  102.96us         1  102.96us  102.96us  102.96us  add_matrix(float*, float*, float*, int, int)
                    1.55%  1.6320us         2     816ns     320ns  1.3120us  [CUDA memcpy HtoD]
                    0.91%     961ns         1     961ns     961ns     961ns  [CUDA memcpy DtoH]
      API calls:   68.63%  135.98ms         3  45.327ms  5.9000us  135.94ms  cudaMalloc
                   18.66%  36.968ms         1  36.968ms  36.968ms  36.968ms  cuDevicePrimaryCtxRelease
                   11.90%  23.590ms         3  7.8632ms  59.200us  23.255ms  cudaMemcpy
                    0.29%  571.40us         3  190.47us  6.2000us  534.40us  cudaFree
                    0.22%  440.20us         1  440.20us  440.20us  440.20us  cuLibraryLoadData
                    0.16%  325.50us         1  325.50us  325.50us  325.50us  cuLibraryUnload
                    0.12%  229.40us         1  229.40us  229.40us  229.40us  cudaLaunchKernel
                    0.01%  28.500us       114     250ns     100ns  2.2000us  cuDeviceGetAttribute
                    0.00%  6.1000us         3  2.0330us     300ns  5.1000us  cuDeviceGetCount
                    0.00%  5.7000us         2  2.8500us     300ns  5.4000us  cuDeviceGet
                    0.00%  3.3000us         1  3.3000us  3.3000us  3.3000us  cuModuleGetLoadingMode
                    0.00%     800ns         1     800ns     800ns     800ns  cuDeviceGetName
                    0.00%     400ns         1     400ns     400ns     400ns  cuDeviceTotalMem
                    0.00%     300ns         1     300ns     300ns     300ns  cuDeviceGetLuid
                    0.00%     200ns         1     200ns     200ns     200ns  cuDeviceGetUuid

D:\Parallel Computing Labs\Lab2\requirement>_
```

Comments:

- ✓ Case 3*4 it is clear that the kernel 1 is the fastest regarding computing the addition function and nearly the 3 kernels have near copying time form host to device and vice versa 😊
- ✓ We think whatever the matrix size, kernel (1) will be the fastest Are we right ?! 🤔 Let's see <3

Bench Marking:

Notice that these results are based on running matrix produced as random numbers generated by a script :D. [Runed on Colab]

Matrix Shape	Kernel(1) [element]	Kernel(2) [row]	Kernel(3) [col]
3x4	80.672us	279.52us	213.41us
2x2	80.159us	143.94us	144.10us
2x4	80.480us	277.12us	145.79us
Total of 10,000 elements			
100x100	33.184ms	14.142ms	13.420ms
50x200	31.617ms	17.931ms	10.002ms
200x50	28.655ms	10.427ms	17.857ms
10000x1	32.122ms	30.463ms	448.05ms
1x10000	24.259ms	286.08ms	33.404ms
Total of 50,000 elements			
50000x1	238.71ms	1.16151s	1.56368s
1x50000	318.95ms	1.43044s	1.03092s
1000x50	1.54740s	44.100ms	89.304ms
50x1000	1.10863s	87.770ms	43.907ms
Total of 1,000,000 elements			

1000x1000	133.538s	882.08ms	876.51ms
1000000x1	4.85198s	114.753s	28.5801s
1x1000000	6.47006s	28.4147s	141.907s

Notes:

For the First Three example since matrix dim is very small we don't sense a lot of improvement between the 3 kernels so we will apply like stress test

Total of 10,000-element Matrix

For 100x100:

- ✓ The three kernels are nearly the same except that kernel(1) is bit higher due to more threads are required 100x100 while in both kernel(2)&(3) only 100 thread is required.

For 50x200:

- ✓ Kernel(3) is the best bec only 200 thread is required each thread make only 50 operation :D

For 200x50:

- ✓ Kernel(2) is the best bec only 200 thread is required each thread make only 50 operation :D

For 10000x1:

- ✓ Worst Is Kernel(3) 448.05 ms because simply this is computed by single thread (No parallelization) :D

For 1x10000:

- ✓ Worst Is Kernel(2) 286.08 ms because simply this is computed by single thread (No parallelization) :D

Total of 1,000,000-element Matrix

For 1000x1000:

- ✓ It is clear that the worst Time is for Kernel(1) where each thread is computing 1 element but since here the large no of elements we need 1000x1000 thread in total which may not be available to be used together so some is computed with the available threads and finishes then others (Scarcity of Thread compared to the huge no of threads required)
- ✓ While we get better time for kernel(2) & kernel(3) due to less no of threads required in both cases [1000 only :D] (This proves that our claim above is Wrong: kernel(1) is always the best ❌❌)

For 1x1000000 & 1000000x1:

- ✓ The Best will be the one will the orientation corresponding to the no of threads and the other will be the worst because on thread is responsible for the computation (No parallelism)

Conclusion:

We have corrected our faulty claim that kernel(1) will be always the same :D. It depends according to the total no of elements.

But it is clear that of course if the no of rows are more then kernel(2) is the best and kernel(3) will be the worst. And the same for the cols case .

That's All 😁