



Cairo University



Faculty of Engineering
Cairo University

Parallel Computing

#CMP4005

Lab 2 Requirement

Submitted to:

ENG/ Mohamed Abdullah

Submitted By:

NAME	SEC	BN	ID
Yasmine Ashraf Ghanem	2	37	9203707
Yasmin Abdullah Nasser	2	38	9203717

Testcase 1

Number of Columns >> Number of Rows

The first testcase we ran each with a matrix of dimensions 10x2 (10 rows and 2 columns)

- *Kernel 1: each thread produces one output matrix element*

The nvprof command showed the following:

```
==17716== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	56.28%	3.2960us	1	3.2960us	3.2960us	3.2960us	matrix_addition(float*, float*, float*, int, int)
	28.42%	1.6640us	1	1.6640us	1.6640us	1.6640us	[CUDA memcpy DtoH]
	15.30%	896ns	2	448ns	224ns	672ns	[CUDA memcpy HtoD]
API calls:	83.47%	150.14ms	3	50.046ms	3.9000us	150.12ms	cudaMalloc
	15.68%	28.206ms	1	28.206ms	28.206ms	28.206ms	cuDevicePrimaryCtxRelease
	0.25%	454.30us	1	454.30us	454.30us	454.30us	cuLibraryLoadData
	0.24%	433.80us	3	144.60us	4.6000us	396.10us	cudaFree
	0.24%	429.70us	3	143.23us	37.500us	323.70us	cudaMemcpy
	0.06%	103.10us	1	103.10us	103.10us	103.10us	cudaLaunchKernel
	0.03%	50.400us	114	442ns	100ns	10.700us	cuDeviceGetAttribute
	0.02%	38.600us	1	38.600us	38.600us	38.600us	cuLibraryUnload
	0.00%	6.2000us	3	2.0660us	200ns	5.3000us	cuDeviceGetCount
	0.00%	4.7000us	2	2.3500us	400ns	4.3000us	cuDeviceGet
	0.00%	4.5000us	1	4.5000us	4.5000us	4.5000us	cuModuleGetLoadingMode
	0.00%	1.0000us	1	1.0000us	1.0000us	1.0000us	cuDeviceGetName
	0.00%	600ns	1	600ns	600ns	600ns	cuDeviceTotalMem
	0.00%	600ns	1	600ns	600ns	600ns	cuDeviceGetLuid
	0.00%	300ns	1	300ns	300ns	300ns	cuDeviceGetUuid

The kernel (matrix_addition) took approximately 3.3 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one element in the output matrix which means that 20 threads were responsible for the addition of the whole matrix.

- *Kernel 2: each thread produces one output matrix row*

The nvprof command showed the following:

```
==25540== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	68.90%	5.5990us	1	5.5990us	5.5990us	5.5990us	matrix_addition(float*, float*, float*, int, int)
	20.47%	1.6630us	1	1.6630us	1.6630us	1.6630us	[CUDA memcpy DtoH]
	10.63%	864ns	2	432ns	192ns	672ns	[CUDA memcpy HtoD]
API calls:	85.78%	153.50ms	3	51.166ms	3.4000us	153.48ms	cudaMalloc
	13.44%	24.051ms	1	24.051ms	24.051ms	24.051ms	cuDevicePrimaryCtxRelease
	0.27%	488.50us	1	488.50us	488.50us	488.50us	cuLibraryLoadData
	0.21%	371.10us	3	123.70us	36.700us	275.60us	cudaMemcpy
	0.20%	354.90us	3	118.30us	4.2000us	333.50us	cudaFree
	0.04%	76.600us	1	76.600us	76.600us	76.600us	cudaLaunchKernel
	0.03%	51.700us	114	453ns	200ns	6.0000us	cuDeviceGetAttribute
	0.02%	38.000us	1	38.000us	38.000us	38.000us	cuLibraryUnload
	0.00%	7.1000us	3	2.3660us	200ns	6.0000us	cuDeviceGetCount
	0.00%	5.9000us	2	2.9500us	400ns	5.5000us	cuDeviceGet
	0.00%	5.0000us	1	5.0000us	5.0000us	5.0000us	cuModuleGetLoadingMode
	0.00%	1.1000us	1	1.1000us	1.1000us	1.1000us	cuDeviceGetName
	0.00%	600ns	1	600ns	600ns	600ns	cuDeviceGetLuid
	0.00%	400ns	1	400ns	400ns	400ns	cuDeviceTotalMem
	0.00%	300ns	1	300ns	300ns	300ns	cuDeviceGetUuid

The kernel (matrix_addition) took approximately 5.6 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one row in the whole

matrix which means that 2 threads were responsible for the addition of the whole matrix simultaneously.

- *Kernel 3: each thread produces one output matrix column*

The nvprof command showed the following:

```
==29480== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	47.84%	2.4640us	1	2.4640us	2.4640us	2.4640us	matrix_addition(float*, float*, float*, int, int)
	34.79%	1.7920us	1	1.7920us	1.7920us	1.7920us	[CUDA memcpy DtoH]
	17.38%	895ns	2	447ns	223ns	672ns	[CUDA memcpy HtoD]
API calls:	79.43%	101.41ms	3	33.804ms	2.8000us	101.40ms	cudaMalloc
	19.48%	24.866ms	1	24.866ms	24.866ms	24.866ms	cuDevicePrimaryCtxRelease
	0.35%	445.70us	3	148.57us	3.5000us	409.50us	cudaFree
	0.32%	412.30us	3	137.43us	47.200us	256.60us	cudaMemcpy
	0.26%	336.90us	1	336.90us	336.90us	336.90us	cuLibraryLoadData
	0.09%	112.00us	1	112.00us	112.00us	112.00us	cudaLaunchKernel
	0.03%	34.500us	114	302ns	100ns	4.6000us	cuDeviceGetAttribute
	0.02%	24.500us	1	24.500us	24.500us	24.500us	cuLibraryUnload
	0.01%	15.500us	2	7.7500us	300ns	15.200us	cuDeviceGet
	0.01%	7.2000us	3	2.4000us	300ns	5.1000us	cuDeviceGetCount
	0.00%	3.9000us	1	3.9000us	3.9000us	3.9000us	cuModuleGetLoadingMode
	0.00%	2.9000us	1	2.9000us	2.9000us	2.9000us	cuDeviceGetName
	0.00%	2.1000us	1	2.1000us	2.1000us	2.1000us	cuDeviceTotalMem
	0.00%	400ns	1	400ns	400ns	400ns	cuDeviceGetLuid
	0.00%	200ns	1	200ns	200ns	200ns	cuDeviceGetUuid

The kernel (matrix_addition) took approximately 2.5 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one column in the output matrix which means that 10 threads were responsible for the addition of the whole matrix.

In the first testcase the since the number of rows was greater than the number of columns the second kernel which applies parallelism in the level of the matrix rows performs the addition in the least amount of time; the third kernel which applies the parallelism on the columns' level performs the addition in almost double the time. Lastly, the first kernel which parallelizes over the whole matrix with each thread performing the addition of one element performs the addition in a time greater than kernel 2 but less than kernel 3.

Testcase 2

Number of Rows >> Number of Columns

The second testcase we ran each with a matrix of dimensions 2x10 (2 rows and 10 columns)

- *Kernel 1: each thread produces one output matrix element*

The nvprof command showed the following:

```
==24880== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	54.94%	3.0410us	1	3.0410us	3.0410us	3.0410us	matrix_addition(float*, float*, float*, int, int)
	28.89%	1.5990us	1	1.5990us	1.5990us	1.5990us	[CUDA memcpy DtoH]
	16.17%	895ns	2	447ns	223ns	672ns	[CUDA memcpy HtoD]
API calls:	81.51%	154.41ms	3	51.470ms	2.9000us	154.39ms	cudaMalloc
	17.71%	33.555ms	1	33.555ms	33.555ms	33.555ms	cuDevicePrimaryCtxRelease
	0.26%	490.50us	3	163.50us	4.8000us	434.20us	cudaFree
	0.24%	445.90us	1	445.90us	445.90us	445.90us	cuLibraryLoadData
	0.20%	382.60us	3	127.53us	27.500us	298.70us	cudaMemcpy
	0.04%	81.500us	1	81.500us	81.500us	81.500us	cudaLaunchKernel
	0.02%	41.300us	1	41.300us	41.300us	41.300us	cuLibraryUnload
	0.01%	22.700us	114	199ns	100ns	1.9000us	cuDeviceGetAttribute
	0.00%	5.6000us	3	1.8660us	200ns	4.9000us	cuDeviceGetCount
	0.00%	3.7000us	2	1.8500us	100ns	3.6000us	cuDeviceGet
	0.00%	2.2000us	1	2.2000us	2.2000us	2.2000us	cuModuleGetLoadingMode
	0.00%	700ns	1	700ns	700ns	700ns	cuDeviceGetName
	0.00%	500ns	1	500ns	500ns	500ns	cuDeviceTotalMem
	0.00%	500ns	1	500ns	500ns	500ns	cuDeviceGetLuid
	0.00%	200ns	1	200ns	200ns	200ns	cuDeviceGetUuid

The kernel (`matrix_addition`) took approximately 3.0 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one element in the output matrix which means that 20 threads were responsible for the addition of the whole matrix.

- Kernel 2: each thread produces one output matrix row
- The `nvprof` command showed the following:

```
==22576== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	57.95%	3.6160us	1	3.6160us	3.6160us	3.6160us	matrix_addition(float*, float*, float*, int, int)
	26.67%	1.6640us	1	1.6640us	1.6640us	1.6640us	[CUDA memcpy DtoH]
	15.38%	960ns	2	480ns	192ns	768ns	[CUDA memcpy HtoD]
API calls:	86.23%	157.81ms	3	52.603ms	5.3000us	157.77ms	cudaMalloc
	12.85%	23.514ms	1	23.514ms	23.514ms	23.514ms	cuDevicePrimaryCtxRelease
	0.33%	609.40us	1	609.40us	609.40us	609.40us	cuLibraryLoadData
	0.30%	548.50us	3	182.83us	5.9000us	470.60us	cudaFree
	0.18%	326.40us	3	108.80us	55.300us	194.90us	cudaMemcpy
	0.06%	101.90us	1	101.90us	101.90us	101.90us	cudaLaunchKernel
	0.02%	45.000us	1	45.000us	45.000us	45.000us	cuLibraryUnload
	0.02%	38.900us	114	341ns	100ns	3.4000us	cuDeviceGetAttribute
	0.00%	8.4000us	2	4.2000us	300ns	8.1000us	cuDeviceGet
	0.00%	5.1000us	3	1.7000us	200ns	4.2000us	cuDeviceGetCount
	0.00%	3.5000us	1	3.5000us	3.5000us	3.5000us	cuModuleGetLoadingMode
	0.00%	1.1000us	1	1.1000us	1.1000us	1.1000us	cuDeviceGetName
	0.00%	600ns	1	600ns	600ns	600ns	cuDeviceTotalMem
	0.00%	600ns	1	600ns	600ns	600ns	cuDeviceGetLuid
	0.00%	300ns	1	300ns	300ns	300ns	cuDeviceGetUuid

The kernel (`matrix_addition`) took approximately 3.0 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one element in the output matrix which means that 10 threads were responsible for the addition of the whole matrix.

- Kernel 3: each thread produces one output matrix column

The `nvprof` command showed the following:

```
==10820== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	68.38%	5.5360us	1	5.5360us	5.5360us	5.5360us	matrix_addition(float*, float*, float*, int, int)
	20.55%	1.6640us	1	1.6640us	1.6640us	1.6640us	[CUDA memcpy DtoH]
	11.07%	896ns	2	448ns	224ns	672ns	[CUDA memcpy HtoD]
API calls:	86.04%	113.32ms	3	37.772ms	2.3000us	113.30ms	cudaMalloc
	13.03%	17.160ms	1	17.160ms	17.160ms	17.160ms	cuDevicePrimaryCtxRelease
	0.48%	637.30us	1	637.30us	637.30us	637.30us	cuLibraryLoadData
	0.19%	247.50us	3	82.500us	27.300us	175.20us	cudaMemcpy
	0.15%	200.30us	3	66.766us	3.7000us	182.00us	cudaFree
	0.04%	55.400us	1	55.400us	55.400us	55.400us	cudaLaunchKernel
	0.03%	36.700us	1	36.700us	36.700us	36.700us	cuLibraryUnload
	0.02%	32.000us	114	280ns	100ns	2.6000us	cuDeviceGetAttribute
	0.00%	6.5000us	3	2.1660us	300ns	5.0000us	cuDeviceGetCount
	0.00%	4.9000us	2	2.4500us	400ns	4.5000us	cuDeviceGet
	0.00%	3.1000us	1	3.1000us	3.1000us	3.1000us	cuModuleGetLoadingMode
	0.00%	1.0000us	1	1.0000us	1.0000us	1.0000us	cuDeviceGetName
	0.00%	600ns	1	600ns	600ns	600ns	cuDeviceGetLuid
	0.00%	500ns	1	500ns	500ns	500ns	cuDeviceTotalMem
	0.00%	300ns	1	300ns	300ns	300ns	cuDeviceGetUuid

The kernel (*matrix_addition*) took approximately 5.5 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one row in the whole matrix which means that 2 threads were responsible for the addition of the whole matrix simultaneously.

In the second testcase the since the number of columns was greater than the number of rows the second kernel which applies parallelism in the level of the matrix rows performs the addition in the greatest amount of time; the third kernel which applies the parallelism on the columns' level performs the addition in almost half the time. Lastly, the first kernel which parallelizes over the whole matrix with each thread performing the addition of one element performs the addition in a time approximately equal to the time of the first testcase.

Testcase 3

Number of Rows == Number of Columns

The third testcase we ran each with a matrix of dimensions 3x3 (3 rows and 3 columns)

- *Kernel 1: each thread produces one output matrix element*

The *nvprof* command showed the following:

```
==2492== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	57.29%	3.5200us	1	3.5200us	3.5200us	3.5200us	[CUDA memcpy DtoH]
	29.17%	1.7920us	1	1.7920us	1.7920us	1.7920us	matrix_addition(float*, float*, float*, int, int)
	13.54%	832ns	2	416ns	192ns	640ns	[CUDA memcpy HtoD]
API calls:	84.90%	150.10ms	3	50.034ms	4.5000us	150.08ms	cudaMalloc
	14.42%	25.496ms	1	25.496ms	25.496ms	25.496ms	cuDevicePrimaryCtxRelease
	0.27%	482.60us	1	482.60us	482.60us	482.60us	cuLibraryLoadData
	0.17%	292.70us	3	97.566us	39.200us	189.80us	cudaMemcpy
	0.15%	258.40us	3	86.133us	5.8000us	233.30us	cudaFree
	0.04%	66.100us	1	66.100us	66.100us	66.100us	cudaLaunchKernel
	0.03%	48.400us	1	48.400us	48.400us	48.400us	cuLibraryUnload
	0.02%	35.500us	114	311ns	100ns	2.3000us	cuDeviceGetAttribute
	0.00%	5.4000us	3	1.8000us	300ns	4.6000us	cuDeviceGetCount
	0.00%	4.6000us	2	2.3000us	300ns	4.3000us	cuDeviceGet
	0.00%	3.5000us	1	3.5000us	3.5000us	3.5000us	cuModuleGetLoadingMode
	0.00%	900ns	1	900ns	900ns	900ns	cuDeviceGetName
	0.00%	600ns	1	600ns	600ns	600ns	cuDeviceGetLuid
	0.00%	500ns	1	500ns	500ns	500ns	cuDeviceTotalMem
	0.00%	300ns	1	300ns	300ns	300ns	cuDeviceGetUuid

The kernel (`matrix_addition`) took approximately 3.5 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one element in the output matrix which means that 9 threads were responsible for the addition of the whole matrix simultaneously.

- Kernel 2: each thread produces one output matrix row

The `nvprof` command showed the following:

```
==27896== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	60.20%	3.8720us	1	3.8720us	3.8720us	3.8720us	<code>matrix_addition(float*, float*, float*, int, int)</code>
	24.88%	1.6000us	1	1.6000us	1.6000us	1.6000us	<code>[CUDA memcpy DtoH]</code>
	14.93%	960ns	2	480ns	224ns	736ns	<code>[CUDA memcpy HtoD]</code>
API calls:	81.79%	117.52ms	3	39.173ms	2.5000us	117.48ms	<code>cudaMalloc</code>
	17.24%	24.767ms	1	24.767ms	24.767ms	24.767ms	<code>cuDevicePrimaryCtxRelease</code>
	0.33%	468.60us	3	156.20us	29.900us	344.80us	<code>cudaMemcpy</code>
	0.26%	369.00us	1	369.00us	369.00us	369.00us	<code>cuLibraryLoadData</code>
	0.24%	343.80us	3	114.60us	3.2000us	328.80us	<code>cudaFree</code>
	0.08%	114.10us	1	114.10us	114.10us	114.10us	<code>cudaLaunchKernel</code>
	0.04%	51.300us	114	450ns	100ns	32.000us	<code>cuDeviceGetAttribute</code>
	0.03%	36.000us	1	36.000us	36.000us	36.000us	<code>cuLibraryUnload</code>
	0.00%	4.9000us	3	1.6330us	200ns	4.3000us	<code>cuDeviceGetCount</code>
	0.00%	3.5000us	2	1.7500us	200ns	3.3000us	<code>cuDeviceGet</code>
	0.00%	1.7000us	1	1.7000us	1.7000us	1.7000us	<code>cuModuleGetLoadingMode</code>
	0.00%	800ns	1	800ns	800ns	800ns	<code>cuDeviceGetName</code>
	0.00%	400ns	1	400ns	400ns	400ns	<code>cuDeviceTotalMem</code>
	0.00%	300ns	1	300ns	300ns	300ns	<code>cuDeviceGetLuid</code>
	0.00%	200ns	1	200ns	200ns	200ns	<code>cuDeviceGetUuid</code>

The kernel (`matrix_addition`) took approximately 3.9 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one element in the output matrix which means that 9 threads were responsible for the addition of the whole matrix.

- Kernel 3: each thread produces one output matrix column

The `nvprof` command showed the following:

```
==26872== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	62.00%	3.9680us	1	3.9680us	3.9680us	3.9680us	<code>matrix_addition(float*, float*, float*, int, int)</code>
	24.50%	1.5680us	1	1.5680us	1.5680us	1.5680us	<code>[CUDA memcpy DtoH]</code>
	13.50%	864ns	2	432ns	192ns	672ns	<code>[CUDA memcpy HtoD]</code>
API calls:	82.68%	116.39ms	3	38.796ms	2.1000us	116.38ms	<code>cudaMalloc</code>
	16.11%	22.672ms	1	22.672ms	22.672ms	22.672ms	<code>cuDevicePrimaryCtxRelease</code>
	0.47%	665.80us	3	221.93us	2.7000us	629.00us	<code>cudaFree</code>
	0.33%	469.30us	3	156.43us	23.500us	347.90us	<code>cudaMemcpy</code>
	0.26%	369.80us	1	369.80us	369.80us	369.80us	<code>cuLibraryLoadData</code>
	0.08%	110.90us	1	110.90us	110.90us	110.90us	<code>cudaLaunchKernel</code>
	0.04%	61.800us	1	61.800us	61.800us	61.800us	<code>cuLibraryUnload</code>
	0.01%	19.700us	114	172ns	100ns	2.0000us	<code>cuDeviceGetAttribute</code>
	0.00%	3.8000us	3	1.2660us	200ns	3.4000us	<code>cuDeviceGetCount</code>
	0.00%	3.6000us	2	1.8000us	200ns	3.4000us	<code>cuDeviceGet</code>
	0.00%	1.9000us	1	1.9000us	1.9000us	1.9000us	<code>cuModuleGetLoadingMode</code>
	0.00%	800ns	1	800ns	800ns	800ns	<code>cuDeviceGetName</code>
	0.00%	400ns	1	400ns	400ns	400ns	<code>cuDeviceTotalMem</code>
	0.00%	300ns	1	300ns	300ns	300ns	<code>cuDeviceGetLuid</code>
	0.00%	100ns	1	100ns	100ns	100ns	<code>cuDeviceGetUuid</code>

The kernel (`matrix_addition`) took approximately 4.0 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one element in the output matrix which means that 9 threads were responsible for the addition of the whole matrix.

In the third testcase since the number of columns and the number of rows are equal all the kernels performed the matrix addition in approximately the same time since kernel 2 and kernel 3 apply the same level of parallelism and kernel 1 is consistent with all the other testcases.

Testcase 4

Number of Rows == Number of Columns

The fourth testcase we ran each with a matrix of dimensions 10x10 to see the effect on kernel1

- *Kernel 1: each thread produces one output matrix element*

The nvprof command showed the following:

```
==22572== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	54.24%	3.0720us	1	3.0720us	3.0720us	3.0720us	matrix_addition(float*, float*, float*, int, int)
	29.38%	1.6640us	1	1.6640us	1.6640us	1.6640us	[CUDA memcpy DtoH]
	16.38%	928ns	2	464ns	256ns	672ns	[CUDA memcpy HtoD]
API calls:	80.16%	111.33ms	3	37.110ms	2.3000us	111.32ms	cudaMalloc
	18.15%	25.208ms	1	25.208ms	25.208ms	25.208ms	cuDevicePrimaryCtxRelease
	1.03%	1.4308ms	1	1.4308ms	1.4308ms	1.4308ms	cuLibraryLoadData
	0.28%	394.60us	3	131.53us	23.000us	289.80us	cudaMemcpy
	0.23%	314.00us	3	104.67us	3.6000us	281.10us	cudaFree
	0.05%	72.200us	1	72.200us	72.200us	72.200us	cudaLaunchKernel
	0.05%	70.800us	114	621ns	100ns	8.7000us	cuDeviceGetAttribute
	0.03%	42.600us	1	42.600us	42.600us	42.600us	cuLibraryUnload
	0.00%	6.5000us	3	2.1660us	300ns	4.4000us	cuDeviceGetCount
	0.00%	5.4000us	2	2.7000us	300ns	5.1000us	cuDeviceGet
	0.00%	5.4000us	1	5.4000us	5.4000us	5.4000us	cuDeviceGetName
	0.00%	3.4000us	1	3.4000us	3.4000us	3.4000us	cuModuleGetLoadingMode
	0.00%	3.1000us	1	3.1000us	3.1000us	3.1000us	cuDeviceGetLuid
	0.00%	2.4000us	1	2.4000us	2.4000us	2.4000us	cuDeviceTotalMem
	0.00%	200ns	1	200ns	200ns	200ns	cuDeviceGetUuid

The kernel (matrix_addition) took approximately 3.0 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one element in the output matrix which means that 100 threads were responsible for the addition of the whole matrix.

- *Kernel 2: each thread produces one output matrix row*

The nvprof command showed the following:

```
==28652== Profiling result:
```

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:	69.65%	5.7280us	1	5.7280us	5.7280us	5.7280us	matrix_addition(float*, float*, float*, int, int)
	19.47%	1.6010us	1	1.6010us	1.6010us	1.6010us	[CUDA memcpy DtoH]
	10.88%	895ns	2	447ns	223ns	672ns	[CUDA memcpy HtoD]
API calls:	83.37%	139.39ms	3	46.464ms	2.3000us	139.39ms	cudaMalloc
	15.81%	26.427ms	1	26.427ms	26.427ms	26.427ms	cuDevicePrimaryCtxRelease
	0.28%	474.50us	3	158.17us	44.900us	296.30us	cudaMemcpy
	0.23%	387.80us	3	129.27us	5.8000us	346.40us	cudaFree
	0.20%	335.10us	1	335.10us	335.10us	335.10us	cuLibraryLoadData
	0.05%	82.600us	1	82.600us	82.600us	82.600us	cudaLaunchKernel
	0.03%	51.900us	1	51.900us	51.900us	51.900us	cuLibraryUnload
	0.01%	24.800us	114	217ns	100ns	2.3000us	cuDeviceGetAttribute
	0.00%	3.8000us	2	1.9000us	200ns	3.6000us	cuDeviceGet
	0.00%	3.7000us	3	1.2330us	200ns	3.1000us	cuDeviceGetCount
	0.00%	3.2000us	1	3.2000us	3.2000us	3.2000us	cuModuleGetLoadingMode
	0.00%	1.0000us	1	1.0000us	1.0000us	1.0000us	cuDeviceGetName
	0.00%	400ns	1	400ns	400ns	400ns	cuDeviceGetLuid
	0.00%	300ns	1	300ns	300ns	300ns	cuDeviceGetUuid
	0.00%	200ns	1	200ns	200ns	200ns	cuDeviceTotalMem

The kernel (`matrix_addition`) took approximately 5.7 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one row in the output matrix which means that 10 threads were responsible for the addition of the whole matrix.

- *Kernel 3: each thread produces one output matrix column*

The `nvprof` command showed the following:

```
==6192== Profiling result:
```

	Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities:		67.69%	5.6320us	1	5.6320us	5.6320us	5.6320us	<code>matrix_addition(float*, float*, float*, int, int)</code>
		21.15%	1.7600us	1	1.7600us	1.7600us	1.7600us	<code>[CUDA memcpy DtoH]</code>
		11.15%	928ns	2	464ns	256ns	672ns	<code>[CUDA memcpy HtoD]</code>
API calls:		80.18%	99.812ms	3	33.271ms	2.2000us	99.806ms	<code>cudaMalloc</code>
		18.72%	23.306ms	1	23.306ms	23.306ms	23.306ms	<code>cuDevicePrimaryCtxRelease</code>
		0.35%	437.10us	3	145.70us	23.000us	325.70us	<code>cudaMemcpy</code>
		0.33%	408.80us	1	408.80us	408.80us	408.80us	<code>cuLibraryLoadData</code>
		0.23%	292.50us	3	97.500us	3.6000us	267.90us	<code>cudaFree</code>
		0.10%	128.30us	1	128.30us	128.30us	128.30us	<code>cudaLaunchKernel</code>
		0.04%	48.700us	1	48.700us	48.700us	48.700us	<code>cuLibraryUnload</code>
		0.02%	19.300us	114	169ns	100ns	1.7000us	<code>cuDeviceGetAttribute</code>
		0.01%	17.900us	2	8.9500us	100ns	17.800us	<code>cuDeviceGet</code>
		0.00%	4.1000us	3	1.3660us	200ns	3.5000us	<code>cuDeviceGetCount</code>
		0.00%	2.1000us	1	2.1000us	2.1000us	2.1000us	<code>cuModuleGetLoadingMode</code>
		0.00%	900ns	1	900ns	900ns	900ns	<code>cuDeviceGetName</code>
		0.00%	400ns	1	400ns	400ns	400ns	<code>cuDeviceTotalMem</code>
		0.00%	300ns	1	300ns	300ns	300ns	<code>cuDeviceGetLuid</code>
		0.00%	200ns	1	200ns	200ns	200ns	<code>cuDeviceGetUuid</code>

The kernel (`matrix_addition`) took approximately 5.6 microseconds to perform the matrix addition on the two matrices. In this kernel each thread is responsible for the output of one column in the output matrix which means that 10 threads were responsible for the addition of the whole matrix.

In the third testcase since the number of columns and the number of rows are equal all the kernels performed the matrix addition in approximately the same time since kernel 2 and kernel 3 apply the same level of parallelism and kernel 1 is consistent with all the other testcases.

Conclusion

In conclusion, based on the testcases applied and the `nvprof` command the best performance is achieved depending on the nature of the matrix; if the matrices consists of number of rows \gg the number of columns then kernel 2 is the best choice (each thread is responsible for the output of one row in the output matrix). If the matrix consists of the number of columns \gg number of rows then the best approach is to apply the parallelism over the columns like kernel 3 (each thread is responsible for the output of one column in the output matrix). Executing kernel 1 where each thread is responsible for one element in the output matrix gives an average and close time with all the testcases we tried which is closer to the least time achieved each time so it doesn't give the best performance but doesn't give the worst performance either. However, when we increased the number of rows and columns significantly with the rows \approx columns the difference in time was conveyed between kernel 1 and kernel 2 and 3;

where kernel 1 performed the addition in the least amount of time with approximately half the time it took kernels 2 and 3. Therefore, when the matrix is large with approximately equal number of rows and columns it is best to use kernel 1 or to parallelise each element in the matrix.