CUDA Lab 2. Thread configuration

1. Understand how to calculate global index of a threads in 1D blocks and 1D grids
2. Learn how to organize and specify threads in 2D and 3D blocks
3. Learn how to organize and specify blocks in 2D and 3D grids
4. Understand how to calculate global index of a threads in 2D and 3D blocks and grids

Continue to work on the CUDA sample on vector addition used in the first lab, where only one 1D block of threads is used.

Now consider using multiple 1D thread blocks, say, two or three thread blocks, with the number of threads per block being 3, 4, 5, 6, ….

```
addKernel << <2, 3 >> > (dev_c, dev_a, dev_b);
addKernel << <2, 4 >> > (dev_c, dev_a, dev_b);
addKernel << <2, 5 >> > (dev_c, dev_a, dev_b);
addKernel << <2, 6 >> > (dev_c, dev_a, dev_b);
addKernel << <3, 2 >> > (dev_c, dev_a, dev_b);

…
```

Observe the results calculated based on different thread configurations shown above. You may find that the addition may not always be calculated properly for some thread configurations.

Exercise 1. Understand the block and thread indices

List the values for the built-in variables **threadIdx.x** and **blockIdx.x** corresponding to the following thread configurations used for executing the kernel *addKernel*( ) function on GPU:

1) addKernel << <1, 5 >> > (dev_c, dev_a, dev_b);
2) addKernel << <2, 3 >> > (dev_c, dev_a, dev_b);
3) addKernel << <3, 2 >> > (dev_c, dev_a, dev_b);
4) addKernel << <6, 1 >> > (dev_c, dev_a, dev_b);

Exercise 2. Find vector addition using multiple 1D thread blocks

For the vector addition problem considered in the CUDA template, find the solution based on the following thread configurations by modifying the following line of CUDA code:

```
int i = threadIdx.x;
```

1) addKernel << <2, 3 >> > (dev_c, dev_a, dev_b);
2) addKernel << <3, 2 >> > (dev_c, dev_a, dev_b);
3) addKernel << <6, 1 >> > (dev_c, dev_a, dev_b);

Exercise 3. Understand the thread indices for 2D blocks

List the values for the built-in variables **threadIdx.x** and **threadIdx.y** corresponding to the following thread configurations used for executing the kernel *addKernel*( ) function on GPU:

1) addKernel << <1, dim3(2, 3) >> > (dev_c, dev_a, dev_b);
2) addKernel << <1, dim3(3, 3) >> > (dev_c, dev_a, dev_b);
3) addKernel << <1, dim3(5, 1) >> > (dev_c, dev_a, dev_b);

Exercise 4. Find vector addition using one 2D thread block

For the vector addition problem considered in the CUDA template, find the solution based on the following thread configurations by modifying the following line of CUDA code:

```
int i = threadIdx.x;
```

1) addKernel << <1, dim3(2, 3) >> > (dev_c, dev_a, dev_b);
2) addKernel << <1, dim3(3, 2) >> > (dev_c, dev_a, dev_b);
3) addKernel << <1, dim3(5, 1) >> > (dev_c, dev_a, dev_b);

Exercise 5. Find vector addition using multiple 2D thread blocks

For the vector addition problem considered in the CUDA template, find the solution based on the following thread configurations by modifying the following line of CUDA code:

```
int i = threadIdx.x;
```

1) addKernel << < dim3(1, 3), dim3(3, 1) >> > (dev_c, dev_a, dev_b);
2) addKernel << < dim3(2, 3), dim3(2, 2) >> > (dev_c, dev_a, dev_b);
3) addKernel << < dim3(2, 2), dim3(2, 3) >> > (dev_c, dev_a, dev_b);

Exercise 6. Matrix addition

Matrices are two dimensional arrays of numbers arranged in rows and columns. Like vector addition, matrix addition is a per-element addition, which can be done in parallel.

Procedurally define the two matrices of 32x32, say, by initializing the elements A(i, j) and B(i,j) located at the i-th row and j-th column as:

```
A[i][j]= i+j;

B[i][j]= (i+j)*10
```

Write a CUDA program to find the addition of two matrices.