LESSON 2

Matrix manipulation

TOPICS

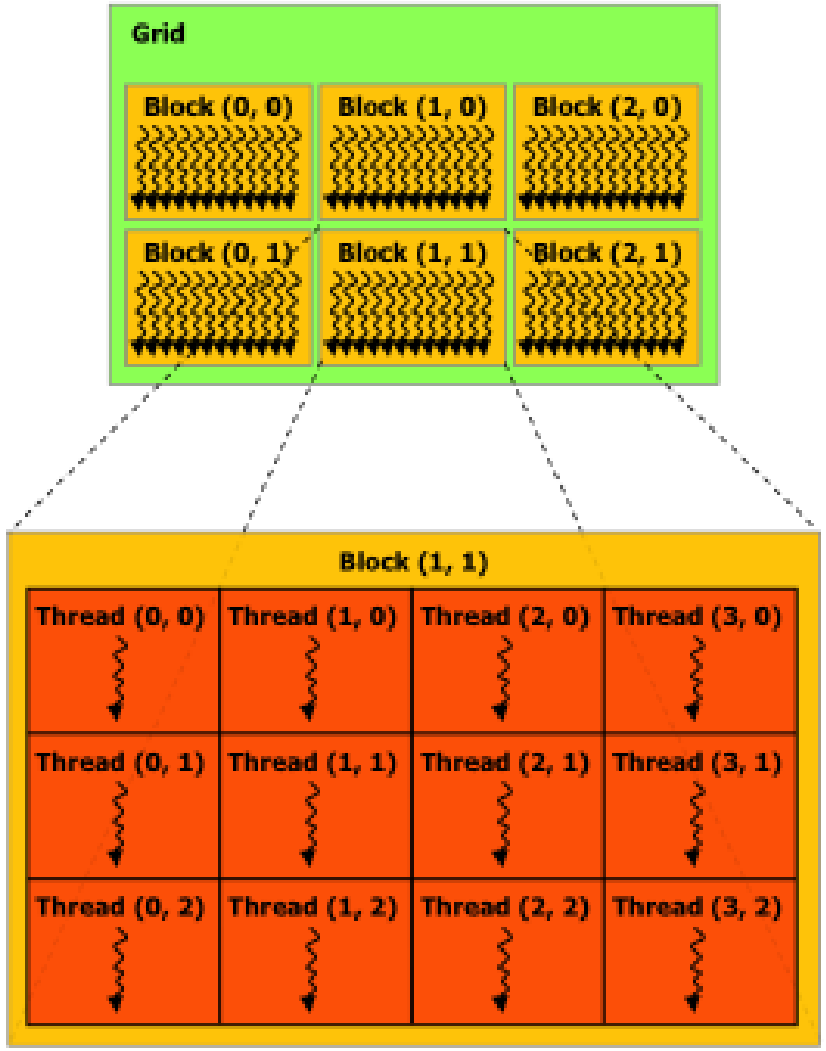
- How to create a matrix of blocks and threads in the device
- How to move in a matrix that is stored as an array

Key words: *dim3, matrix, BLOCK_SIZE, STRIDE.*

More about grids, blocks and threads

- CUDA runtime allows you to launch a two-dimensional grid of blocks, and each block is a three-dimensional array of threads.

- To create a matrix of blocks and threads of two dimension we are going to use *dim3*, that is the way to create (encapsulate multidimensional tuple) a matrix of two or three dimensions.



- We can have different configurations using dim3, for example:
  - dim3(C,1,1) = dim3(C) = C
  - <<<1,10>>> = <<<dim3(1,1,1), dim3(10,1,1)>>>
- In order to have a grid, it is possible to use:
  - kernel<<<dim3(Ax, Ay, Az), dim3(Bx, By, Bz)>>>()
  - <<<dim3(10,2), dim3(4,4)>>>
- When one of the dimension is no specified, it is replaced automatically by 1.

- A toy matrix of 6x6 is mapped to an array of 36 elements. The objective of the process is to divide the matrix in small blocks of 3x3 elements and the kernel will be in charge of divide the element 0 of each small block by the other elements in the small block; the result will be stored in a matrix of the same size.

- The next table shows how the elements are organized in the matrix. The *BLOCK_SIZE* is 3 because it indicates the length of each small block. The *STRIDE* is 6, and it indicates the full length of the matrix.

0	1	2	0	1	2
3	4	5	3	4	5
6	7	8	6	7	8
0	1	2	0	1	2
3	4	5	3	4	5
6	7	8	6	7	8

- On the device, each small block is mapped to a block in the grid, and for every operation, a thread will exist, therefore there are as many threads as elements in the matrix.

- The execution configuration will need to use the dim3 type to generate the grid:
  - *dim3 ThreadsSmallBlocks(3, 3);*
  - *dim3 GridSmallBlocks(2 , 2);*

Block 0,0	Block 1,0
Block 0,1	Block 1,1

- The trick in this code is to know how to generate de indexes to make the operation, due to the fact that the matrix is organized as an array of 36 elements instead of a matrix of 6x6.

- To calculate the dividend —that is the element 0— of each small block it is necessary to take into account the BLOCK_SIZE and the STRIDE.

0	1	2	0	1	2
3	4	5	3	4	5
6	7	8	6	7	8
0	1	2	0	1	2
3	4	5	3	4	5
6	7	8	6	7	8

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
0	1	2	0	1	2	3	4	5	3	4	5	6	7	8	6	7	8	0	1	2	0	1	2	3	4	5	3	4	5	6	7	8	6	7	8

- The indexes of each small block that have data 0 in the above table are {0, 3, 18, 21}. And using the formula:

```
int bx = blockIdx.x;
int by = blockIdx.y;

        d_a[ (by * BLOCK_SIZE + 0) * STRIDE + (bx * BLOCK_SIZE + 0) ];
```

d_a[(0*3+0)*6+(0*3+0)] = a[0]

d_a[(0*3+0)*6+(1*3+0)] = a[3]

d_a[(1*3+0)*6+(0*3+0)] = a[18]

d_a[(1*3+0)*6+(1*3+0)] = a[21]

- To calculate the indexes of the other elements of the matrix we need to change 0 for the thread index:

```
int tx = threadIdx.x;
int ty = threadIdx.y;

        d_a[ (by * BLOCK_SIZE + ty) * STRIDE + (bx * BLOCK_SIZE + tx) ];
```

Example

- 01matrix_manipulation: This program must divide the element 0 of each 3x3 block by all the elements in the same block.

Example

- 02matrices_addition: This program adds two square matrices.

Practice

- 03matrices_additionv2: This program adds two square matrices of 500 elements.

Example

- 04matrix_transpose: This program executes the transposed operation in a small matrix.

Practice

- 05matrix_transposev2: This program must execute the transposed operation in a big matrix.