

Parallel Computing Workshop

Ms. Kranti Ingale

Prime Minister's Research Fellows (**PMRF**) Scholar

CSE, IIT Madras

Roadmap

- Introduction to GPU
- CUDA Program Flow and CPU-GPU Communication
- Thread organization (Grids, Blocks, Threads, 1D/2D)
- CUDA Memory Model
- CUDA Functions
- CUDA Thrust

Roadmap

- Introduction to GPU
- CUDA Program Flow and CPU-GPU Communication
- Thread organization (Grids, Blocks, Threads, 1D/2D)
- CUDA Memory Model
- CUDA Functions
- CUDA Thrust

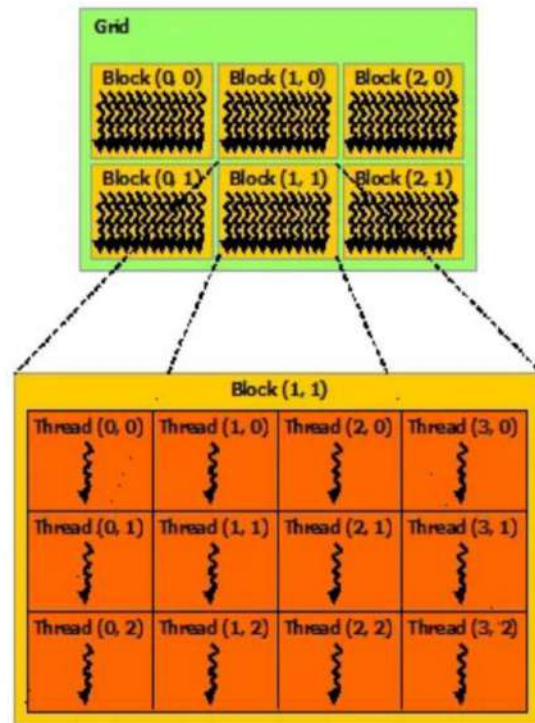
Thread organization (Grids, Blocks, Threads)

- A kernel (GPU function) is launched as a collection of thread blocks called **Grid**.
- Grid size is defined using the number of blocks.
- For eg. Grid of size 8 contains 8 thread blocks.

Thread organization (Grids, Blocks, Threads)

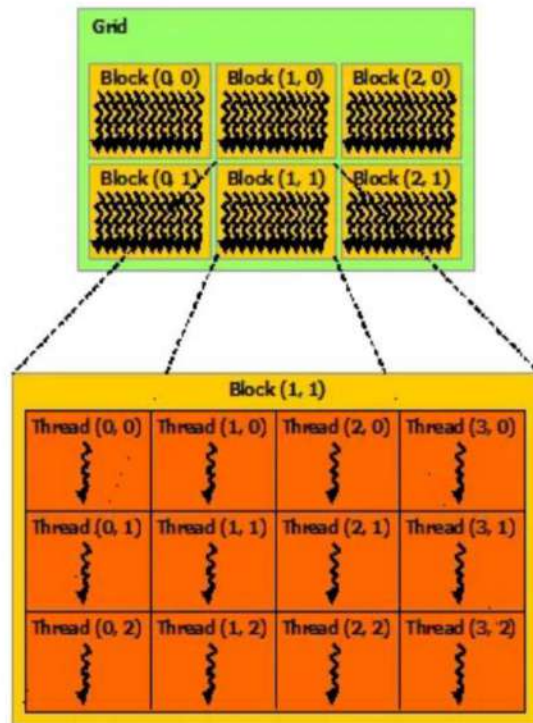
- **Block** : A thread-block is a 3D array of threads.
- **Thread** : Single execution unit that runs GPU function (kernel) on GPU.

Standard Variables



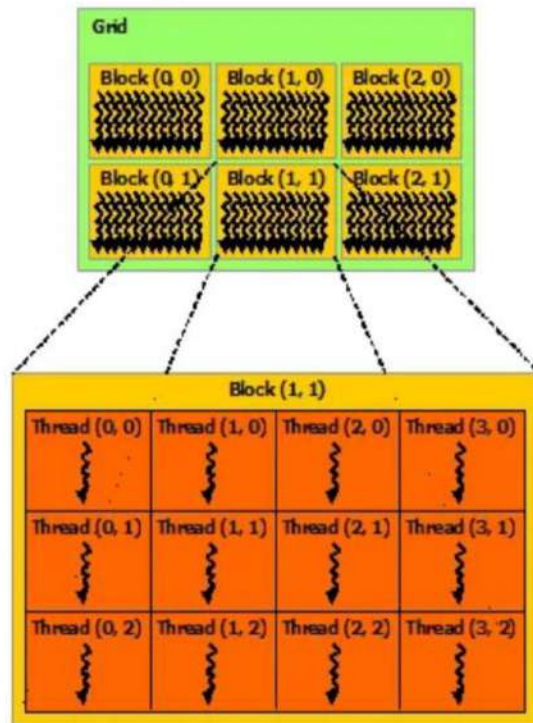
- **Dimension of Grid (Number of blocks in the grid)**

Standard Variables



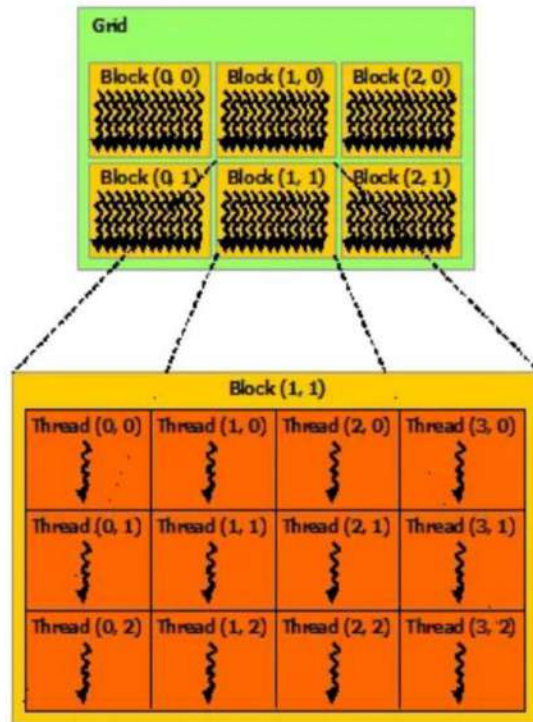
- **Dimension of Grid (Number of blocks in the grid)**
gridDim.x : number of blocks in the x dimension of the grid

Standard Variables



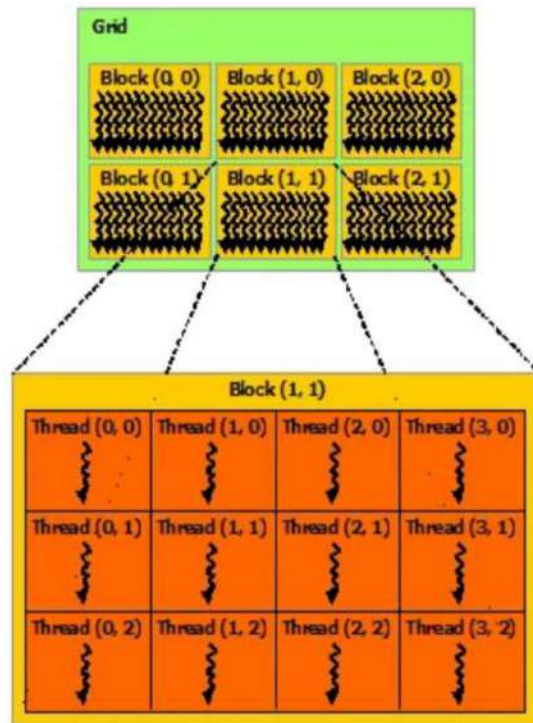
- **Dimension of Grid (Number of blocks in the grid)**
gridDim.x : number of blocks in the x dimension of the grid
gridDim.y : number of blocks in the y dimension of the grid

Standard Variables



- **Dimension of Grid (Number of blocks in the grid)**
gridDim.x : number of blocks in the x dimension of the grid
gridDim.y : number of blocks in the y dimension of the grid
gridDim.z : number of blocks in the z dimension of the grid

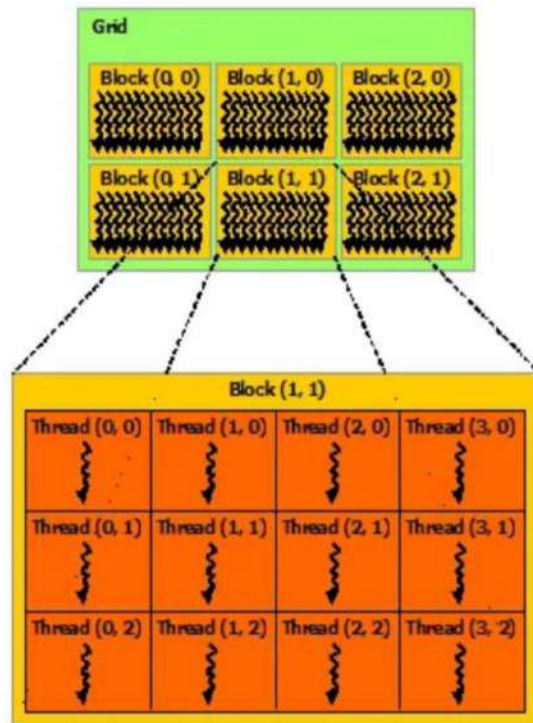
Standard Variables



- **Dimension of block (Number of threads in a block)**

blockDim.x : number of threads in the x dimension of the block

Standard Variables

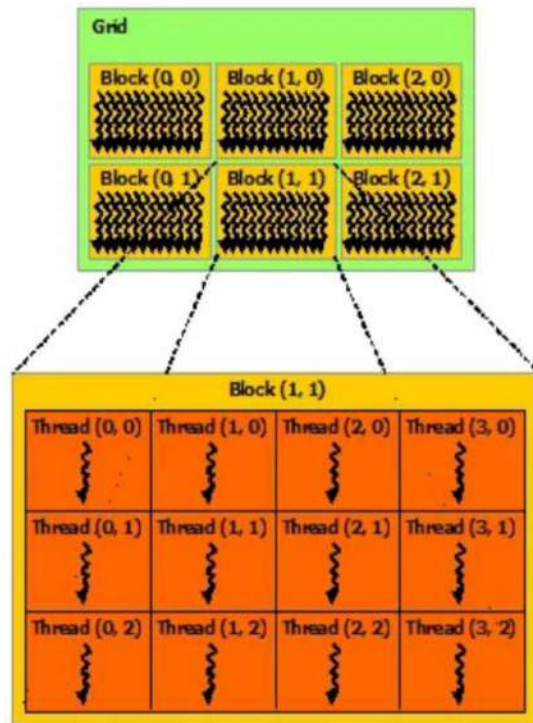


- **Dimension of block (Number of threads in a block)**

blockDim.x : number of threads in the x dimension of the block

blockDim.y : number of threads in the y dimension of the block

Standard Variables



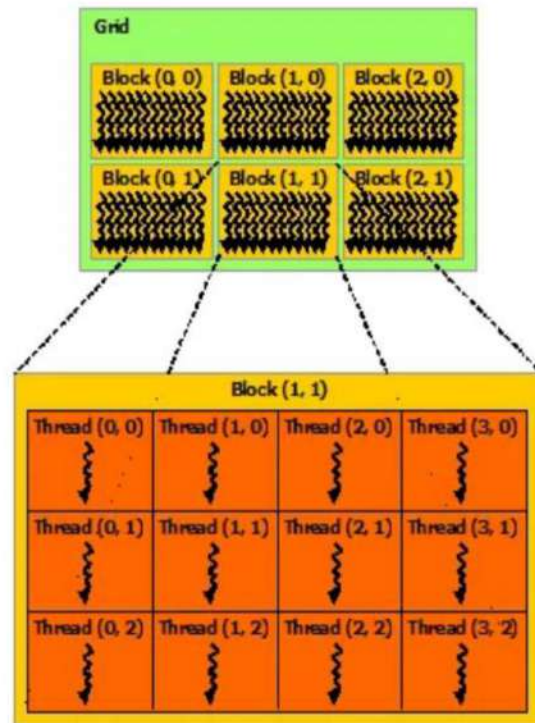
- **Dimension of block (Number of threads in a block)**

blockDim.x : number of threads in the x dimension of the block

blockDim.y : number of threads in the y dimension of the block

blockDim.z : number of threads in the z dimension of the block

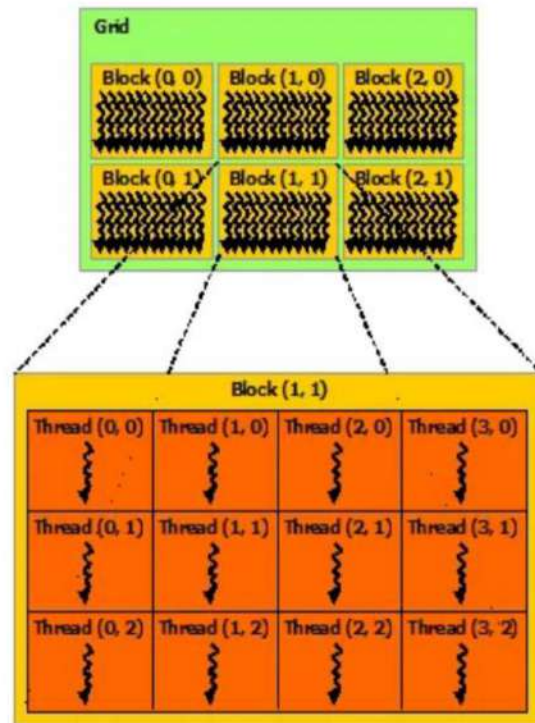
Standard Variables



- **Block Index**

blockIdx.x : block's index in x dimension

Standard Variables

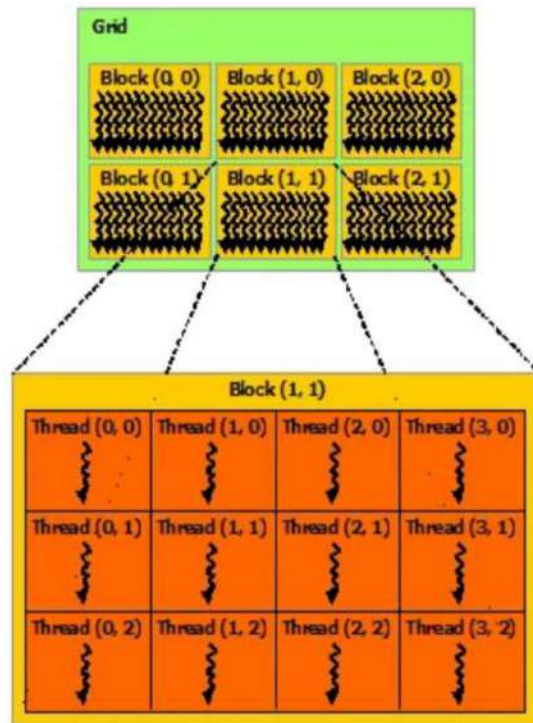


- **Block Index**

blockIdx.x : block's index in x dimension

blockIdx.y : block's index in y dimension

Standard Variables



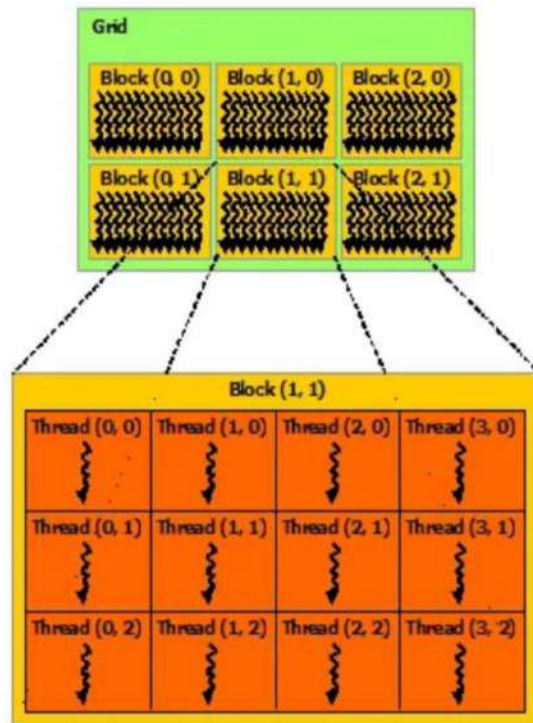
- **Block Index**

blockIdx.x : block's index in x dimension

blockIdx.y : block's index in y dimension

blockIdx.z : block's index in z dimension

Standard Variables



- **Block Index**

blockIdx.x : block's index in x dimension

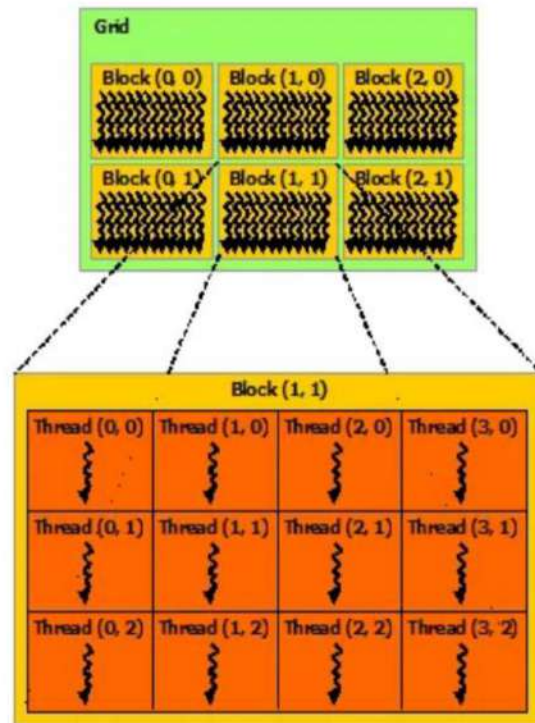
blockIdx.y : block's index in y dimension

blockIdx.z : block's index in z dimension

block (1,0)

blockIdx.x = 1 , blockIdx.y = 0

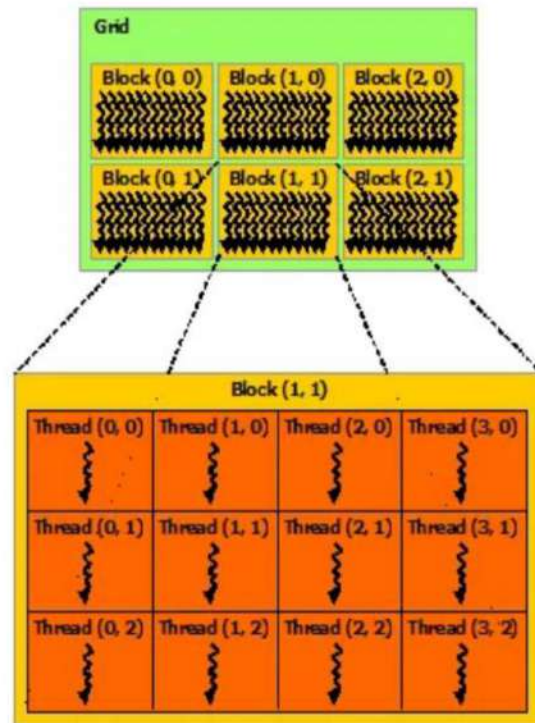
Standard Variables



- Thread Index

threadIdx.x : thread's index in x dimension

Standard Variables

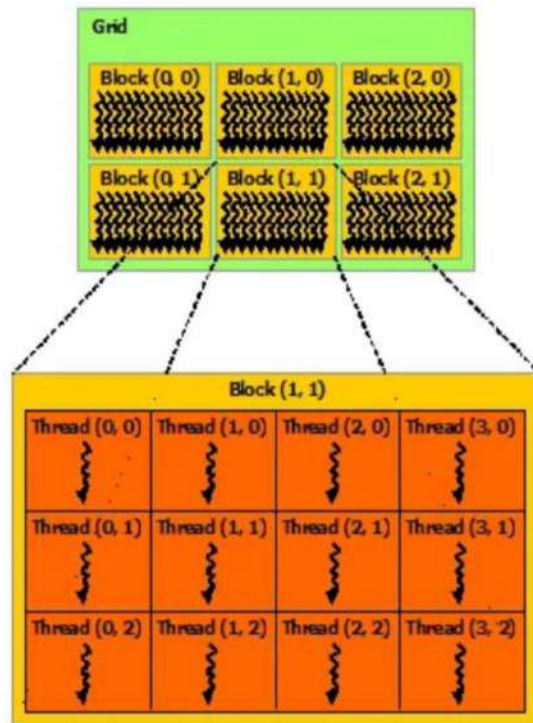


- **Thread Index**

threadIdx.x : thread's index in x dimension

threadIdx.y : thread's index in y dimension

Standard Variables



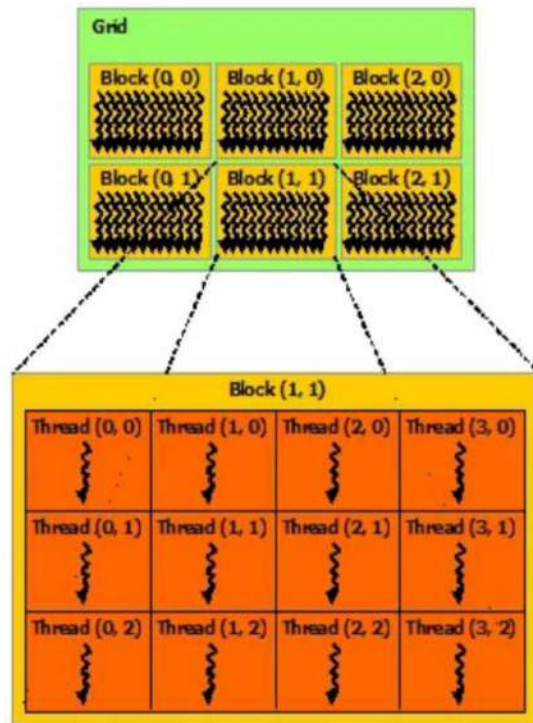
- Thread Index

threadIdx.x : thread's index in x dimension

threadIdx.y : thread's index in y dimension

threadIdx.z : thread's index in z dimension

Standard Variables



- Thread Index

threadIdx.x : thread's index in x dimension

threadIdx.y : thread's index in y dimension

threadIdx.z : thread's index in z dimension

thread (3,1)

threadIdx.x = 3 , threadIdx.y = 1

Thread indexing : 1D grid of 1D blocks

- `kernel_name<<<numBlocks,threadsPerBlock >>>();`
- **`dkernel<<<3,4>>>():`**
 `gridDim.x = 3`
 `blockDim.x = 4`

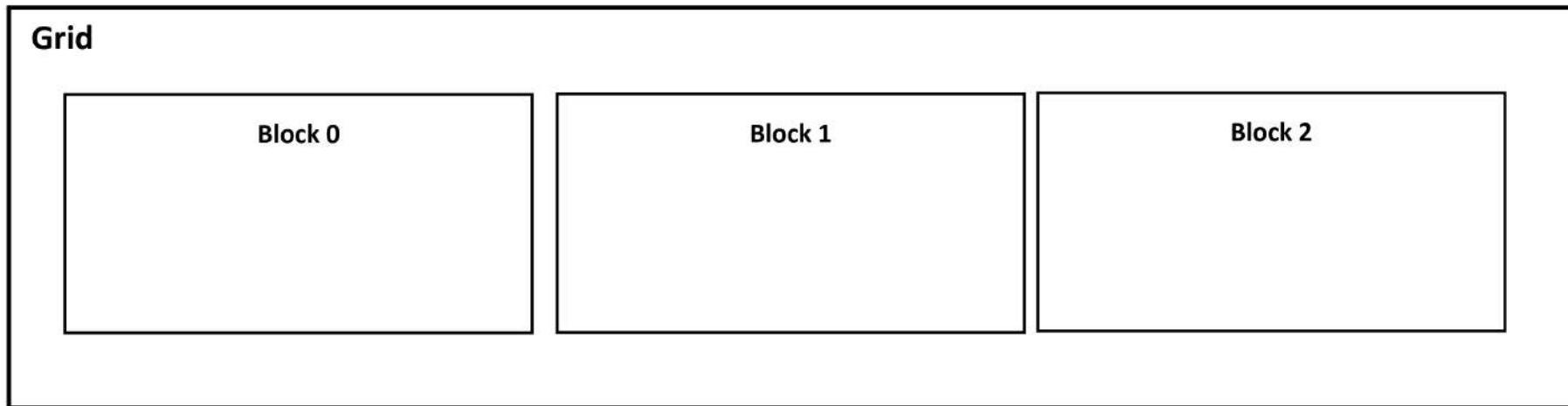
Thread indexing : 1D grid of 1D blocks

- `dkernel<<<3,4>>>()`



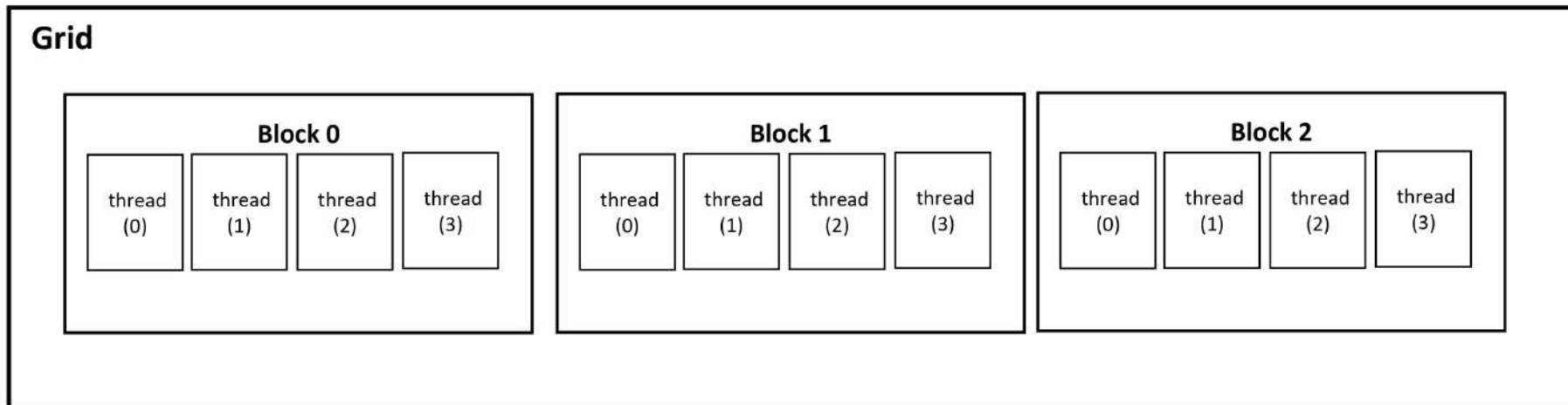
Thread indexing : 1D grid of 1D blocks

- `dkernel<<<3,4>>>()`



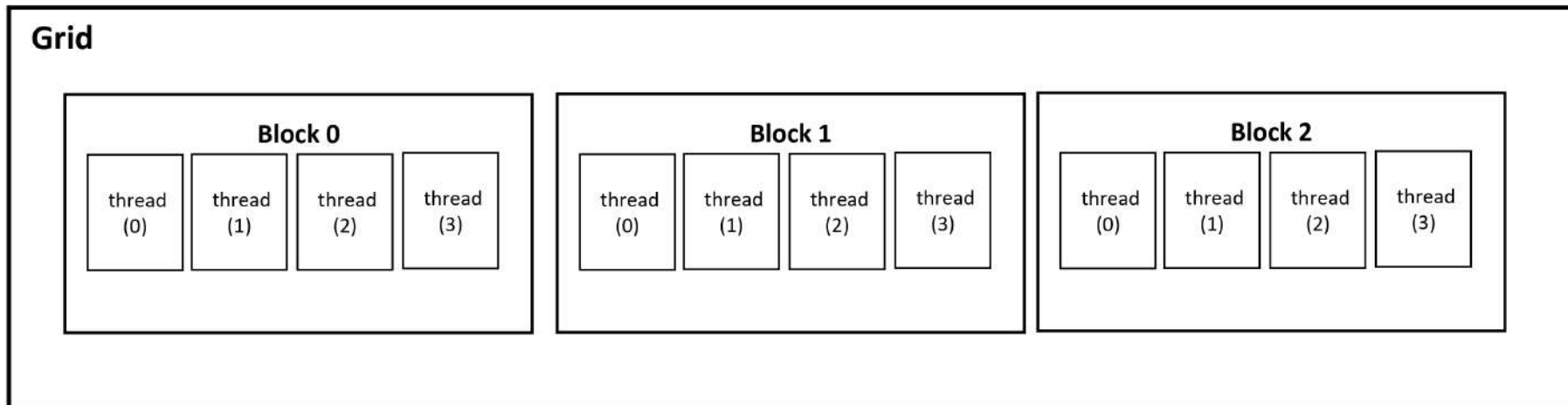
Thread indexing : 1D grid of 1D blocks

- `dkernel<<<3,4>>>()`



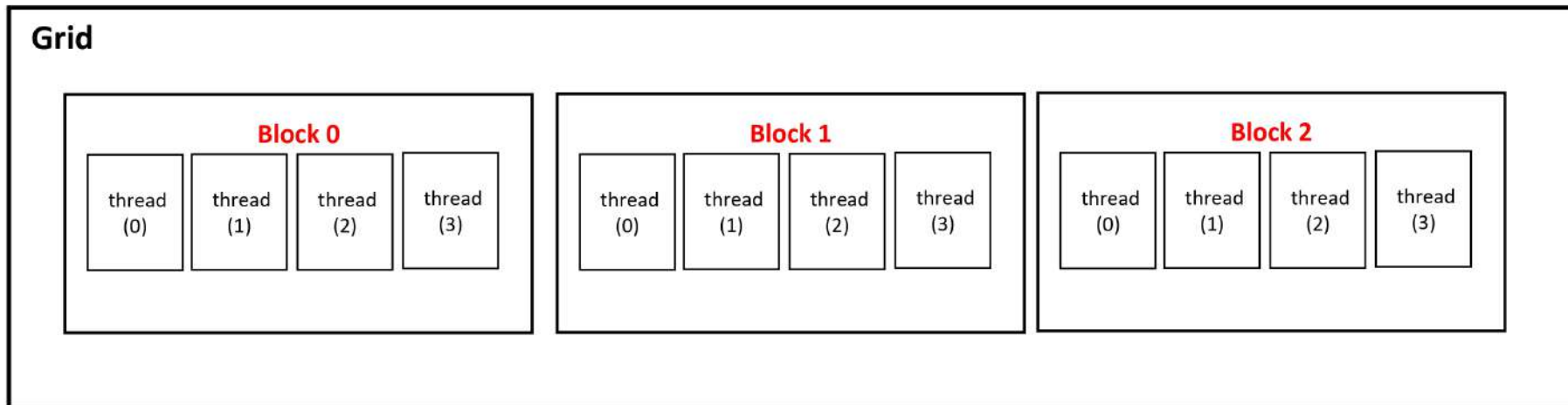
Thread indexing : 1D grid of 1D blocks

- $\text{threadId} = (\text{blockIdx.x} * \text{blockDim.x}) + \text{threadIdx.x}$



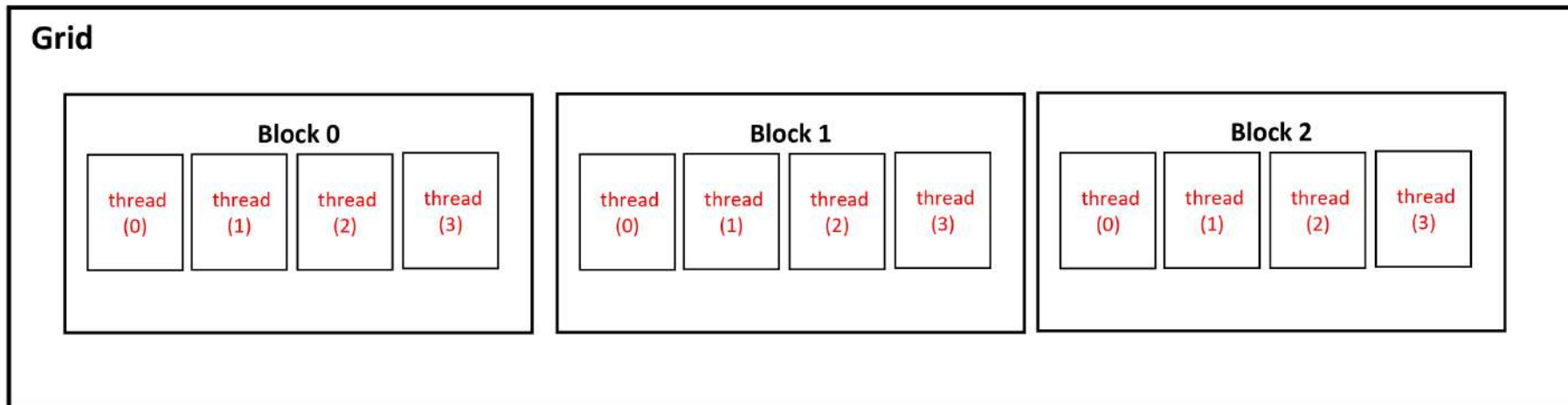
Thread indexing : 1D grid of 1D blocks

- $\text{threadId} = (\text{blockIdx.x} * \text{blockDim.x}) + \text{threadIdx.x}$



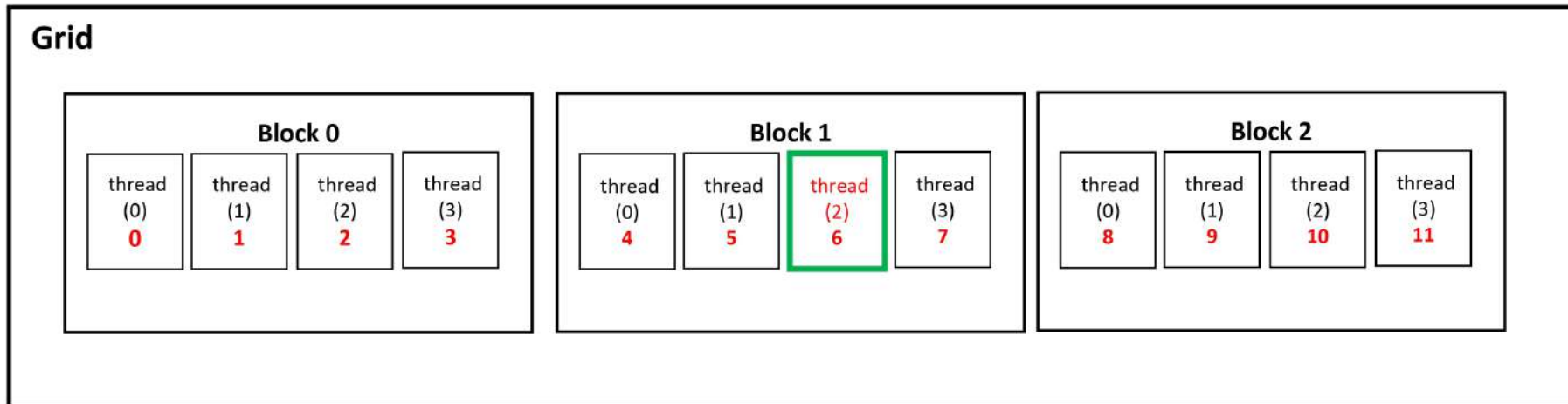
Thread indexing : 1D grid of 1D blocks

- $\text{threadId} = (\text{blockIdx.x} * \text{blockDim.x}) + \text{threadIdx.x}$



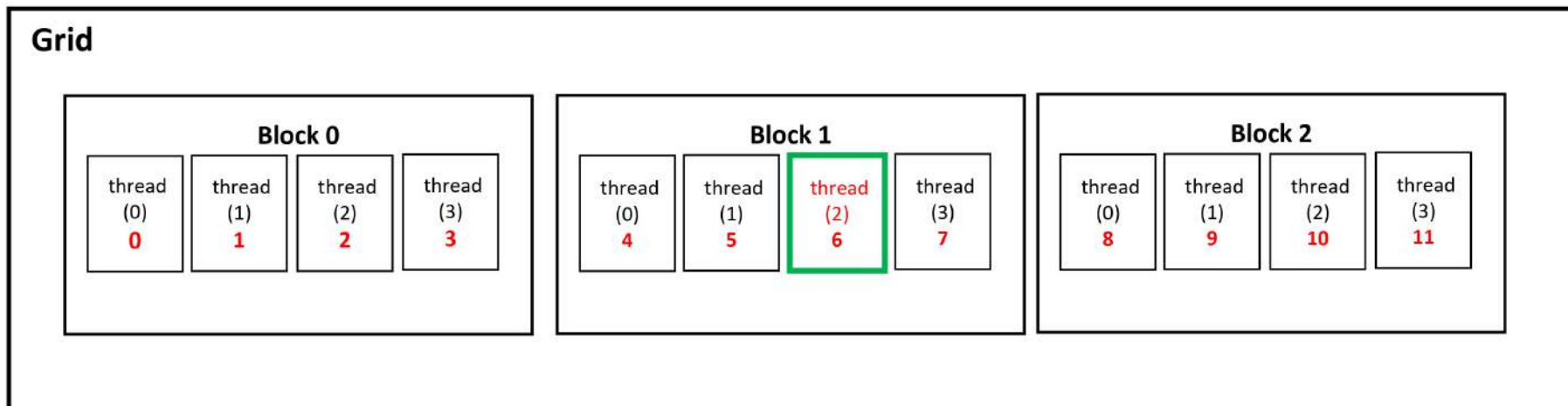
Thread indexing : 1D grid of 1D blocks

- $\text{threadId} = (\text{blockIdx.x} * \text{blockDim.x}) + \text{threadIdx.x}$



Thread indexing : 1D grid of 1D blocks

- $\text{threadId} = (\text{blockIdx.x} * \text{blockDim.x}) + \text{threadIdx.x}$
- $\text{threadId} = (1 * 4) + 2 = 4 + 2 = 6$



Example-1

- `dkernel<<<4,5>>>(A); //kernel launch`

```
__global__ void dkernel(int *A)
{
    int i=blockIdx.x * blockDim.x + threadIdx.x
    A[i]=i;
}
```

Example-1

- `dkernel<<<4,5>>>(A); //kernel launch`

```
__global__ void dkernel(int *A)
{
    int i=blockIdx.x * blockDim.x + threadIdx.x
    A[i]=i;
}
```

Output :

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19
```

Example-2

- `dkernel<<<4,5>>>(A); //kernel launch`

```
__global__ void dkernel(int *A)
{
    int i=blockIdx.x * blockDim.x + threadIdx.x
    A[i]=blockDim.x;
}
```


Example-2

- `dkernel<<<4,5>>>(A); //kernel launch`

```
__global__ void dkernel(int *A)
{
    int i=blockIdx.x * blockDim.x + threadIdx.x
    A[i]=blockDim.x;
}
```

Output :

5
5
5
.. 20 times

Example-3

- `dkernel<<<4,5>>>(A); //kernel launch`

```
__global__ void dkernel(int *A)
{
    int i=blockIdx.x * blockDim.x + threadIdx.x
    A[i]=blockIdx.x;
}
```

Example-3

- `dkernel<<<4,5>>>(A); //kernel launch`

```
__global__ void dkernel(int *A)
{
    int i=blockIdx.x * blockDim.x + threadIdx.x
    A[i]=blockIdx.x;
}
```

Output :

0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3

Example-4

- `dkernel<<<4,5>>>(A); //kernel launch`

```
__global__ void dkernel(int *A)
{
    int i=blockIdx.x * blockDim.x + threadIdx.x
    A[i]=threadIdx.x;
}
```

Example-4

- `dkernel<<<4,5>>>(A); //kernel launch`

```
__global__ void dkernel(int *A)
{
    int i=blockIdx.x * blockDim.x + threadIdx.x
    A[i]=threadIdx.x;
}
```

Output :

0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4

Thread indexing : 1D grid of 2D blocks

- `dkernel<<< 1D, 2D>>>();`

Thread indexing : 1D grid of 2D blocks

- `dkernel<<< 1D, 2D>>>();`
- `dim3 block(N, M, 1);` *// N= 5 , M=4*

Thread indexing : 1D grid of 2D blocks

- `dkernel<<<1D, 2D>>>();`
- `dim3 block(N, M, 1); // N= 5 , M=4`
- `dkernel<<<(1,1,1), (5,4,1)>>>();`

Thread indexing : 1D grid of 2D blocks

- `dkernel<<<(1,1,1), (5,4,1)>>>();`
- `gridDim.x = gridDim.y = gridDim.z = 1`

Thread indexing : 1D grid of 2D blocks

- `dkernel<<<(1,1,1), (5,4,1)>>>();`
- `gridDim.x = gridDim.y = gridDim.z = 1`
- `blockDim.x = 5 , blockDim.y = 4 , blockDim.z = 1`

Thread indexing : 1D grid of 2D blocks

- $\text{blockIdx.x} = \text{blockIdx.y} = \text{blockIdx.z} = 0$

Thread indexing : 1D grid of 2D blocks

- $\text{blockIdx.x} = \text{blockIdx.y} = \text{blockIdx.z} = 0$
- $\text{threadIdx.x} = 0 \text{ to } 4$
- $\text{threadIdx.y} = 0 \text{ to } 3$
- $\text{threadIdx.z} = 0$

Thread indexing : 1D grid of 2D blocks

- $\text{unsigned id} = \text{threadIdx.x} * \text{blockDim.y} + \text{threadIdx.y} ;$
 $= (0..4) * 4 + (0..3)$

Output :

```
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
```

Example-5

- dim3 block(3, 4, 1);
- K<<<1, block>>>();

```
__global__ void dkernel(int *A)
{
    printf("%d\n", threadIdx.x + threadIdx.y);
}
```

Example-5

- dim3 block(3, 4, 1);
- K<<<1, block>>>();

```
__global__ void dkernel(int *A)
{
    printf("%d\n", threadIdx.x + threadIdx.y);
}
```

Output :

0 1 2 1 2 3 2 3 4 3 4 5

Example-6

- `dim3 grid(2, 3, 4); dim3 block(5, 6, 7);`
- `dkernel <<< grid, block>>>();`

```
__global__ void dkernel() {  
    if(threadIdx.x == 0 && blockIdx.x == 0 && threadIdx.y == 0 &&  
        blockIdx.y == 0 && threadIdx.z == 0 && blockIdx.z == 0)  
    {  
        printf("%d %d %d %d %d %d.\n", gridDim.x, gridDim.y, gridDim.z,  
            blockDim.x, blockDim.y, blockDim.z);  
    }  
}
```


Example-6

- dim3 grid(2, 3, 4); dim3 block(5, 6, 7);
- dkernel <<< grid, block>>>();

```
__global__ void dkernel() {  
    if(threadIdx.x == 0 && blockIdx.x == 0 && threadIdx.y == 0 &&  
        blockIdx.y == 0 && threadIdx.z == 0 && blockIdx.z == 0)  
    {  
        printf("%d %d %d %d %d %d.\n", gridDim.x, gridDim.y, gridDim.z,  
            blockDim.x, blockDim.y, blockDim.z);  
    }  
}
```

Number of threads launched = 2 *

3 * 4 * 5 * 6 * 7

Example-6

- `dim3 grid(2, 3, 4); dim3 block(5, 6, 7);`
- `dkernel <<< grid, block>>>();`

```
__global__ void dkernel() {  
    if(threadIdx.x == 0 && blockIdx.x == 0 && threadIdx.y == 0 &&  
        blockIdx.y == 0 && threadIdx.z == 0 && blockIdx.z == 0)  
    {  
        printf("%d %d %d %d %d %d.\n", gridDim.x, gridDim.y, gridDim.z,  
            blockDim.x, blockDim.y, blockDim.z);  
    }  
}
```

**Number of threads launched = 2 *
3 * 4 * 5 * 6 * 7.**

**Number of threads in a thread-
block = 5 * 6 * 7**

Example-6

- `dim3 grid(2, 3, 4); dim3 block(5, 6, 7);`
- `dkernel <<< grid, block>>>();`

```
__global__ void dkernel() {  
    if(threadIdx.x == 0 && blockIdx.x == 0 && threadIdx.y == 0 &&  
        blockIdx.y == 0 && threadIdx.z == 0 && blockIdx.z == 0)  
    {  
        printf("%d %d %d %d %d %d.\n", gridDim.x, gridDim.y, gridDim.z,  
            blockDim.x, blockDim.y, blockDim.z);  
    }  
}
```

Number of threads launched = $2 * 3 * 4 * 5 * 6 * 7$.

Number of threads in a thread-block = $5 * 6 * 7$

Number of thread-blocks in the grid = $2 * 3 * 4$

Example-7

- dim3 grid(6, 7, 8); dim3 block(2, 3, 4);
- dkernel <<< grid, block>>>();

```
__global__ void dkernel() {  
    if(threadIdx.x == 0 && blockIdx.x == 0 && threadIdx.y == 0 &&  
        blockIdx.y == 0 && threadIdx.z == 0 && blockIdx.z == 0)  
    {  
        printf("%d %d %d %d %d %d.\n", gridDim.x, gridDim.y, gridDim.z,  
            blockDim.x, blockDim.y, blockDim.z);  
    }  
}
```

Output:

6 7 8 2 3 4

Thank You