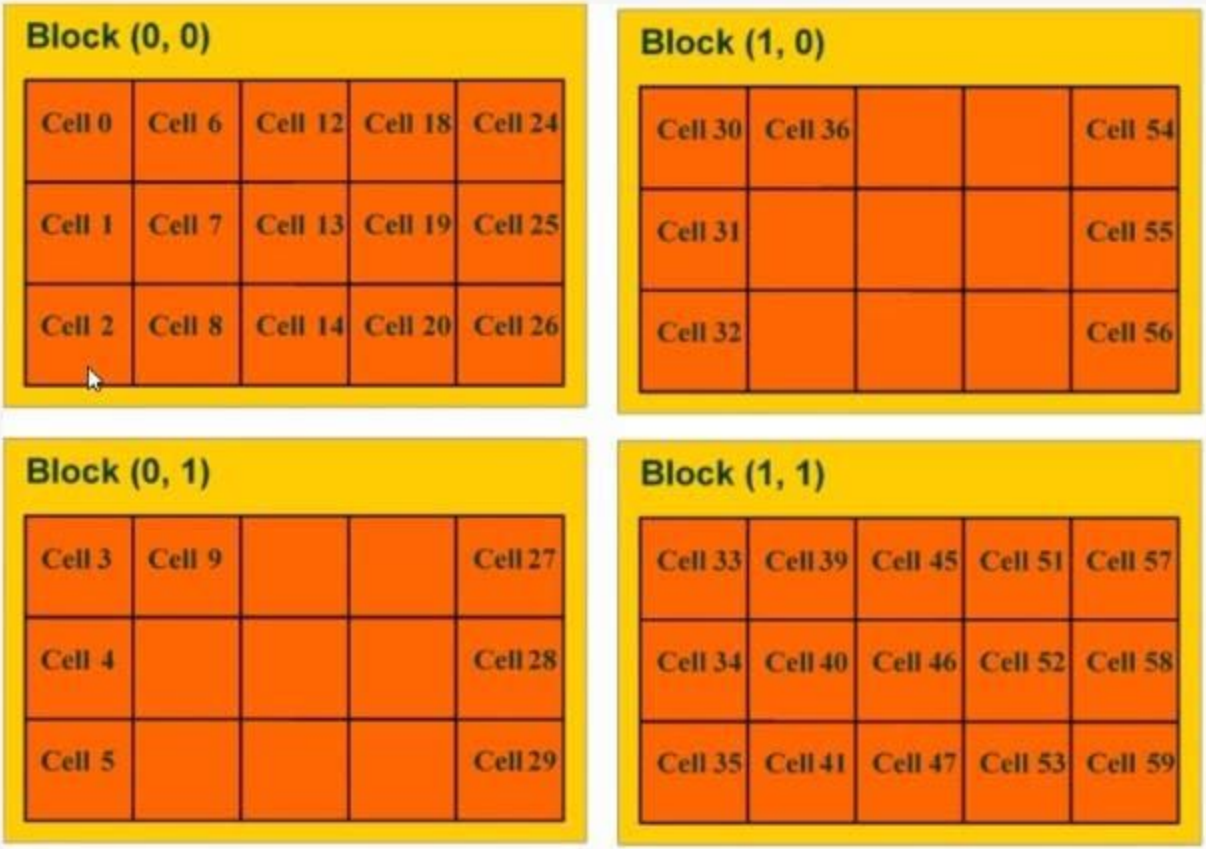
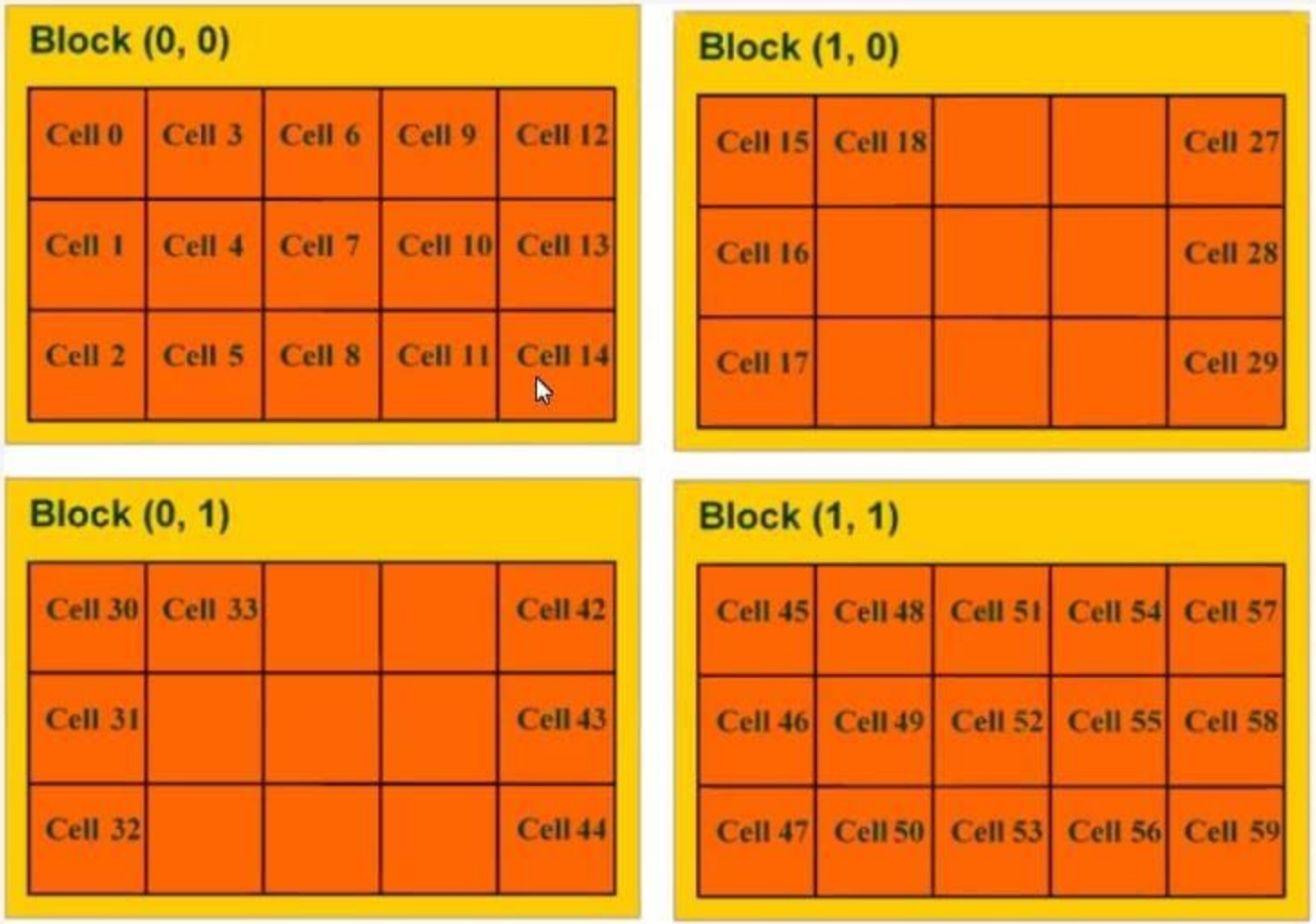



```
cell_ID = threadIdx.y + blockIdx.y *  blockDim.y
+ (threadIdx.x  * (gridDim.y * blockDim.y)
+ (blockIdx.x * gridDim.y ) * (blockDim.x * blockDim.y);
```

We would like to run a kernel on grid configured as M * N matrix of thread blocks. Every thread handles only one cell. Give the statement that calculates the ***cell_id*** for each thread as shown in the following figure:




```
cell_ID = threadIdx.y + threadIdx.x * blockDim.y +
( blockIdx.y * gridDim.x + blockIdx.x )* (blockDim.x * blockDim.y);
```

Let's consider a (N by N) Matrix of integers . We assume that the space memory on the GPU device for the Matrix is already allocated.

Complete the following program to upload the elements of the Matrix from the host to the GPU device.

```
#define N 16
int main(void) {
  int *a;     // The host copie of the Matrix a
  int *d_a;  //  The device copie of the Matrix a
  int size = N * N * sizeof(int);

  // Allocate space for the host copie of a and setup input values
  a = (int *)malloc(size); random_ints(a, N * N);


cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
```

Give the statements that allow to allocate a space memory in the GPU device for a N by N Matrix of integers.

```
int size = N * N * sizeof(int);
int *d_a;
cudaMalloc((void **)&d_a, size);
```

Give the statements that displays the name et the number of multi-processors of the current GPU device .

```
cudaDeviceProp prop;
int dev ;
cudaGetDevice( &dev );
cudaGetDeviceProperties( &prop, dev);
printf( "Name: %s\n", prop.name );
printf( "Multiproc: %d\n", prop.multiProcessorCount );
```

Give the CUDA statements that allow to get the number of available GPU devises.

```
int count;
cudaGetDeviceCount( &count);
```