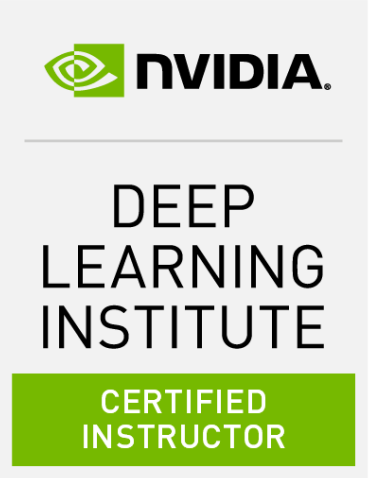
CUDA Programming – 2D Thread Organization

by
Dr. Nileshchandra Pikle
Assistant Professor
&
"A certified CUDA instructor by NVIDIA"



Thread Organization Extended to 2D

- **Threads can be organized in 2D or 3D**
- **dim3** is an integer vector type that can be used in CUDA code.
- Its most common application is to pass the grid and block dimensions in a kernel invocation.
- Eg.

```
dim3 grid(x, y, z)
```

Grid is a vector of **3 dimension** and of type **dim3**

Thread Organization Extended to 2D

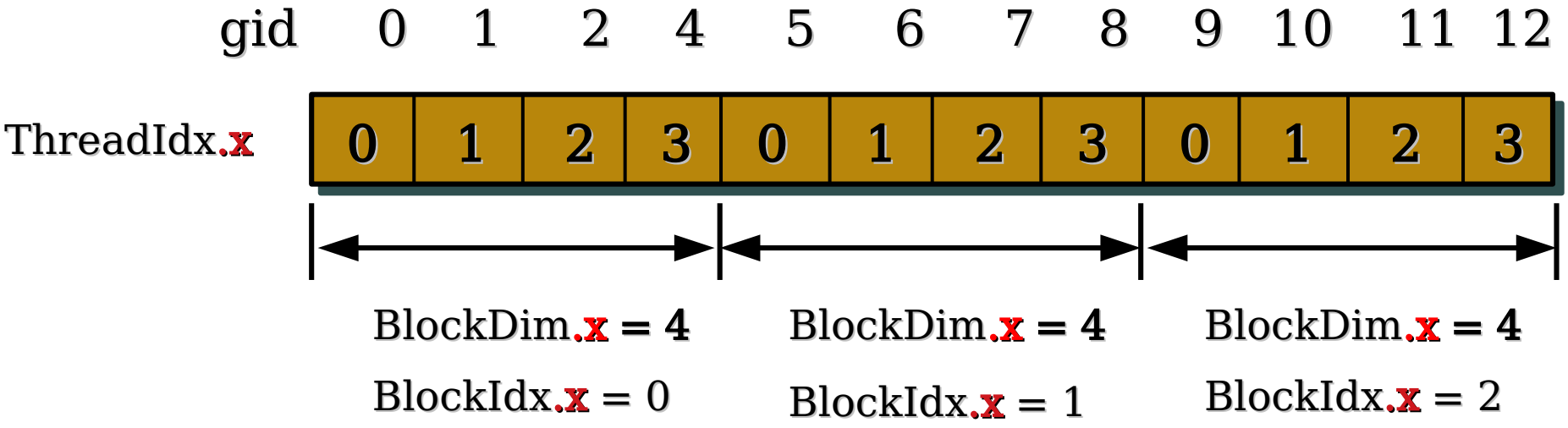
- **Threads can be organized in 2D or 3D**
- **dim3** is an integer vector type that can be used in CUDA code.
- Its most common application is to pass the grid and block dimensions in a kernel invocation.
- Eg.

```
dim3 grid( 512 );          // 512 x 1 x 1
dim3 block( 1024, 1024 ); // 1024 x 1024 x 1
fooKernel<<< grid, block >>>();
```

Thread Organization

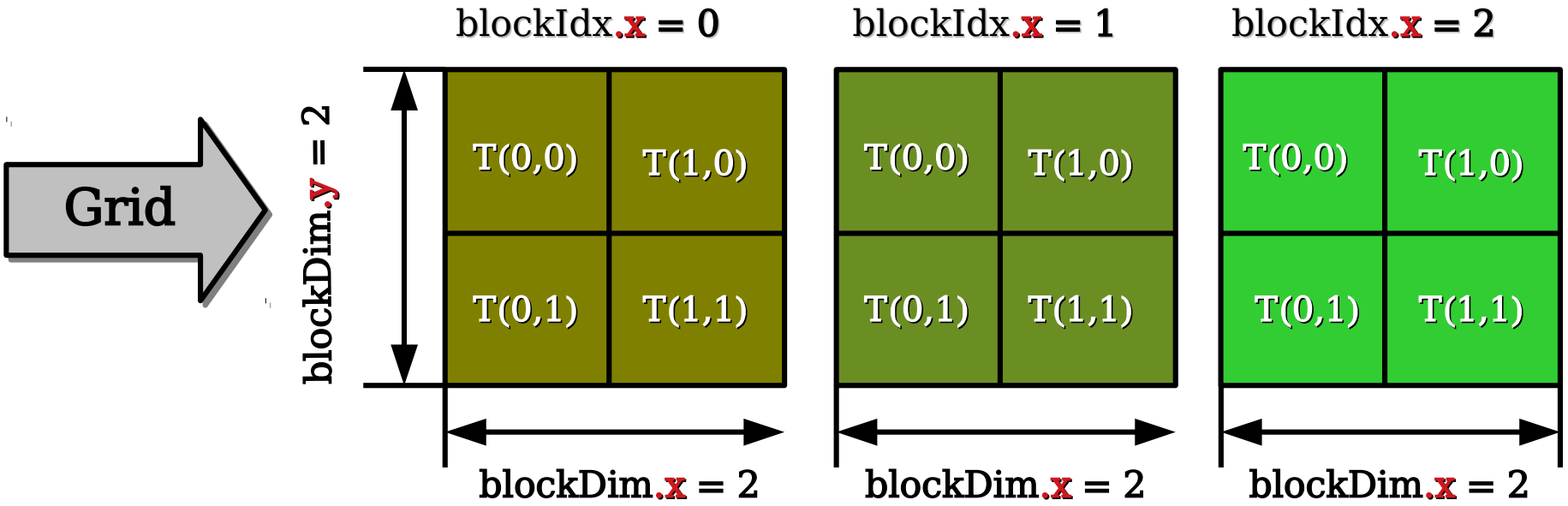
- 1D grid and 1D block

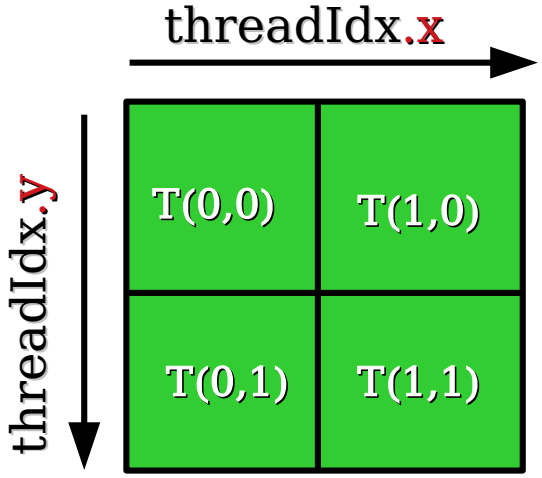
```
int gid = blockIdx.x *blockDim.x + threadIdx.x;
```



Thread Organization

- 1D grid and 2D block

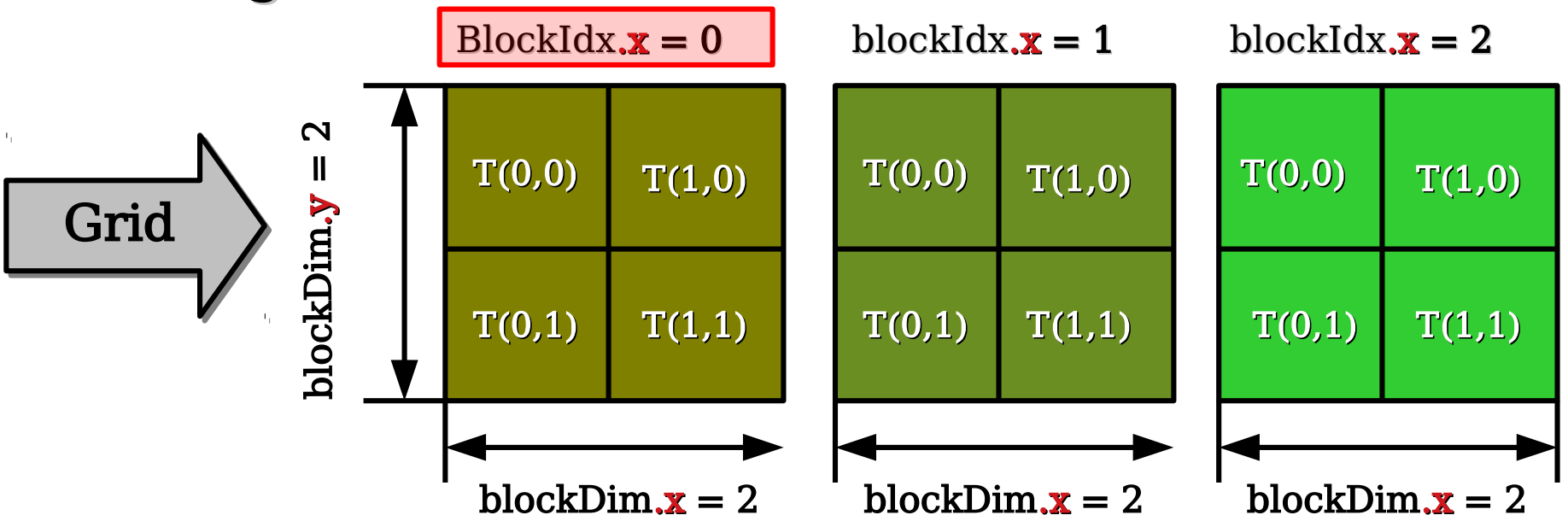


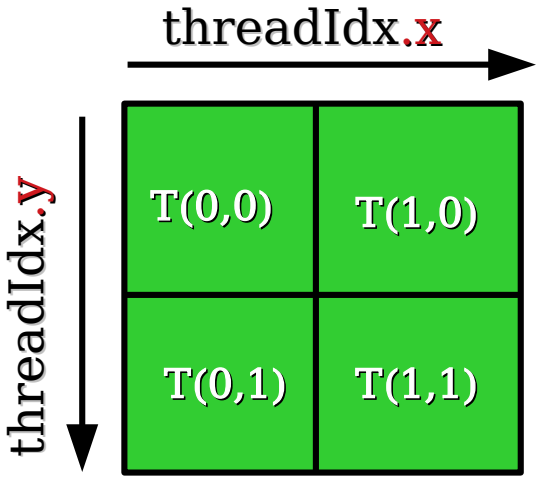



```
int gid = blockIdx.x * blockDim.x *blockDim.y
          threadIdx.y * blockDim.x + threadIdx.x;
```

Thread Organization

- 1D grid and 2D block

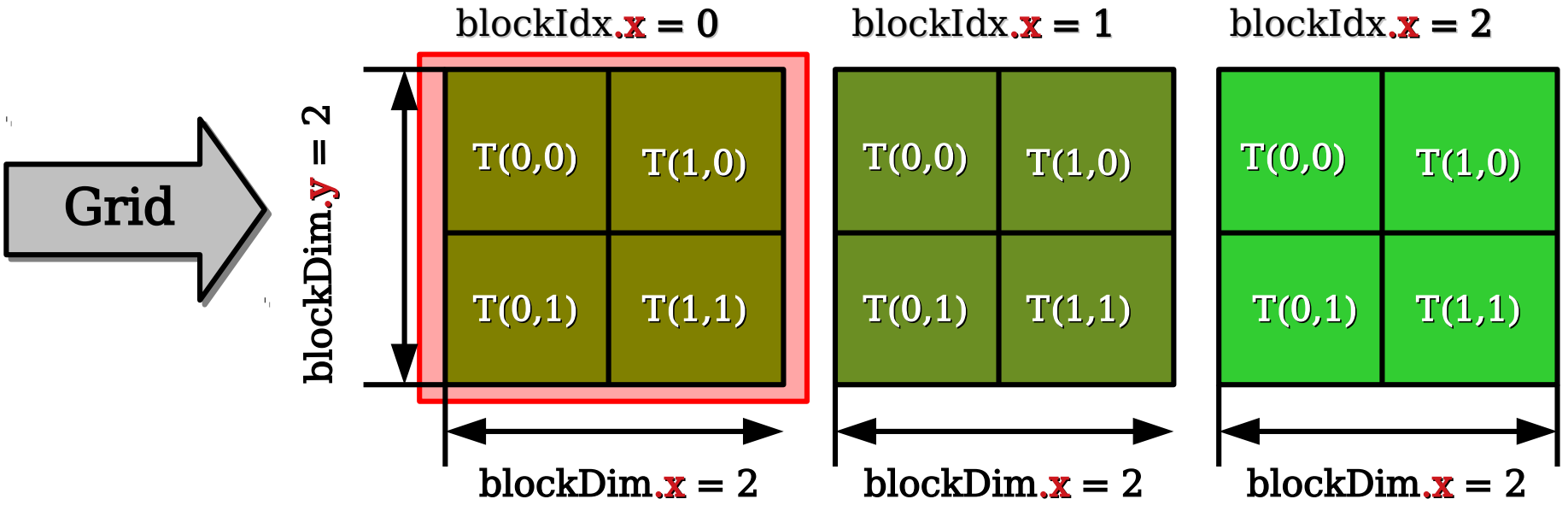


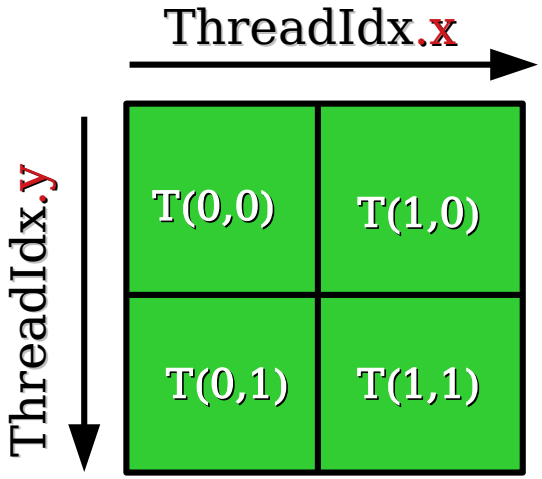



```
int gid = blockIdx.x * blockDim.x *blockDim.y
          threadIdx.y * blockDim.x + threadIdx.x;
```

Thread Organization

- 1D grid and 2D block

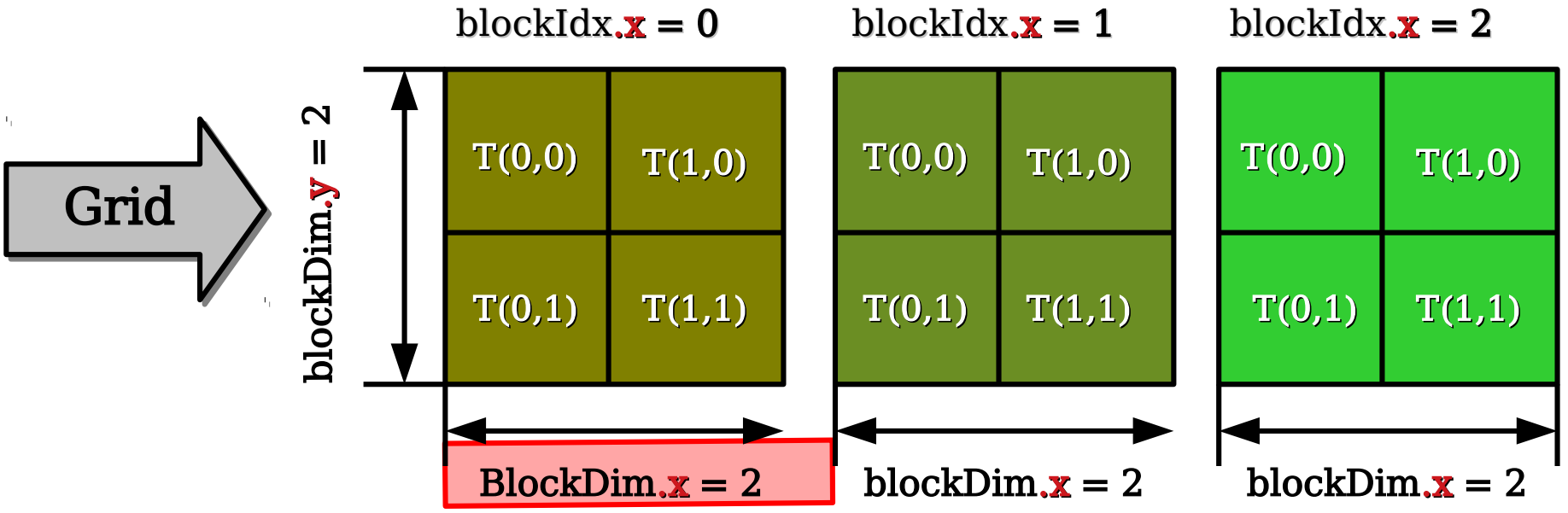


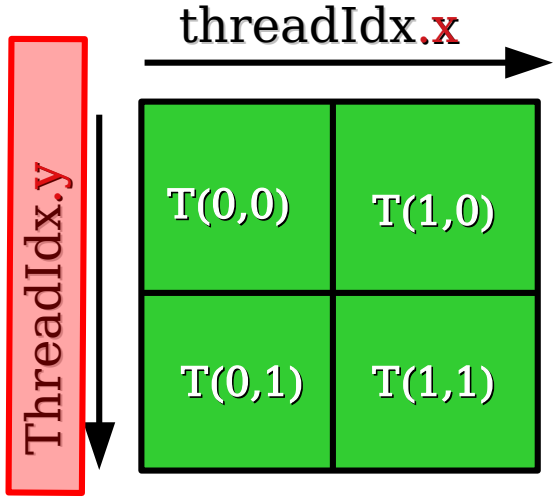



```
int gid = blockIdx.x * blockDim.x *blockDim.y
          threadIdx.y * blockDim.x + threadIdx.x;
```

Thread Organization

- 1D grid and 2D block

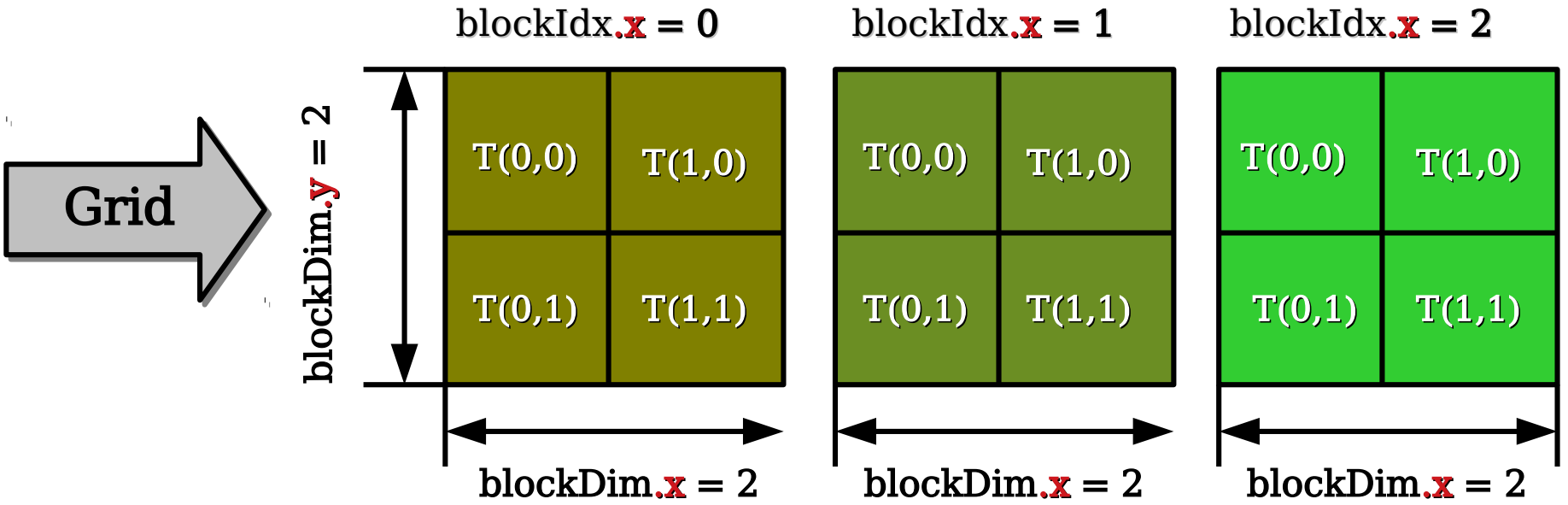


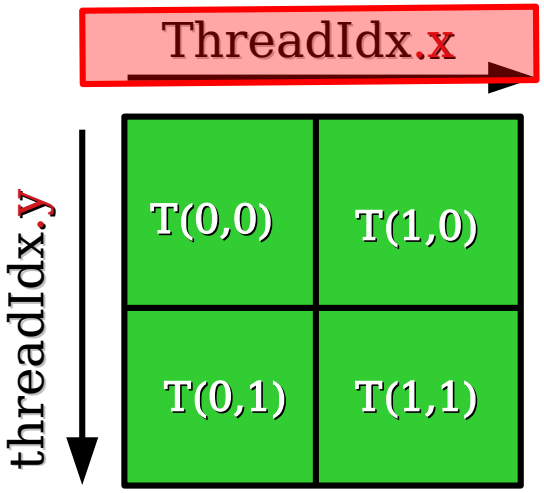



```
int gid = blockIdx.x * blockDim.x *blockDim.y
          threadIdx.y * blockDim.x + threadIdx.x;
```

Thread Organization

- 1D grid and 2D block

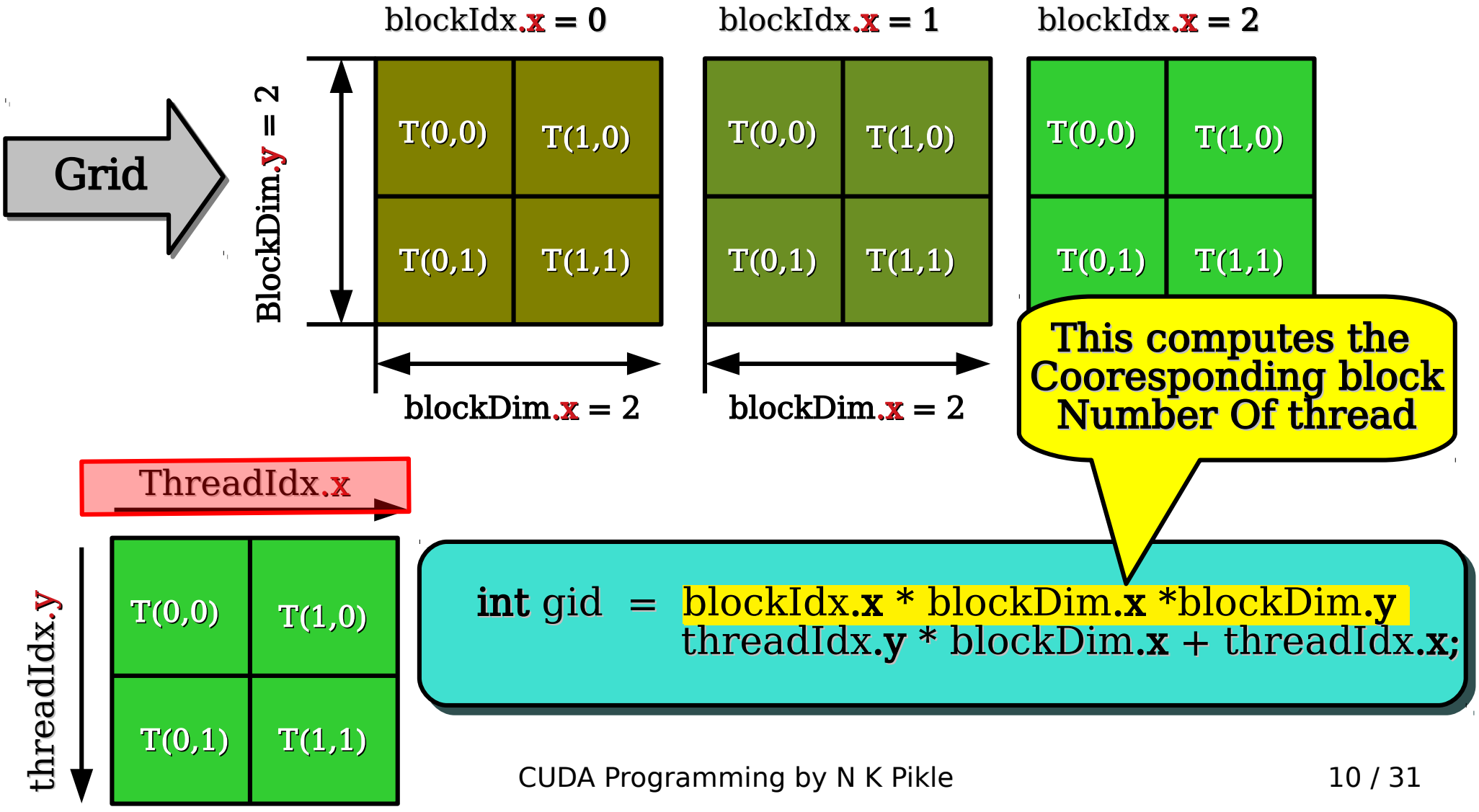





```
int gid = blockIdx.x * blockDim.x *blockDim.y
          threadIdx.y * blockDim.x + threadIdx.x;
```

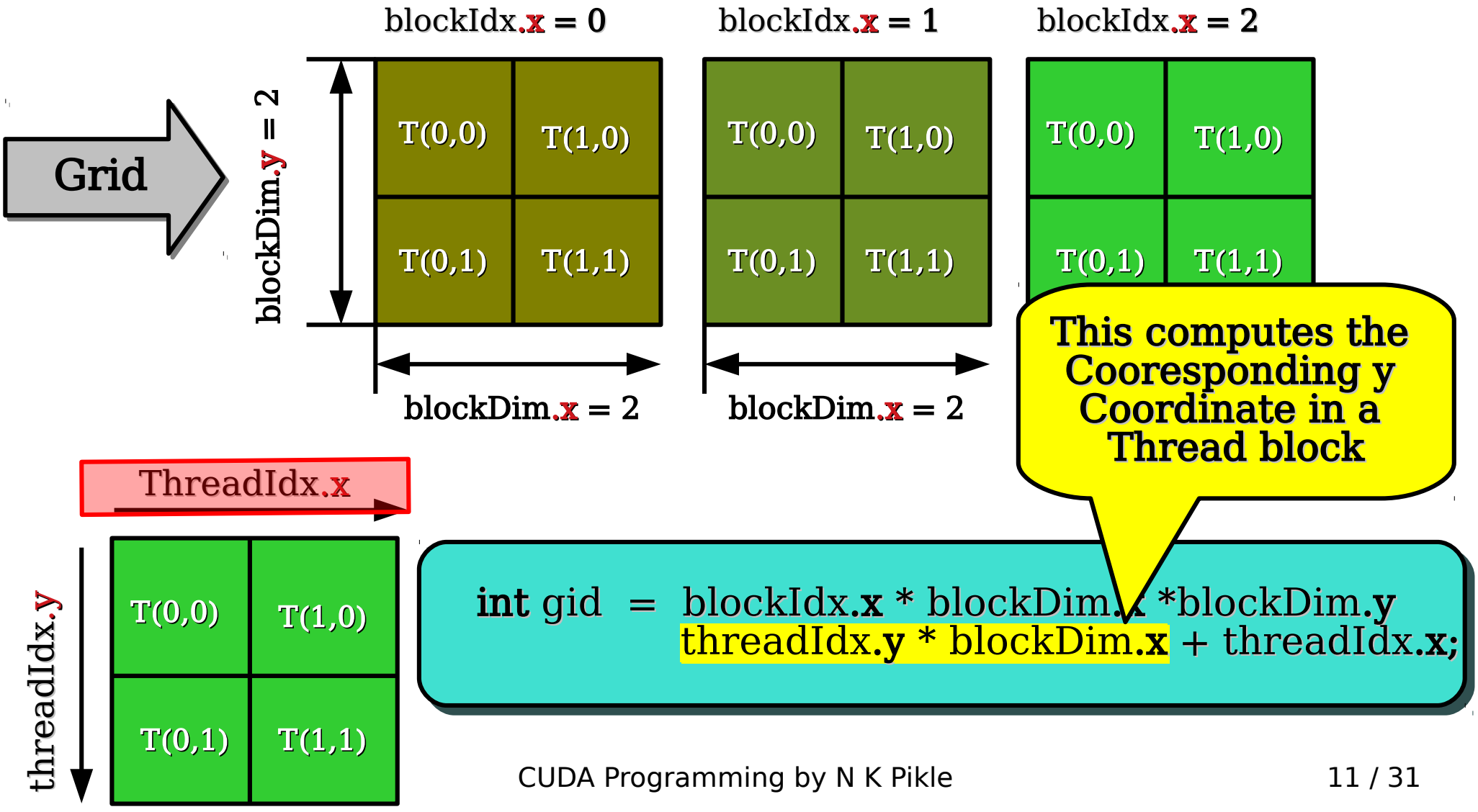
Thread Organization

- 1D grid and 2D block



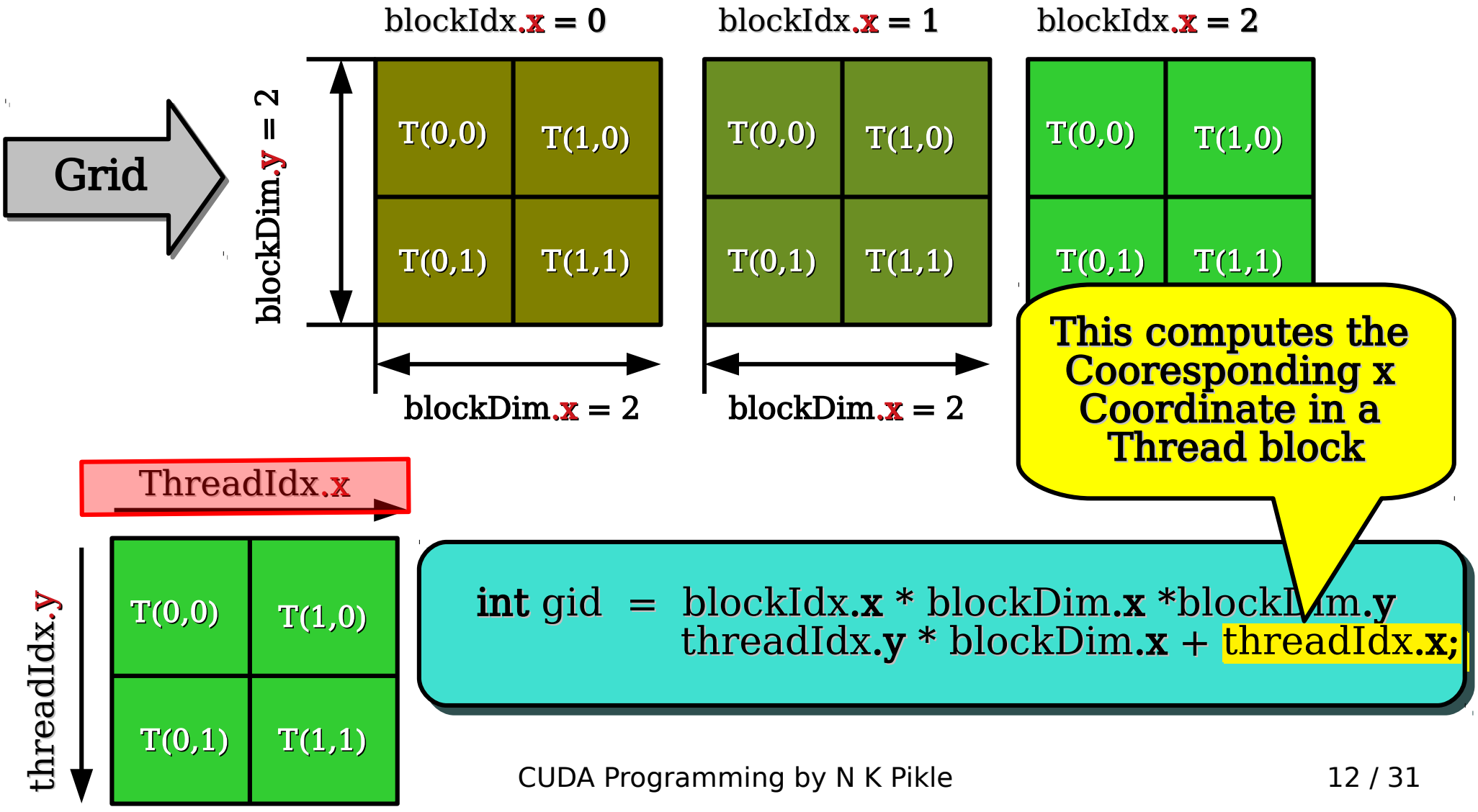
Thread Organization

- 1D grid and 2D block



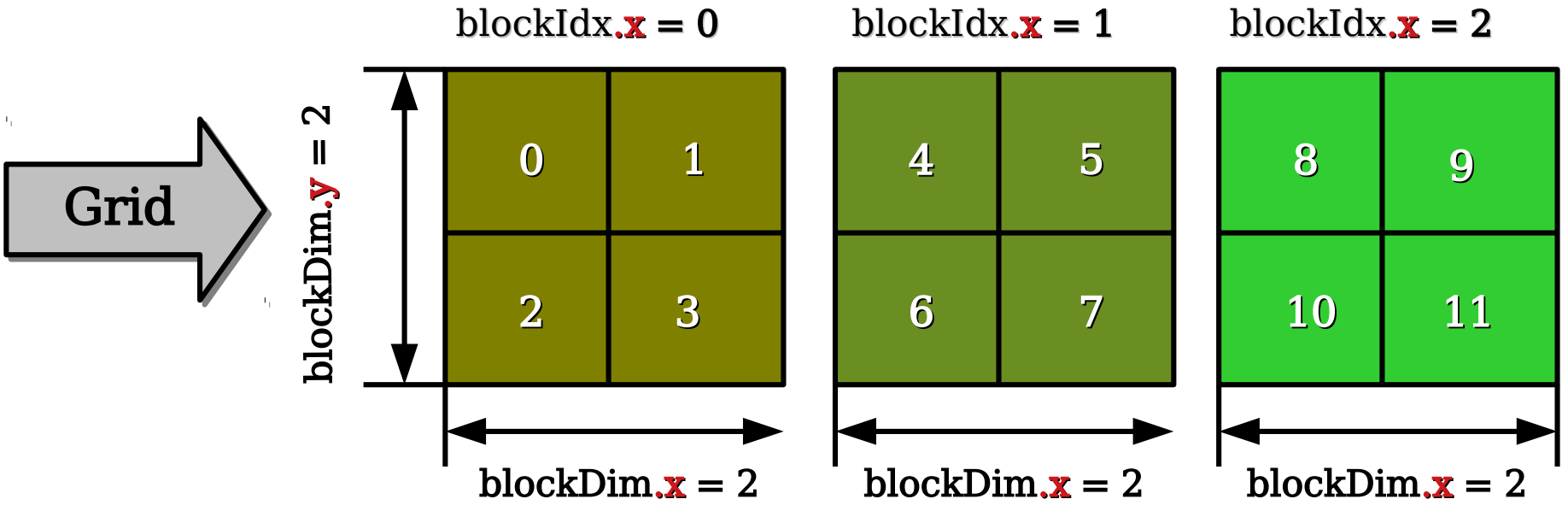
Thread Organization

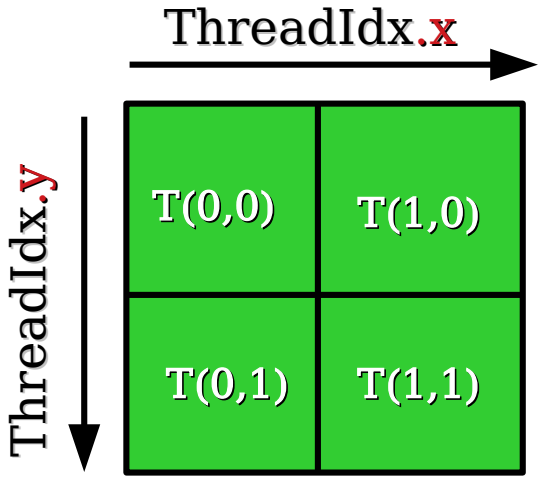
- 1D grid and 2D block



Thread Organization

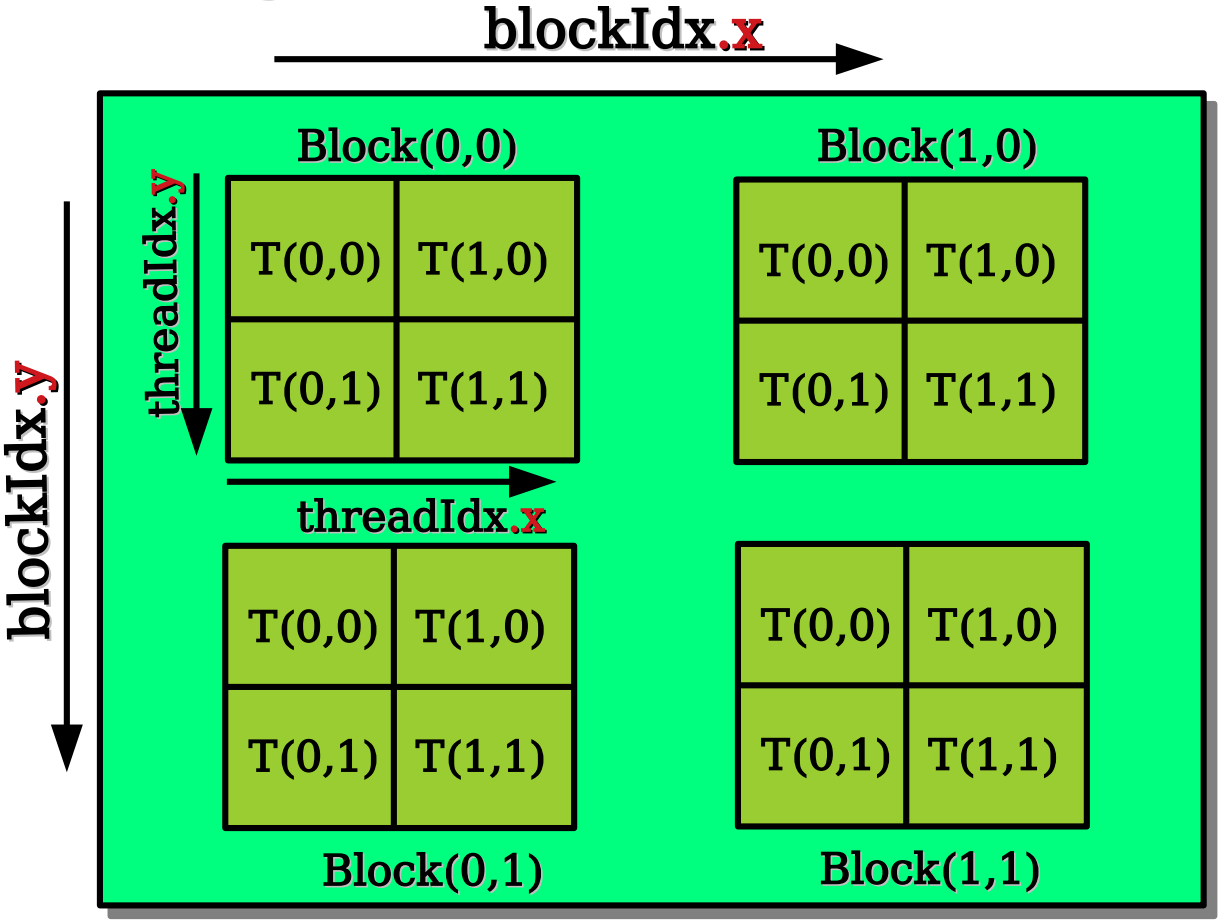
- 1D grid and 2D block





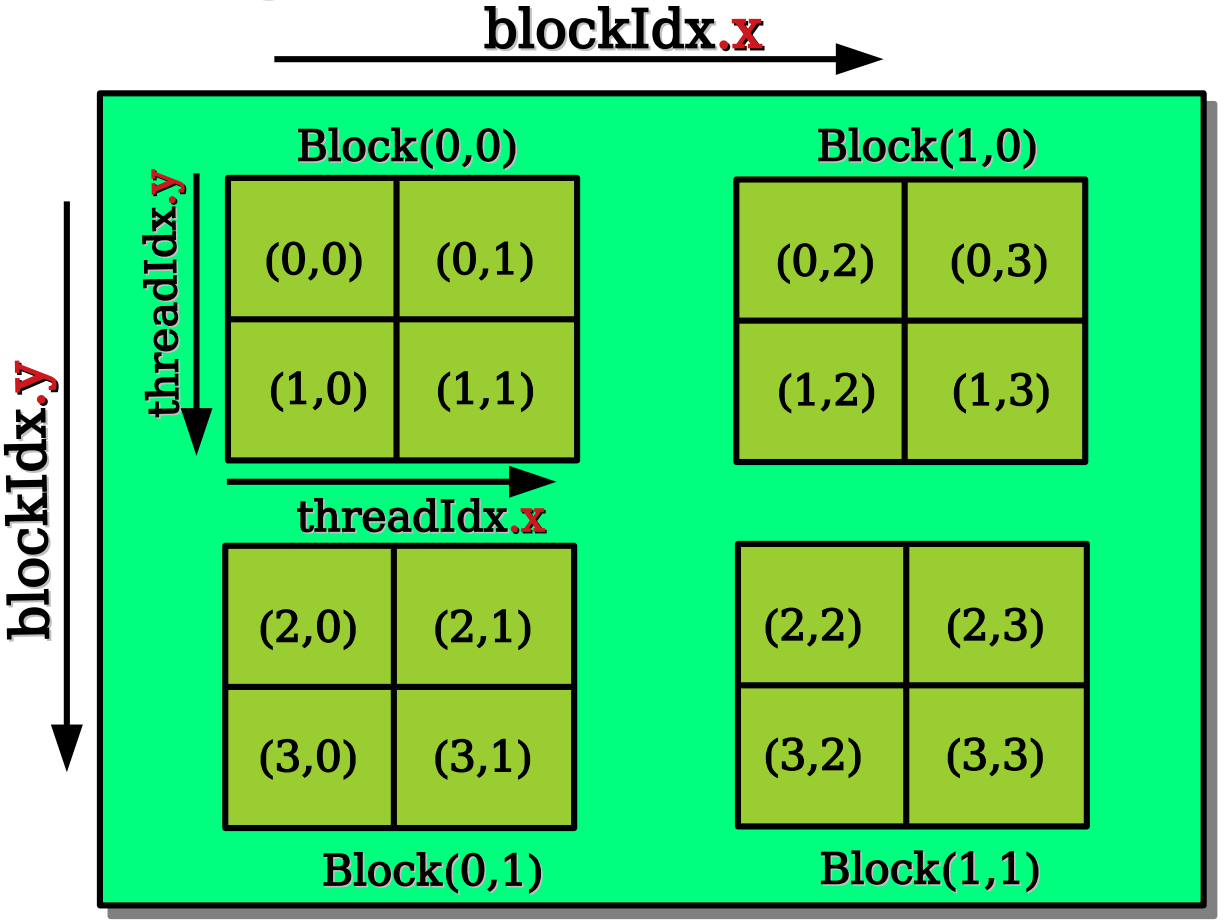

```
int gid = blockIdx.x * blockDim.x *blockDim.y
          threadIdx.y * blockDim.x + threadIdx.x;
```

Thread Organization 2D block & 2D grid



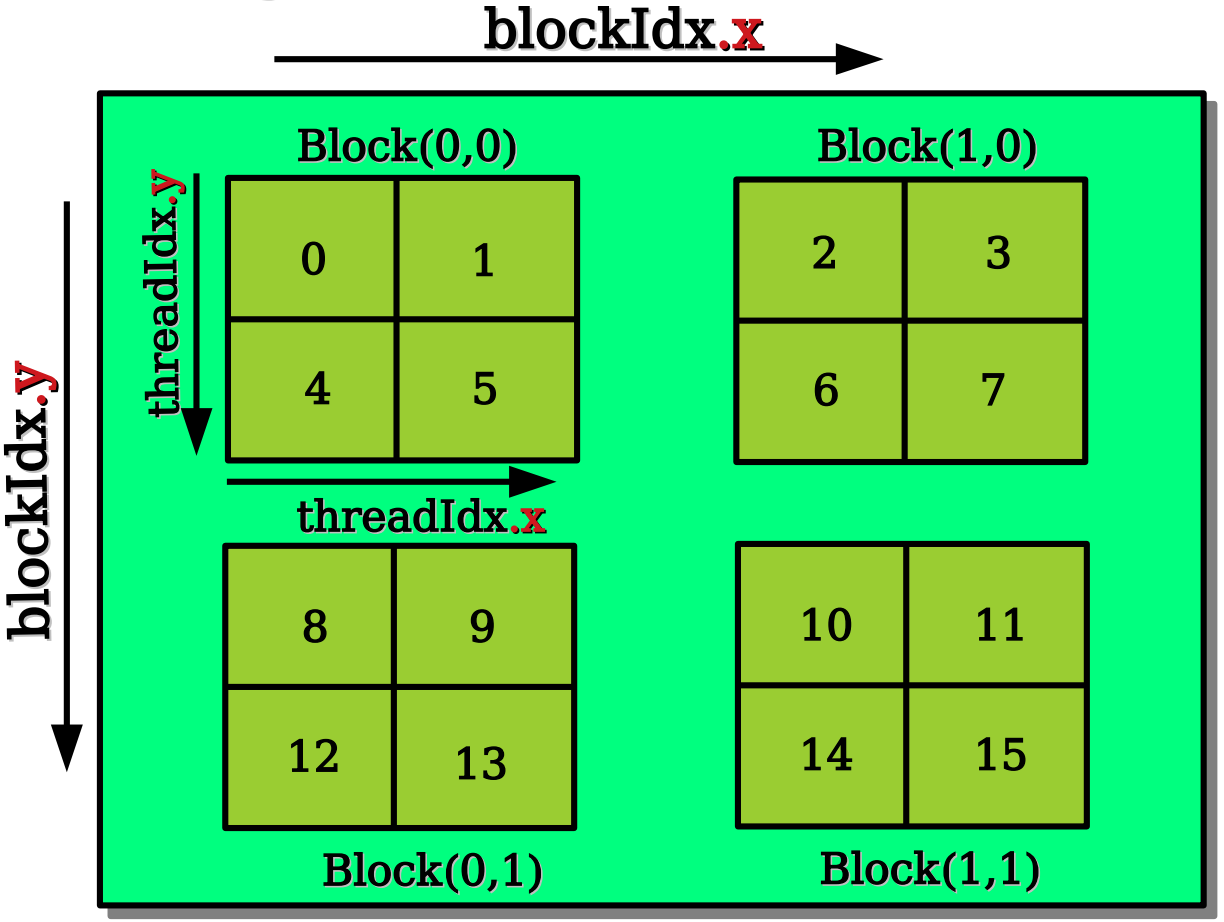

```
int row = threadIdx.y + blockIdx.y * blockDim.y
int col  = threadIdx.x + blockIdx.x * blockDim.x
```

Thread Organization 2D block & 2D grid




```
int row = threadIdx.y + blockIdx.y * blockDim.y
int col  = threadIdx.x + blockIdx.x * blockDim.x
```

Thread Organization 2D block & 2D grid




```
int gid = row * blockDim.x * gridDim.x + col
```

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

Input Image

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Program 2D convolution Parallel

To take care of boundary conditions either apply zeroPadding or avoid the out of range computations

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	4	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	1	6	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	1	1	6	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	1	1	1	6	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	1	1	1	1	4	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	6	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	9	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	9	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	9	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	1	6	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Program 2D convolution Parallel

Convolution mask

1	1	1
1	1	1
1	1	1

$$O[i][j] = \sum_{s=-a}^{+a} \sum_{t=-a}^{+a} mask[s][t] * A[i+s][j+t]$$

Final Output Image

4	6	6	6	4
6	9	9	9	6
6	9	9	9	6
6	9	9	9	6
4	6	6	6	4

Program 2D convolution Parallel

Refer program 2D_Convolution.cu

Program 2D convolution Parallel

- Summary

1. **CUDA programming model** – Heterogeneous Computing
2. **CUDA Thread organization:** Threads grouped together to form blocks and blocks to grid.
   -Threads can be organized as 1D, 2D or 3D.
3. **Case studies:** To understand basic CUDA programming model, thread and data mapping in 1D and 2D