CSISF301: Principles of Programming Languages

CUDA PROGRAMMING

Tutorial

BITS Pilani

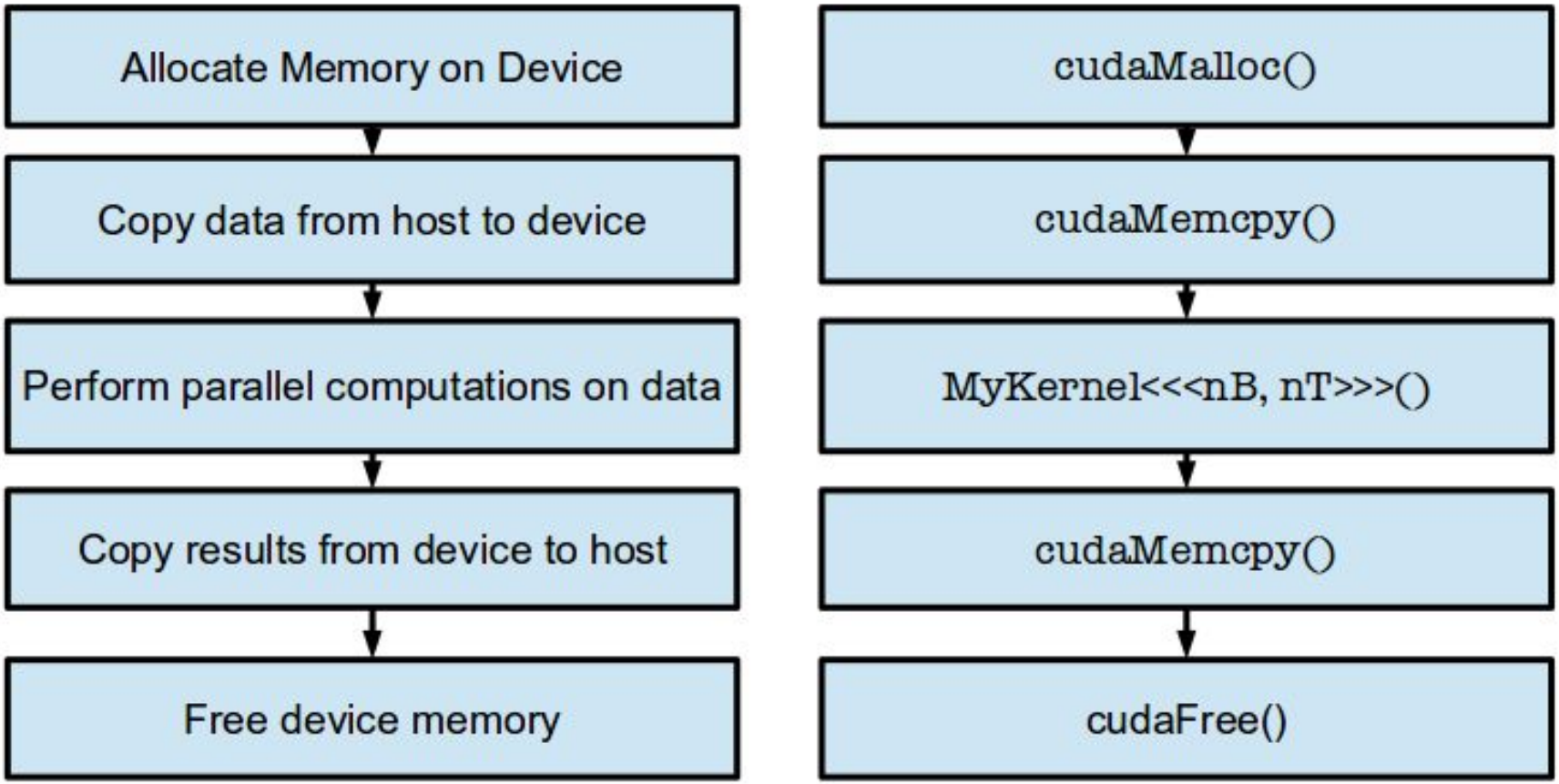
Pilani | Dubai | Goa | Hyderabad

August 6, 2015

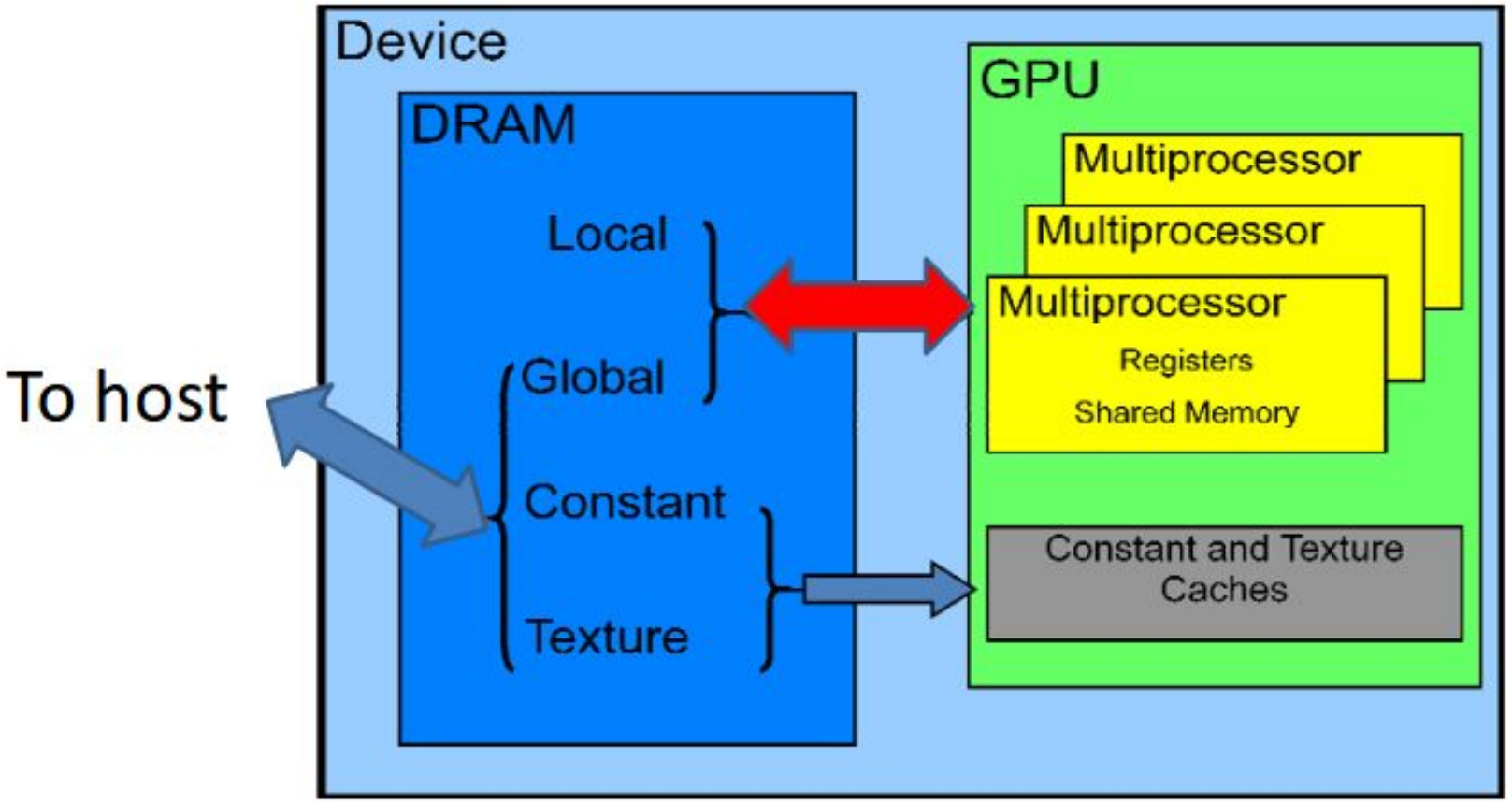
CUDA -RECAP

- Heterogeneous Computing
- Processing Flow
- Threads, Blocks, Grids, Warp
- CUDA Memory Model
- Kernel Declaration and Invocation
- Vector Addition Using CUDA

Simple CUDA Program



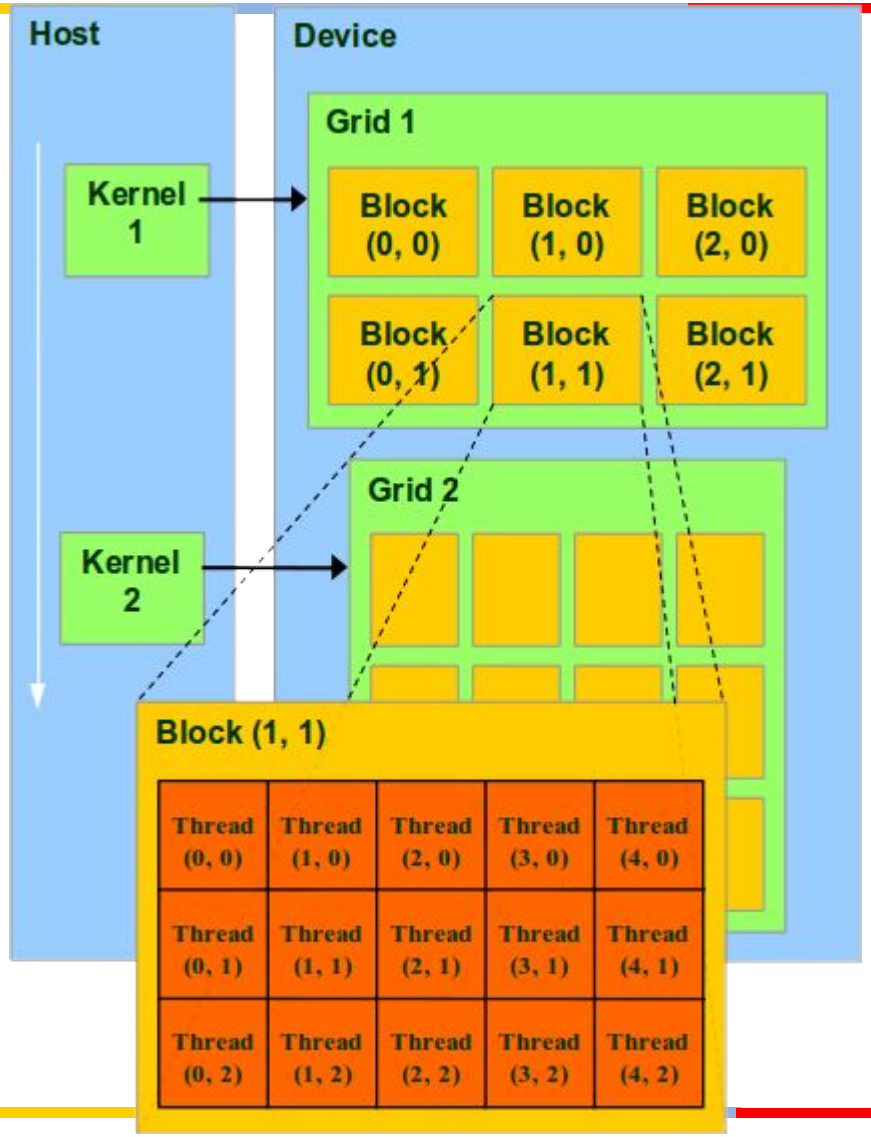
Memory Model in CUDA



Memory Model in CUDA

Memory	Location	Cached	Access	Scope	Lifetime
Register	On chip	N	R/W	1 thread	Thread
Local	RAM	N	R/W	1 thread	Thread
Shared	On chip	N	R/W	Threads in a block	Block
Global	RAM	N	R/W	All thread + host	Host allocation
Constant	RAM	Y	R	All thread + host	Host allocation
Texture	RAM	Y	R	All thread + host	Host allocation

Thread Batching: Grids and Blocks



GRID

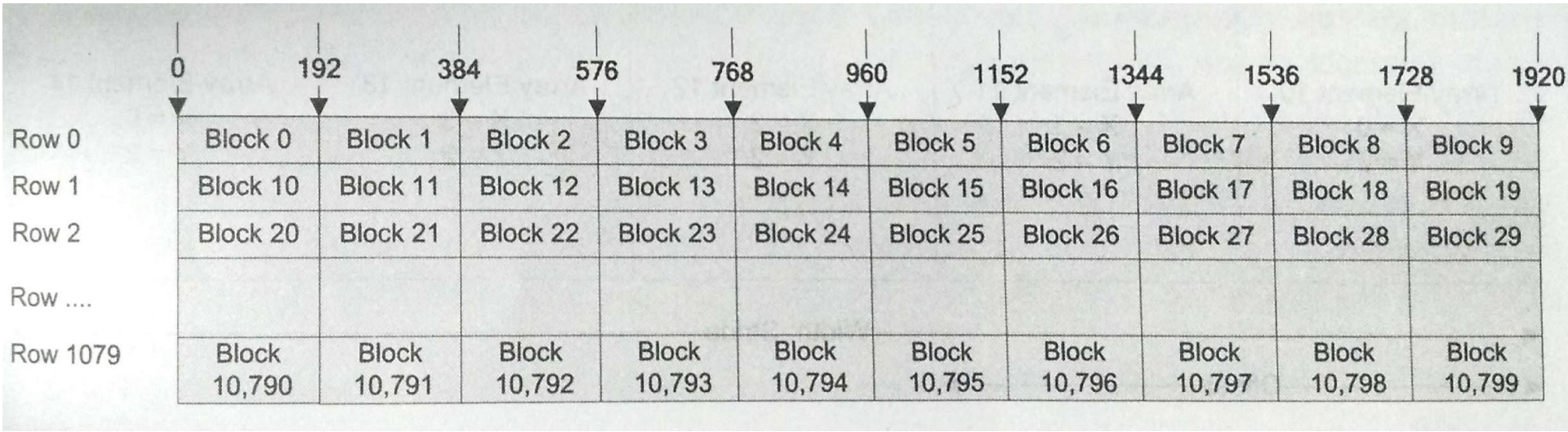
- Set of blocks where you have X and Y axis for 2D mapping.

Ex.

- HD image of 1920*1080 resolution
- Ideal scenario :
  - min 192 threads per block
  - Thread size of multiple of X axis

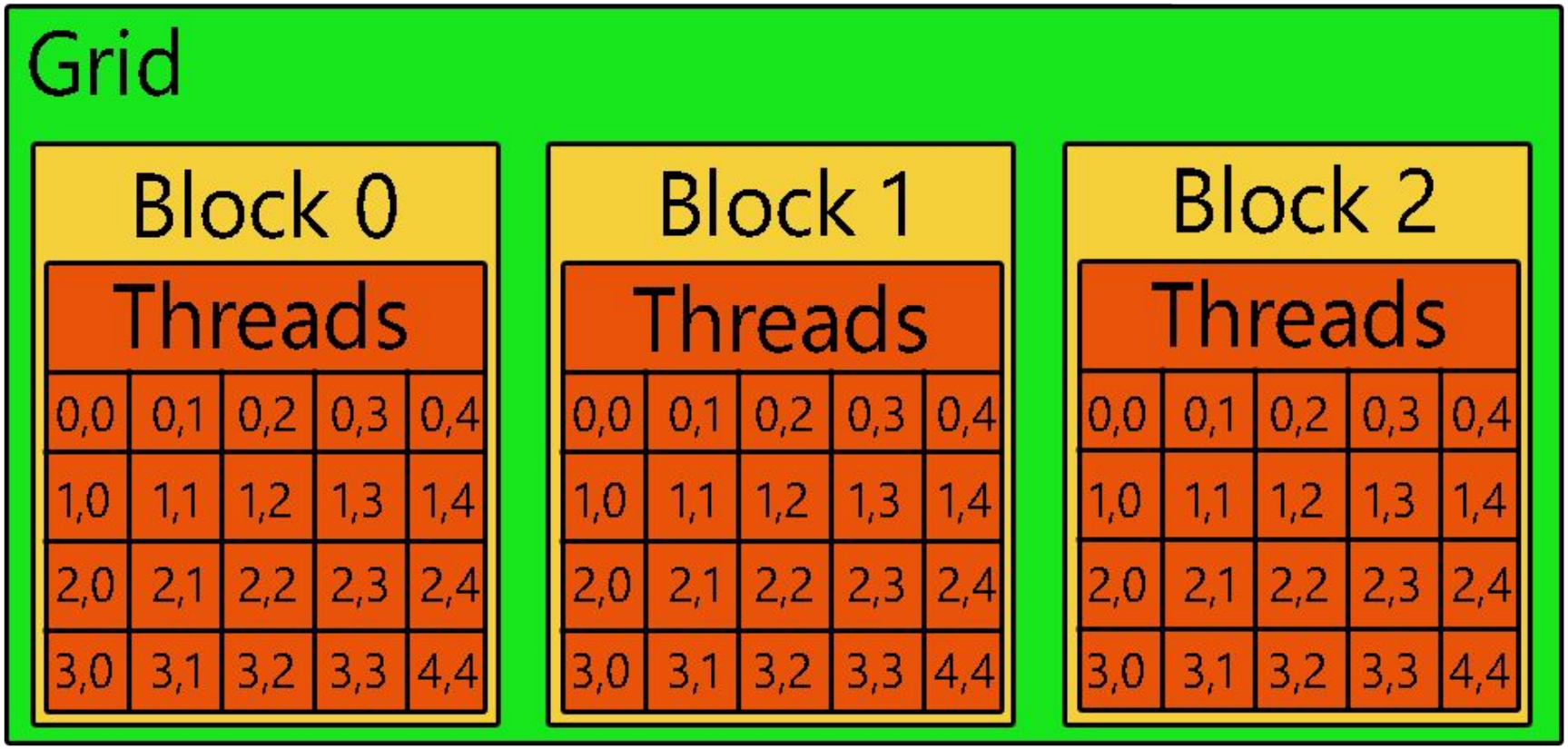
GRID

- HD image of 1920*1080 resolution
- Thread index along x axis, row index along y axis
- 1080 rows of 10 blocks (1080*10=10,800 blocks)
- 10800 blocks of 198 threads (10800*198=2073600 threads)
- One thread per pixel



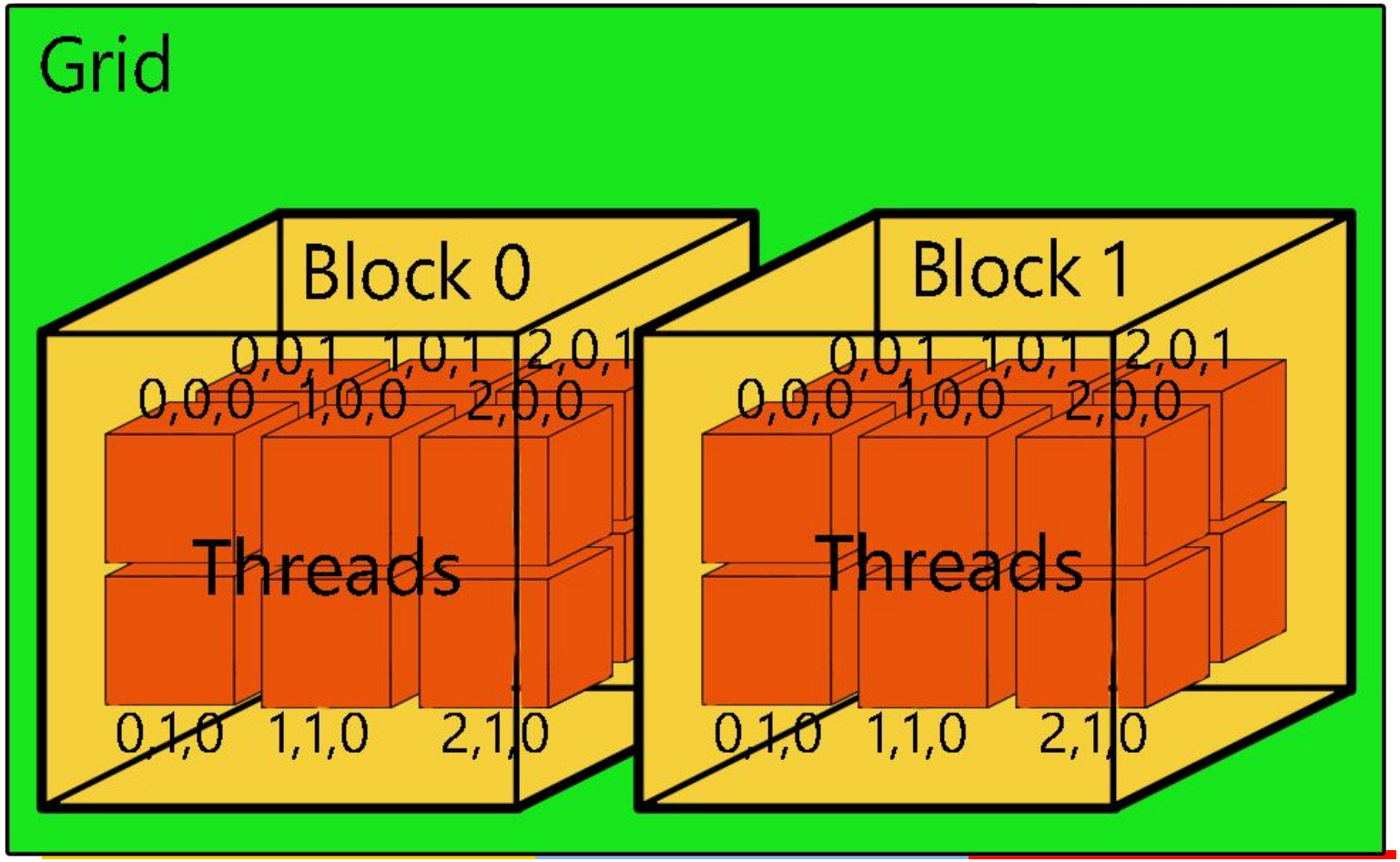
Grid

1D Grid of 2D Blocks



Grid

1D Grid of 3D Blocks



Executing a Kernel

Execution rules:

- All threads in a grid execute the same kernel function
- A grid is organized as a 2D array of blocks
- All blocks in a grid have the same dimension
- Total size of a block is limited to 512 or 1024 threads

Definitions:

- gridDim: This variable contains the dimensions of the grid (gridDim.x and gridDim.y)
- blockIdx: This variable contains the block index within the grid
- blockDim: This variable contains the dimensions of the block (blockDim.x, blockDim.y and blockDim.z)
- threadIdx: This variable contains the thread index within the block.

1D Grid of 1D Block

```
bx = cuda.blockIdx.x
bw = cuda.blockDim.x
tx = cuda.threadIdx.x
i = tx + bx * bw
```

2D Grid of 2D Blocks

```
tx = cuda.threadIdx.x
ty = cuda.threadIdx.y
bx = cuda.blockIdx.x
by = cuda.blockIdx.y
bw = cuda.blockDim.x
bh = cuda.blockDim.y
i = tx + bx * bw
j = ty + by * bh
```

3D Grid of 3D Blocks

```
tx = cuda.threadIdx.x
ty = cuda.threadIdx.y
tz = cuda.threadIdx.z
bx = cuda.blockIdx.x
by = cuda.blockIdx.y
bz = cuda.blockIdx.y
bw = cuda.blockDim.x
bh = cuda.blockDim.y
bd = cuda.blockDim.z
i = tx + bx * bw
j = ty + by * bh
k = tz + bz * hd
```

Example Vector Addition

```
#define N 500
#define NBLOCK 5
#define NTHREAD 10
__global__ void vec_adder(int n, float* a, float *b)
{
  int i;
  Int k=(N * blockIdx.x )/NBLOCK + (threadIdx.x*N)/(NBLOCK*NTHREAD);
   for (i=k;i<k+n;i++)
    {
          a[i] = a[i] + b[i];
    }
}
```

Vector Addition - Kernel Call

```
cudaMemcpy(gpu_a, host_a, sizeof(float) * n,
cudaMemcpyHostToDevice);
cudaMemcpy(gpu_b, host_b, sizeof(float) * n,
cudaMemcpyHostToDevice);
vec_adder<<<NBLOCK, NTHREAD>>>(N / (NBLOCK * NTHREAD),
  gpu_a, gpu_b);
cudaMemcpy(host_c, gpu_a, sizeof(float) * n,
cudaMemcpyDeviceToHost);
```

C Function

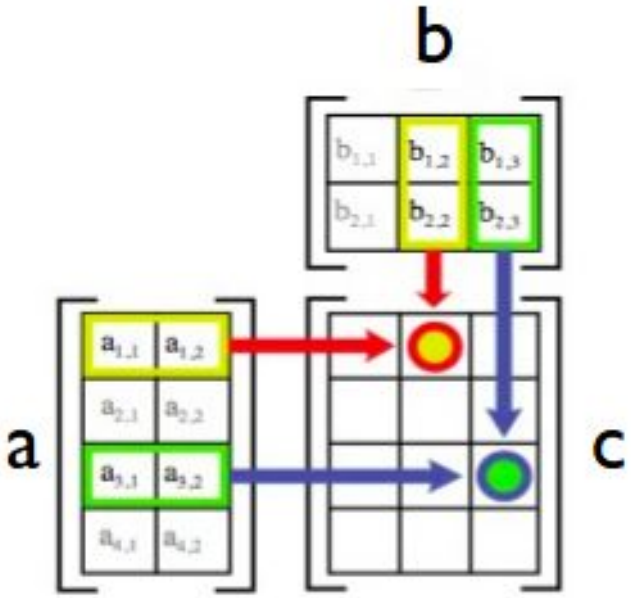
```
void matrixMult (int a[N][N], int b[N][N], int c[N][N], int width)
{
  for (int i = 0; i < width; i++) {
    for (int j = 0; j < width; j++) {
      int sum = 0;
      for (int k = 0; k < width; k++) {
        int m = a[i][k];
        int n = b[k][j];
        sum += m * n;
      }
      c[i][j] = sum;
    }
  }
}
```

CUDA Kernel

```
__global__ void matrixMult (int *a, int *b, int *c, int width) {
 int k, sum = 0;

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

 if(col < width && row < width) {
  for (k = 0; k < width; k++)
   sum += a[row * width + k] * b[k * width + col];
  c[row * width + col] = sum;
 }
}
```



Warps

- CUDA threads don't operate independently, they are in units called warps that operate in lock-step
- A warp consists of 32 threads
- Since threads in a warp act in lock-step…
  - When one thread accesses memory, they all access memory
  - If 2 threads in a warp diverge at an if statement, every thread evaluates both paths
    - They activate and deactivate to achieve the desired effects

How to run CUDA program?

- Login to Kosambi cluster
  - ssh popl_stud@10.1.9.230
  - Password: POPL_2017_18
- Go to node 02
  - ssh n02
- Load Cuda75 toolkit
  - module load cuda75/toolkit/7.5.18
- Compile Cuda code
  - nvcc vecAdd.cu
- Run the executable
  - ./a.out

THANK YOU