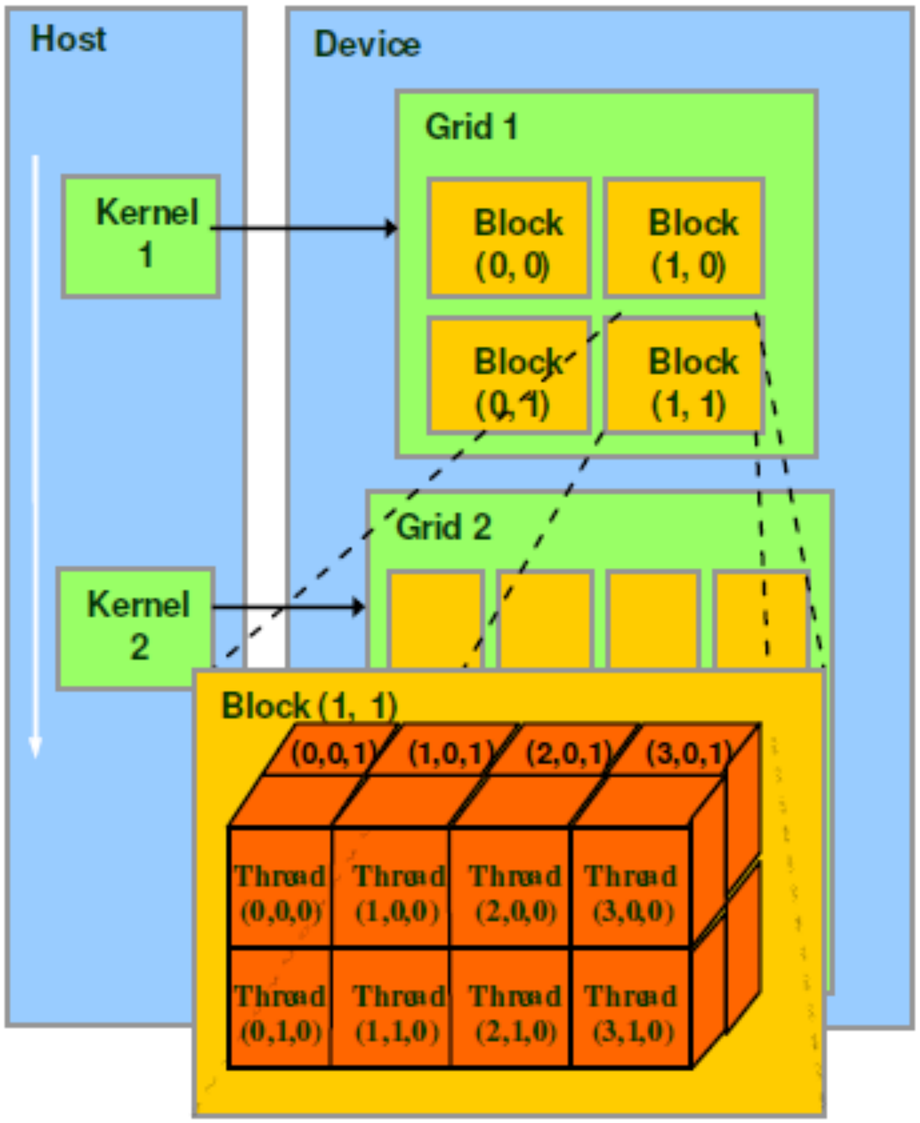
CUDA Thread Indexing Cheatsheet

If you are a CUDA parallel programmer but sometimes you cannot wrap your head around thread indexing just like me then you are at the right place.

Many problems are naturally described in a flat, linear style mimicking our mental model of C's memory layout. However, other tasks, especially those encountered in the computational sciences, are naturally embedded in two or three dimensions. For example, image processing tasks typically impose a regular 2D raster over the problem domain while computational fluid dynamics might be most naturally expressed by partitioning a volume over 3D grid.



NVIDIA CUDA Thread Model

Sometimes it can be a bit tricky to figure out the global (unique) thread index, especially if you are working with multi-dimensional grids of multi-dimensional blocks of threads. I could not really find a simple cheat-sheet that would demonstrate what exactly you need to do to calculate a global thread index for every configuration you might need to use. I know that with a little effort anyone can figure it out but I thought I would share some of my code with you to make your life easier. At the end of the day, sharing is caring :)

Download example code, which you can compile with nvcc simpleIndexing.cu -o simpleIndexing -arch=sm_20

1D grid of 1D blocks

```
__device__
int getGlobalIdx_1D_1D(){
      return blockIdx.x *blockDim.x + threadIdx.x;
}
```

1D grid of 2D blocks

```
__device__
int getGlobalIdx_1D_2D(){
      return blockIdx.x * blockDim.x * blockDim.y
             + threadIdx.y * blockDim.x + threadIdx.x;
}
```

1D grid of 3D blocks

```
__device__
int getGlobalIdx_1D_3D(){
      return blockIdx.x * blockDim.x * blockDim.y * blockDim.z
            + threadIdx.z * blockDim.y * blockDim.x
            + threadIdx.y * blockDim.x + threadIdx.x;
}
```

2D grid of 1D blocks

```
__device__ int getGlobalIdx_2D_1D(){
      int blockId   = blockIdx.y * gridDim.x + blockIdx.x;
      int threadId = blockId * blockDim.x + threadIdx.x;
      return threadId;
}
```

2D grid of 2D blocks

```
 __device__
int getGlobalIdx_2D_2D(){
      int blockId = blockIdx.x + blockIdx.y * gridDim.x;
      int threadId = blockId * (blockDim.x * blockDim.y)
                     + (threadIdx.y * blockDim.x) + threadIdx.x;
      return threadId;
}
```

2D grid of 3D blocks

```
__device__
int getGlobalIdx_2D_3D(){
      int blockId = blockIdx.x + blockIdx.y * gridDim.x;
      int threadId = blockId * (blockDim.x * blockDim.y * blockDim.z)
                     + (threadIdx.z * (blockDim.x * blockDim.y))
                     + (threadIdx.y * blockDim.x) + threadIdx.x;
      return threadId;
}
```

3D grid of 1D blocks

```
__device__
int getGlobalIdx_3D_1D(){
      int blockId = blockIdx.x + blockIdx.y * gridDim.x
                    + gridDim.x * gridDim.y * blockIdx.z;
      int threadId = blockId * blockDim.x + threadIdx.x;
      return threadId;
}
```

3D grid of 2D blocks

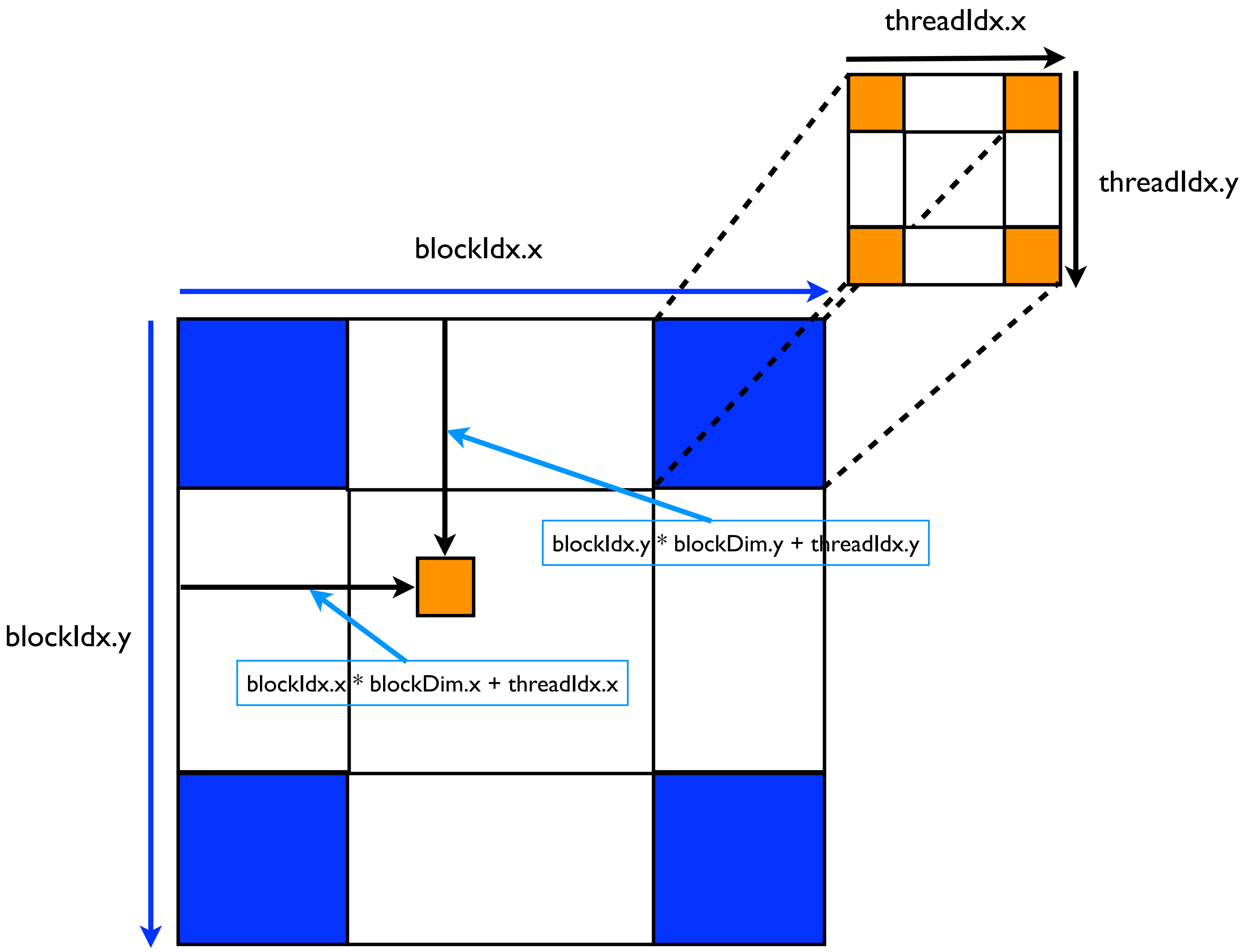
```
__device__
int getGlobalIdx_3D_2D(){
      int blockId = blockIdx.x + blockIdx.y * gridDim.x
                    + gridDim.x * gridDim.y * blockIdx.z;
      int threadId = blockId * (blockDim.x * blockDim.y)
                     + (threadIdx.y * blockDim.x) + threadIdx.x;
      return threadId;
}
```

3D grid of 3D blocks

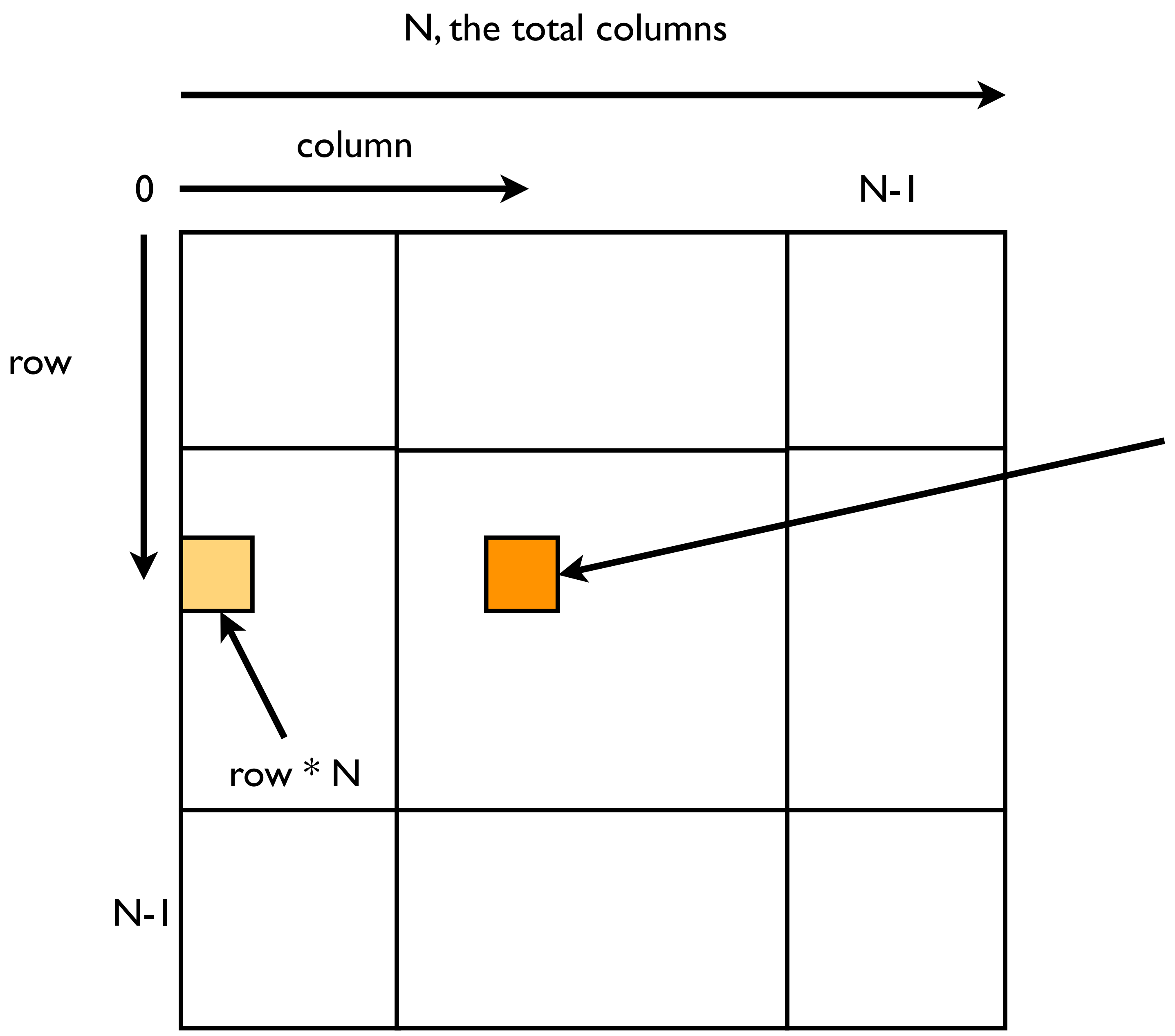
```
__device__
int getGlobalIdx_3D_3D(){
      int blockId = blockIdx.x + blockIdx.y * gridDim.x
                     + gridDim.x * gridDim.y * blockIdx.z;
      int threadId = blockId * (blockDim.x * blockDim.y * blockDim.z)
                     + (threadIdx.z * (blockDim.x * blockDim.y))
                     + (threadIdx.y * blockDim.x) + threadIdx.x;
      return threadId;
}
```

http://www.martinpeniak.com/index.php?option=com_content&view=article&catid=17:updates&id=288:cuda-thread-indexing-explained

2D Grids and 2D Blocks


threadIdx.x
threadIdx.y
blockIdx.x
blockIdx.y
blockIdx.y * blockDim.y + threadIdx.y
blockIdx.x * blockDim.x + threadIdx.x

Accessing Matrices in Linear Memory



a[row][column] = a[offset]

offset = column + row * N

In CUDA:

```
int col = blockIdx.x*blockDim.x+threadIdx.x;
int row = blockIdx.y*blockDim.y+threadIdx.y;
int index = col + row * N;
A[index] = …
```