GPU Series – VI

CUDA THREAD ORGANIZATION
BLOCKS AND GRIDS

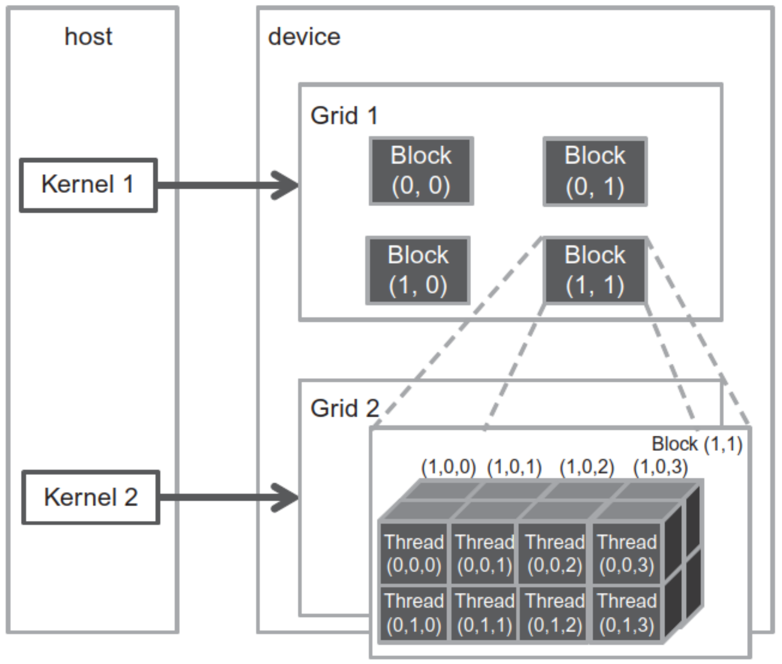

FIGURE 3.1

A multidimensional example of CUDA grid organization.

Img Src: Programming Massively Parallel Processors by David B.Kirk, Wen-mei W.Hwu

Let us understand a bit more about Blocks, Grids and Threads.

CUDA organizes threads in a hierarchical structure to execute parallel tasks efficiently.

CUDA threads are organized into a two-level hierarchy:

1. **Grid**: A grid is a collection of thread blocks.
2. **Block**: Each block is a collection of threads.

This structure allows for scalable parallel execution.

<u>All CUDA Threads in a Grid Execute the Same Kernel Function</u>

In CUDA, a kernel function is a function that runs on the GPU. When you launch a kernel, you start a large number of threads that all execute this same kernel function in parallel.

By having all threads in a grid execute the same kernel, you can perform the same operation on multiple data points simultaneously. This is particularly useful for data-parallel tasks where each thread processes a separate element of an array or matrix.

This model simplifies the design of parallel programs. You write the kernel function once, and it gets executed by many threads, ensuring consistency in operation across all data points.

```
__global__ void vectorAddKernel(float* A, float* B, float* C, int N) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) {
        C[idx] = A[idx] + B[idx];
    }
}
```

In this kernel, every thread computes one element of the resulting vector C. When launched, all threads execute this kernel function, but each thread operates on different data elements.

Threads Are Grouped Into Blocks, and Blocks Are Grouped Into Grids

CUDA organizes threads into a two-level hierarchy: **blocks** and **grids**.

Thread Blocks

A block is a group of threads that can cooperate with each other by:

- Sharing data through shared memory.
- Synchronizing their execution to coordinate access to shared resources.

Grids

A grid is a collection of blocks. The grid allows the CUDA program to scale across multiple blocks, ensuring that more threads can be launched to handle large datasets.

Why Is This Hierarchy Important?

1. This hierarchical organization allows CUDA to scale across different GPU architectures and sizes. Whether you have a small or large dataset, you can adjust the grid and block sizes accordingly.
2. Threads within a block can share data and synchronize their actions, making it easier to implement complex parallel algorithms that require data sharing.

Example

In a matrix multiplication example, you might have:

- **Blocks**: Each block handles a submatrix of the output matrix.
- **Threads**: Each thread within a block computes one element of the submatrix.

All Threads in a Block Share the Same Block Index (blockIdx)

Each block within a grid is uniquely identified by a block index. This block index is used to distinguish between different blocks within the same grid.

Block Index (blockIdx)

- blockIdx.x: The block index in the x-dimension.
- blockIdx.y: The block index in the y-dimension.
- blockIdx.z: The block index in the z-dimension.

Why Is This Important?

1. The block index helps in uniquely identifying each block within a grid, allowing you to assign different parts of the data to different blocks.
2. By using blockIdx, you can partition your data across multiple blocks, enabling parallel processing of different data segments.

Each Thread Within a Block Has a Unique Thread Index (threadIdx)

Within each block, threads are uniquely identified by their thread index. This index is used to distinguish between different threads within the same block.

Thread Index (threadIdx)

- threadIdx.x: The thread index in the x-dimension.
- threadIdx.y: The thread index in the y-dimension.
- threadIdx.z: The thread index in the z-dimension.

Why Is This Important?

1. The thread index allows you to uniquely identify each thread within a block, ensuring that each thread works on a different part of the data.
2. By assigning a unique index to each thread, CUDA ensures that each thread can execute a different part of the kernel function in parallel.

Execution Configuration

The execution configuration parameters in a kernel launch statement specify the dimensions of the grid and each block. These dimensions are represented by **gridDim** and **blockDim** in the kernel functions.

- **Grid**: A grid can be a 3-dimensional array of blocks.
- **Block**: A block can be a 3-dimensional array of threads.

When launching a kernel, you must specify the size of the grid and the blocks in each dimension using the <<< >>> syntax. The first parameter specifies the grid dimensions (number of blocks), and the second parameter specifies the block dimensions (number of threads).

```
dim3 dimGrid(32, 1, 1);     // 32 blocks in x-dimension
dim3 dimBlock(128, 1, 1);   // 128 threads per block in x-dimension
kernelFunction<<<dimGrid, dimBlock>>>(...);
```

This configuration generates a 1D grid consisting of 32 blocks, each containing 128 threads. The total number of threads in the grid is 128 * 32 = 4096.

```
dim3 dimGrid(16, 16, 1);    // 16x16 blocks in x and y dimensions
dim3 dimBlock(16, 16, 1);   // 16x16 threads per block in x and y dimensions
kernelFunction<<<dimGrid, dimBlock>>>(...);
```

This creates a 2D grid with 256 blocks (16x16) and each block containing 256 threads (16x16). The total number of threads is 256 * 256 = 65536.

Thread and Block Indexing

In CUDA, Threads within a block and blocks within a grid are indexed using built-in variables. These indices are used to determine the work each thread performs.

Thread Index (threadIdx)

- **threadIdx.x**: Index of the thread within the block in the x-dimension.
- **threadIdx.y**: Index of the thread within the block in the y-dimension.
- **threadIdx.z**: Index of the thread within the block in the z-dimension.

Block Index (blockIdx)

- **blockIdx.x**: Index of the block within the grid in the x-dimension.
- **blockIdx.y**: Index of the block within the grid in the y-dimension.
- **blockIdx.z**: Index of the block within the grid in the z-dimension.

Grid and Block Dimensions

- **blockDim.x**: Number of threads per block in the x-dimension.
- **blockDim.y**: Number of threads per block in the y-dimension.
- **blockDim.z**: Number of threads per block in the z-dimension.
- **gridDim.x**: Number of blocks per grid in the x-dimension.
- **gridDim.y**: Number of blocks per grid in the y-dimension.
- **gridDim.z**: Number of blocks per grid in the z-dimension.

Pre-Initialization

Within the kernel function, the x, y, and z fields of gridDim and blockDim are pre-initialized according to the execution configuration parameters. The allowed values for gridDim.x, gridDim.y, and gridDim.z range from *1 to 65536*. Block values for blockIdx.x range from *0 to gridDim.x-1*.

Example using Matrix Multiplication:

```
__global__ void matrixMulKernel(float* A, float* B, float* C, int N) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    float sum = 0.0f;

    if (row < N && col < N) {
        for (int k = 0; k < N; k++) {
            sum += A[row * N + k] * B[k * N + col];
        }
        C[row * N + col] = sum;
    }
}
```

```
int N = 1024;  // Size of the matrix
dim3 dimBlock(16, 16, 1);
dim3 dimGrid((N + dimBlock.x - 1) / dimBlock.x, (N + dimBlock.y - 1) / dimBlock.y, 1);

matrixMulKernel<<<dimGrid, dimBlock>>>(A, B, C, N);
```

In this example:

- We define a block with 16x16 threads.
- We define a grid with enough blocks to cover the entire matrix. The grid dimensions are calculated to ensure that all elements of the matrix are processed.

<u>Coming back the diagram:</u>

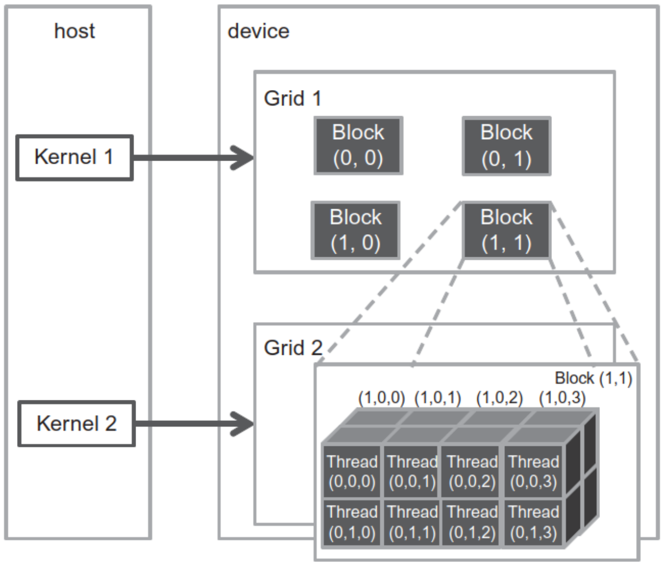

FIGURE 3.1

A multidimensional example of CUDA grid organization.

1. **Kernel 1** and **Kernel 2** represent two different kernel functions.
2. **Grid 1** contains four blocks arranged in a 2x2 configuration, where each block has its own blockIdx.
3. **Grid 2** demonstrates the organization within a block, showing threads with their unique threadIdx values.

Hierarchical Breakdown:

- **Grid 1**: Consists of 2x2 blocks. Each block is identified by its (blockIdx.x, blockIdx.y) coordinates.
- **Block (1,1)** in **Grid 2**: Contains a 2x2 array of threads. Each thread is identified by its (threadIdx.x, threadIdx.y) coordinates.

This hierarchical organization allows CUDA to handle complex data structures and computations efficiently, leveraging parallelism at multiple levels.

To Summarize this article:

- **Grid**: A grid is a collection of thread blocks.

- **Block**: Each block is a collection of threads.

- **All CUDA Threads in a Grid Execute the Same Kernel Function**:

- All threads in a grid execute the same kernel function, ensuring consistency in operations across multiple data points.

- This model simplifies parallel programming by allowing a single kernel function to be executed by many threads in parallel.

- **Threads Are Grouped Into Blocks, and Blocks Are Grouped Into Grids**:

- **Thread Blocks**: Blocks are groups of threads that can share data through shared memory and synchronize their execution.

- **Grids**: A grid is a collection of blocks, enabling the CUDA program to scale and handle large datasets efficiently.

- **Block Indexing (blockIdx)**: Each block within a grid has a unique block index (blockIdx). The block index helps in uniquely identifying and partitioning data across multiple blocks for parallel processing.

- **Thread Indexing (threadIdx)**: Within each block, threads have unique thread indices (threadIdx). These indices ensure that each thread works on different parts of the data, enabling parallel execution of the kernel function.

- Execution configuration parameters specify the dimensions of the grid and blocks in the kernel launch statement. A grid can be a 3-dimensional array of blocks, and a block can be a 3-dimensional array of threads.

- **Thread and Block Indexing Details**:

  Thread Index (threadIdx): Identifies threads within a block.

  Block Index (blockIdx): Identifies blocks within a grid.

  Grid and Block Dimensions (gridDim and blockDim): Specify the number of blocks in a grid and the number of threads in a block, respectively.

- Within the kernel function, the fields of gridDim and blockDim are pre-initialized according to execution configuration parameters.

Reference: Programming Massively Parallel Processors by David B.Kirk, Wen-mei W.Hwu