**Host code (main Function)** 🡪Run in **CPU**. Responsible of calling **CUDA Kernel** with proper device configuration

**Device code** 🡪Run in **GPU**

**Device function = kernel definition 🡪Kernel = functions in devices**

(\_\_host\_\_, \_\_device\_\_ or \_\_global\_\_) Modifiers that specify that a function is going to run in a CUDA enable device.

Additionally, every kernel **should be void** to indicate that nothing is returning from this function call.

If you want to ***return variables from kernel***, we have to explicitly transfer those memories using ***specific CUDA runtime function calls***.

cudaDeviceSynchronize(); //guarantees that host code to wait at this point until all previous kernels finish execution

cudaDeviceReset(); //just reset the device

**Grid:** Collection of the threads launch of a kernel

**Threads Block:** Threads in a grid is organized in to groups called thread blocks. They allow CUDA toolkit to synchronize and manage workload without heavy performance penalties.

Shape

Description automatically generated with low confidenceDiagram

Description automatically generatedGrid can be visualized as three dimensional. Each smaller cube of the big cube represents s one thread: Representation using thread blocks:

Launch kernel🡪 kernel\_name <<< number\_of\_blocks, thread\_per\_block >>> (arguments)

3-dim 🡪vector type (has x,y, and z values.) Initialized to 1 by default

Diagram

Description automatically generated with low confidenceOne dimensional grid with **32 threads** arranged in to **8 thread blocks** and each having **4 threads** in X dimension. Thread block = yellow boxes | threads = blue boxes

**Second Kernel Launch Parameter:** Number of threads in each block

dim3 block(4,1,1) 🡪Dimension of our blocks: 4 threads in X dim.,1 thread Y dim.1 thread and Z dim.

**First Kernel Launch Parameter:** number of blocks in each dimension

dim3 grid(4,1,1) 🡪Dimension of our blocks: 4 threads in X dim.,1 thread Y dim.1 thread and Z dim.

Usually, we refer all threads launch for a kernel as the grid. **Don’t confuse previous grid variable with what it actually represents.**

At the end, we launch the kernel in this format: kernel\_name <<<grid, block>>> (arguments)

Diagram, table

Description automatically generatedAnother example:

dim3 block(8,2,1)

dim3 grid(2,2,1)

We have in total threads = grid = 16 \* 4 = 64 theads

Int nx, ny;

nx = total number of threads in x dimension = 16

ny = total number of threads in y dimension = 4

So

dim3 grid(nx / block.x,ny / block.y,1)

Chart

Description automatically generatedDiagram

Description automatically generatedLimitations of Block Size as in 2017-2019 :

All of the following variables are initialized by CUDA Runtime | *block = thread Block*

**threadIdx**: created for each thread depending on where that particular thread is **located** in the ***thread block***.

A picture containing text

Description automatically generatedExample with a grid of 8 threads in one block all in the x-dimesion:

Calendar

Description automatically generated with medium confidenceExample with a grid of 8 threads in 2 blocks (each with 4 threads) all in x-dim:

A picture containing graphical user interface

Description automatically generatedNow, let’s look at a *2D Grid* with *2 blocks in x-dim* and *2 blocks in y-dim* and each block has 4 threads in x-dim. Even when we have a 2D Grid, ***threads in the thread block arrange in 1D***.

Diagram

Description automatically generated Now, let’s see another 2D grid. But in this case, the ***thread blocks*** are also 2D.

Shape, rectangle

Description automatically generatedExample Implementation: Launch a 2D Grid and print out threadIdx values for each thread in that grid. The grid had **256 threads** arranged in **2 thread blocks** in ***X-dim*** and ***Y-dim:*** a total of **4 thread blocks**, and each block has ***8 threads in X-dim*** and ***eight threads*** ***in Y-dim.***

Cuda runtime initializes the following variable for: | Note: *block = thread Block*

**blockIdx**: Each thread depending on the coordinates of the belonging thread blocks in a grid. The coordinate value of a thread block in a grid. This means the variable value for each thread in each block will get the coordinate value of the block. This is a dim3 type variables (x,y,z)

A picture containing text

Description automatically generatedExample of a one-dimensional grid with ***8 threads*** which are arranged into ***2 blocks in x-dim***. In Y and Z dimension there is only one block.

Example of a two-dimensional grid with ***two blocks*** in each ***x and y dim****,* and each block has ***4 threads in x-dim***.

Diagram

Description automatically generated*Imagine the grid in a cartesian coordinate system and then divide thread blocks into slots (values in each dimension)*

Diagram

Description automatically generatedExample of a two-dimensional grid with ***two blocks*** in each ***x and y dim****,* and each block have ***x and y dim***. In this case, the **blockIdx** will still have the same values as the previous example since it is bounded by the block coordinate values.

Cuda runtime initializes the following variable: | Note: *block = thread Block*

**blockDim**: **represents the dimension values of a thread block**. Notice that all blocks in a grid have the same dimension values. Thus, this variable should be the same in all threads in a grid. Also 3dim type variable.

Diagram

Description automatically generatedExample:

Cuda runtime initializes the following variable: | Note: *block = thread Block*

Shape

Description automatically generated**gridDim:** represents number of thread blocks in each dimension of a grid. Also, 3dim type variable.

Shape, rectangle, polygon

Description automatically generatedWe will use the previous example but with blockDim & gridDim