# CUDA - Threads

In this chapter, we will learn about the CUDA threads. To proceed, we need to first know how resources are assigned to blocks.

## Resource Assignment to Blocks

Execution resources are assigned to threads per block. Resources are organized into Streaming Multiprocessors (SM). Multiple blocks of threads can be assigned to a single SM. The number varies with CUDA device. For example, a CUDA device may allow up to 8 thread blocks to be assigned to an SM. This is the upper limit, and it is not necessary that for any configuration of threads, a SM will run 8 blocks.

For example, if the resources of a SM are not sufficient to run 8 blocks of threads, then the number of blocks that are assigned to it is dynamically reduced by the CUDA runtime. Reduction is done on block granularity. To reduce the amount of threads assigned to a SM, the number of threads is reduced by a block.

In recent CUDA devices, a SM can accommodate up to 1536 threads. The configuration depends upon the programmer. This can be in the form of 3 blocks of 512 threads each, 6 blocks of 256 threads each or 12 blocks of 128 threads each. The upper limit is on the number of threads, and not on the number of blocks.

Thus, the number of threads that can run parallel on a CUDA device is simply the number of SM multiplied by the maximum number of threads each SM can support. In this case, the value comes out to be SM x 1536.

## Synchronization between Threads

The CUDA API has a method, **__syncthreads()** to synchronize threads. When the method is encountered in the kernel, all threads in a block will be blocked at the calling location until each of them reaches the location.

What is the need for it? It ensure phase synchronization. That is, all the threads of a block will now start executing their next phase only after they have finished the previous one. There are certain nuances to this method. For example, if a __syncthreads statement, is present in the kernel, it must be executed by all threads of a block. If it is present inside an if statement, then either all the threads in the block go through the if statement, or none of them does.

If an if-then-else statement is present inside the kernel, then either all the threads will take the **if** path, or all the threads will take the else path. This is implied. As all the threads of a block have to execute the sync method call, if threads took different paths, then they will be blocked forever.

It is the duty of the programmer to be wary of such conditions that may arise.

## Thread Scheduling

After a block of threads is assigned to a SM, it is divided into sets of 32 threads, each called a warp. However, the size of a warp depends upon the implementation. The CUDA specification does not specify it.

Here are some important properties of warps −

- A warp is a unit of thread scheduling in SMs. That is, the granularity of thread scheduling is a warp. A block is divided into warps for scheduling purposes.
- A SM is composed of many SPs (Streaming Processors). These are the actual CUDA cores. Normally, the number of CUDA cores in a SM is less than the total number of threads that are assigned to it. Thus the need for scheduling.
- While a warp is waiting for the results of a previously executed long-latency operation (like data fetch from the RAM), a different warp that is not waiting and is ready to be assigned is selected for execution. This means that threads are always scheduled in a group.
- If more than one warps are on the ready queue, then some priority mechanism can be used for assignment. One such method is the round-robin.
- A warp consists of threads with consecutive threadIdx.x values. For example, threads of the first warp will have threads ids b/w 0 to 31. Threads of the second warp will have thread ids from 32-63.
- All threads in a warp follow the SIMD model. SIMD stands for ‘Single Instruction, Multiple Data’. It is different when compared to SPMD. In SIMD, each thread is executing the same instruction of a kernel at any given time. But the data is always different.

### Latency Tolerance

A lot of processor time goes waste if while a warp is waiting, another warp is not scheduled. This method of utilizing in the latency time of operations with work from other warps is called latency tolerance.