Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)
High Performance Computing Laboratory (DJ19DSL802)

HPC Experiment-5

Bhuvi Ghosh
60009210191

Aim: Accelerating a For Loop with Multiple Blocks of Threads

Theory:

There is a limit to the number of threads that can exist in a thread block: 1024 to be precise. In order to increase the amount of parallelism in accelerated applications, we must be able to coordinate among multiple thread blocks.

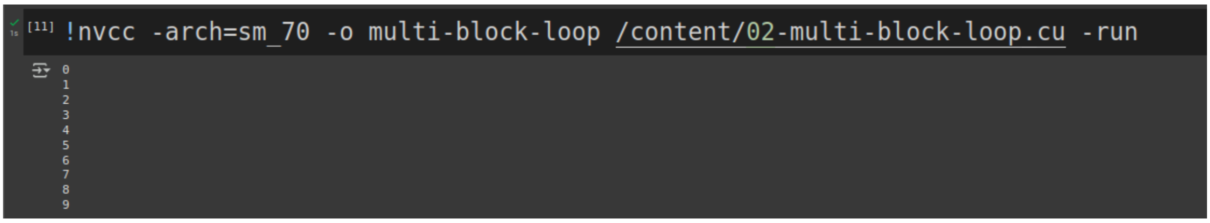
CUDA Kernels have access to a special variable that gives the number of threads in a block: blockDim.x. Using this variable, in conjunction with blockIdx.x and threadIdx.x, increased parallelization can be accomplished by organizing parallel execution across multiple blocks of multiple threads with the idiomatic expression threadIdx.x + blockIdx.x * blockDim.x. Here is a detailed example.

The execution configuration <<<10, 10>>> would launch a grid with a total of 100 threads, contained in 10 blocks of 10 threads. We would therefore hope for each thread to have the ability to calculate some index unique to itself between 0 and 99.

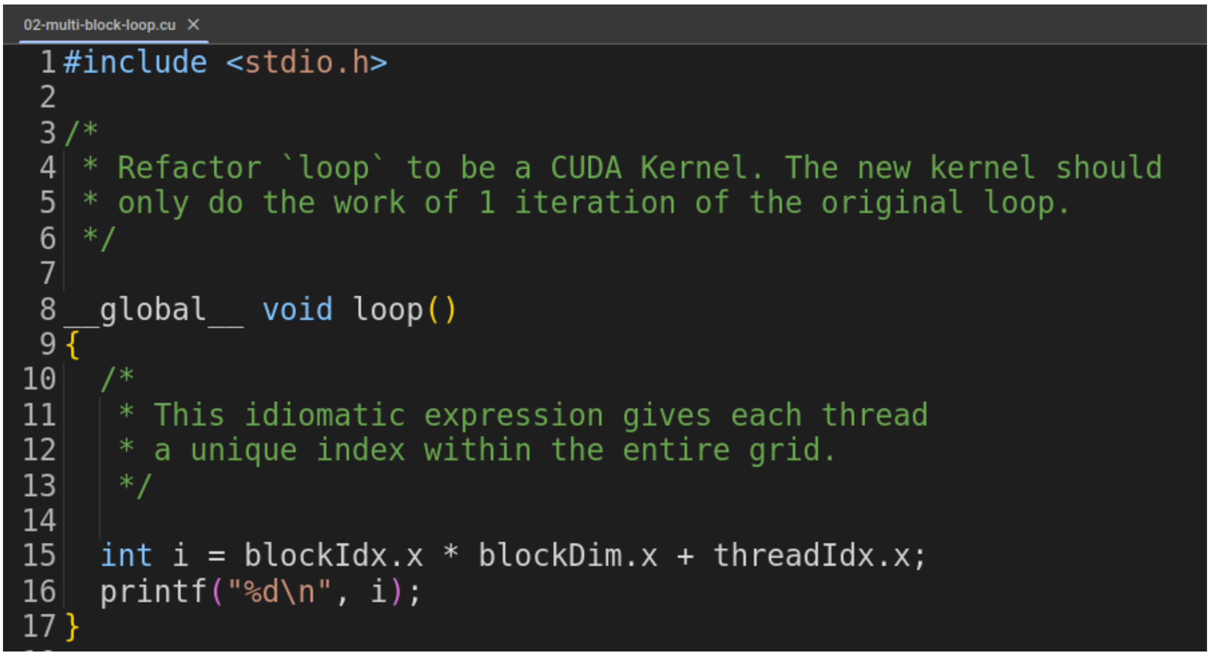
1. If block blockIdx.x equals 0, then blockIdx.x * blockDim.x is 0. Adding to 0 the possible threadIdx.x values 0 through 9, then we can generate the indices 0 through 9 within the 100 thread grid.
2. If block blockIdx.x equals 1, then blockIdx.x * blockDim.x is 10. Adding to 10 the possible threadIdx.x values 0 through 9, then we can generate the indices 10 through 19 within the 100 thread grid.
3. If block blockIdx.x equals 5, then blockIdx.x * blockDim.x is 50. Adding to 50 the possible threadIdx.x values 0 through 9, then we can generate the indices 50 through 59 within the 100 thread grid.
4. If block blockIdx.x equals 9, then blockIdx.x * blockDim.x is 90. Adding to 90 the possible threadIdx.x values 0 through 9, then we can generate the indices 90 through 99 within the 100 thread grid.

Lab Experiment to be performed:

Currently, the loop function inside 02-multi-block-loop.cu runs a for loop that will serially print the numbers 0 through 9. Refactor the loop function to be a CUDA kernel which will launch to execute N iterations in parallel. After successfully refactoring, the numbers 0 through 9 should still be printed. For this exercise, as an additional constraint, use an execution configuration that launches at least 2 blocks of threads. Refer to the solution if you get stuck.



```
[11] !nvcc -arch=sm_70 -o multi-block-loop /content/02-multi-block-loop.cu -run
0
1
2
3
4
5
6
7
8
9
```

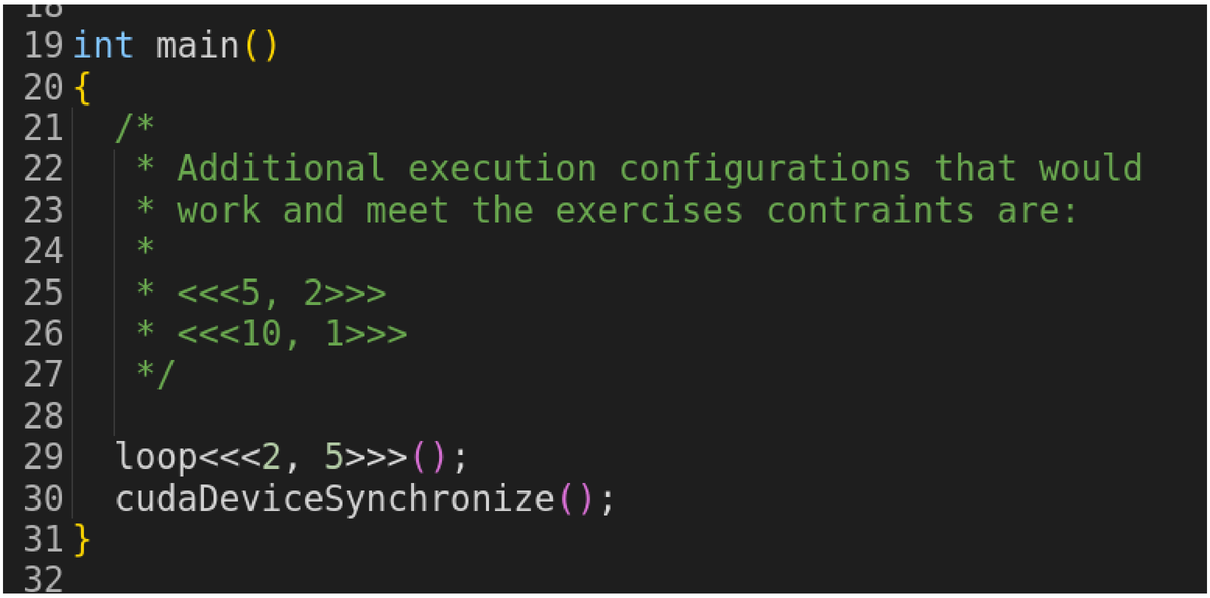


```
#include <stdio.h>

/*
 * Refactor `loop` to be a CUDA Kernel. The new kernel should
 * only do the work of 1 iteration of the original loop.
 */

__global__ void loop()
{
  /*
   * This idiomatic expression gives each thread
   * a unique index within the entire grid.
   */

  int i = blockIdx.x * blockDim.x + threadIdx.x;
  printf("%d\n", i);
}
```



```

int main()
{
  /*
   * Additional execution configurations that would
   * work and meet the exercises contraints are:
   *
   * <<<5, 2>>>
   * <<<10, 1>>>
   */

  loop<<<2, 5>>>();
  cudaDeviceSynchronize();
}

```