Bhuvi Ghosh
60009210191

HPC Experiment-2

Aim: Launch Parallel Kernels

Theory:

The execution configuration allows programmers to specify details about launching the kernel to run in parallel on multiple GPU threads. More precisely, the execution configuration allows programmers to specify how many groups of threads - called thread blocks, or just blocks - and how many threads they would like each thread block to contain. The syntax for this is:

<<< NUMBER_OF_BLOCKS, NUMBER_OF_THREADS_PER_BLOCK>>>

The kernel code is executed by every thread in every thread block configured when the kernel is launched.

Thus, under the assumption that a kernel called someKernel has been defined, the following are true:

someKernel<<<1, 1>>>() is configured to run in a single thread block which has a single thread and will therefore run only once.

someKernel<<<1, 10>>>() is configured to run in a single thread block which has 10 threads and will therefore run 10 times.

someKernel<<<10, 1>>>() is configured to run in 10 thread blocks which each have a single thread and will therefore run 10 times.

someKernel<<<10, 10>>>() is configured to run in 10 thread blocks which each have 10 threads and will therefore run 100 times.

Lab experiment to be performed:

01-first-parallel.cu currently makes a very basic function call that prints the message This should be running in parallel. Follow the steps below to refactor it first to run on the GPU, and then, in parallel, both in a single, and then, in multiple thread blocks. Refer to the solution if you get stuck.

Refactor the firstParallel function to launch as a CUDA kernel on the GPU. You should still be able to see the output of the function after compiling and running 01-first-parallel.cu with the nvcc command just below.

Refactor the firstParallel kernel to execute in parallel on 5 threads, all executing in a single thread block. You should see the output message printed 5 times after compiling and running the code.

Refactor the firstParallel kernel again, this time to execute in parallel inside 5 thread blocks, each containing 5 threads. You should see the output message printed 25 times now after compiling and running.

[4] 2s

```
%%cuda
#include <stdio.h>

__global__ void firstParallel()
{
  printf("This should be running in parallel on the GPU.\n");
}

int main()
{
  // Launch the kernel on the GPU with 1 block and 1 thread
  firstParallel<<<1, 5>>>();

  // Ensure that the CPU waits for the GPU to complete
  cudaDeviceSynchronize();

  return 0;
}
```

```
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
```

[5] 1s

```
%%cuda
#include <stdio.h>

__global__ void firstParallel()
{
  printf("This should be running in parallel on the GPU.\n");
}

int main()
{
  // Launch the kernel on the GPU with 1 block and 1 thread
  firstParallel<<<5, 5>>>();

  // Ensure that the CPU waits for the GPU to complete
  cudaDeviceSynchronize();

  return 0;
}
```

```
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
This should be running in parallel on the GPU.
```

Conclusion:

- By launching the kernel with different execution configurations, we observed how CUDA parallelism works.
- Running the kernel on a **single thread block with multiple threads** showed that each thread executes the kernel independently.
- Increasing the number of **thread blocks** further enhanced parallel execution, confirming that CUDA efficiently distributes computations across multiple threads and blocks.
- This experiment demonstrated the fundamental principles of **parallel execution** in CUDA, providing a foundation for optimizing performance in GPU computing.