Lab Exercise 1 – Writing, Compiling and Executing Simple CUDA Programs

---

Objective:

- Write and understand the structure of a simple "Hello World" CUDA program.
- Compile and run the CUDA program using the nvcc compiler.

---

1. Writing a Simple CUDA "Hello World" Program

In this exercise, we will write a basic CUDA program that launches a kernel to print "Hello, World!" from the GPU. We'll explore how the kernel is executed on the GPU and how the output is displayed.

CUDA Program: Hello World

```
#include <iostream>
#include <cuda_runtime.h>

// CUDA Kernel function to print "Hello, World!"
__global__ void helloWorld() {
   printf("Hello, World! from thread %d\n", threadIdx.x);
}

int main() {
   // Launch the kernel with one block and 10 threads
   helloWorld<<<1, 10>>>();

   // Wait for GPU to finish before continuing
   cudaDeviceSynchronize();

   return 0;
}
```

---

2. Explanation of the Code

Host Code (main function):

- **helloWorld<<<1, 10>>>();**:
  - This line launches the helloWorld kernel with **1 block** and **10 threads**.
  - The kernel will execute in parallel by 10 threads in the block.
- **cudaDeviceSynchronize();**:
  - This function ensures that the CPU waits for the GPU to complete the execution of the kernel before continuing the program.
  - Without this, the CPU might finish execution and exit before the GPU has had a chance to print its output.

Device Code (CUDA Kernel helloWorld):

- **__global__ void helloWorld()**:
  - This is the kernel function. It will be executed on the GPU by multiple threads.
  - The __global__ qualifier specifies that this is a device function that is called from the host (CPU) and runs on the GPU.
- **printf("Hello, World! from thread %d\n", threadIdx.x);**:
  - The printf function is used to print output from the GPU. Each thread will print "Hello, World!" and its thread index (threadIdx.x), which uniquely identifies the thread within its block.
  - threadIdx.x provides the thread's index in the x-dimension of the block. This helps differentiate the outputs from different threads.

---

3. Compiling the Program Using NVCC

To compile the CUDA program, use the NVIDIA CUDA Compiler (nvcc).

Step 1: Compile the Program

Open the terminal, navigate to the directory where your CUDA source file (e.g., hello_world.cu) is located, and run the following command:

```
nvcc -o hello_world hello_world.cu
```

- nvcc: The NVIDIA CUDA Compiler.
- -o hello_world: Specifies the output executable file name (hello_world).
- hello_world.cu: The source code file.

Step 2: Run the Program

After the program has been successfully compiled, run the executable:

```
hello_world.exe
```

---

4. Expected Output

When you run the program, you should see output similar to this, with each thread printing its own unique message:

```
Hello, World! from thread 0
Hello, World! from thread 1
Hello, World! from thread 2
Hello, World! from thread 3
Hello, World! from thread 4
Hello, World! from thread 5
Hello, World! from thread 6
Hello, World! from thread 7
Hello, World! from thread 8
```

```
Hello, World! from thread 9
```

Note that the order of the output may vary, as the threads run in parallel and can print in any order.

---

5. Key Concepts

- **__global__ keyword**: This indicates that the function is a CUDA kernel. It is executed on the GPU by multiple threads.
- **printf in CUDA**: You can use printf to print messages from within a CUDA kernel. Each thread can execute this independently, and the output will come from each thread.
- **Thread Indexing (threadIdx.x)**: Each thread has a unique index, and threadIdx.x is used to identify the thread within a block.
- **Launching Kernels (<<<1, 10>>>)**: The kernel is launched with **1 block** and **10 threads** per block. Each thread will execute the printf statement.

---

6. Common Errors and Troubleshooting

- **Error: "Device function called without sufficient resources"**
    - This might occur if the number of threads or blocks exceeds the available resources (e.g., memory) on your GPU.
    - Ensure that the kernel launch configuration is reasonable and fits within your device's limits.
- **Error: "CUDA runtime error: invalid device function"**
    - This error can occur if your program is compiled for a GPU architecture that is not supported by your current device.

- Ensure you are compiling the code for the correct GPU architecture by specifying the correct -arch flag during compilation (e.g., nvcc -arch=sm_60 for a device with a compute capability of 6.0).

---

7. Summary

- **Writing Simple CUDA Programs**: We wrote a basic CUDA program to print "Hello World!" from the GPU using multiple threads.
- **Compiling with nvcc**: The program was compiled using the nvcc compiler, and the resulting executable was run to see the output from the GPU.
- **Parallel Execution on the GPU**: The program demonstrated the parallel execution of threads in CUDA and how they print their individual results.