Bhuvi Ghosh
60009210191

HPC Experiment-1

Aim:

To write and execute a simple CUDA program that prints "Hello from the CPU" from the CPU and "Hello from the GPU" from the GPU.

Theory:

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA for general computing on NVIDIA GPUs (Graphics Processing Units). By using CUDA, developers can write programs that utilize the power of the GPU for computations that can be done in parallel.

A **GPU kernel** is a function written for execution on the GPU, using the `__global__` qualifier. When calling this function from the CPU, the kernel executes in parallel on multiple threads distributed over the GPU. A **CPU function** is a regular function executed on the host (CPU).

In CUDA programming:

- The **CPU** (host) executes the regular program flow and may launch GPU kernels.
- The **GPU** (device) executes the parallel computations. These computations are expressed through **kernels**, which are executed by a set of threads organized in **blocks** and **grids**.

CUDA Key Concepts

- **GPU Kernel**: A function that is executed on the GPU, which is designed to be executed in parallel by multiple threads. In this case, the kernel prints a message, but in real applications, it can perform parallel computations.
- **CPU (Host)**: The traditional processor, where the main program is executed sequentially. The CPU can call GPU kernels, but it handles other general tasks.
- **GPU (Device)**: The graphics processor, where parallel computations are executed. The GPU runs many threads in parallel to process large volumes of data efficiently.

CUDA Execution Model

CUDA uses a model where a kernel (a special function marked by `__global__`) is executed across many threads. These threads are grouped into **blocks**, and blocks are grouped into **grids**. Threads execute independently but can synchronize and communicate through specific mechanisms.

The general flow of the program:

1. The CPU function runs on the host and performs serial computations or control.
2. The kernel runs on the GPU in parallel by multiple threads.
3. CUDA kernels are launched with a grid structure defined by blocks and threads.

4. Synchronization is often necessary to ensure the GPU computation completes before the program continues or exits.

In this experiment, the program demonstrates basic communication between the CPU and GPU by printing messages using both the CPU and GPU.

Explanation of the Program:

1. **helloCPU function**: This is a simple CPU function that prints "Hello from the CPU" when called.
2. **helloGPU kernel**: This function is a CUDA kernel that runs on the GPU. It uses the `__global__` qualifier to indicate that it's a kernel function. The message "Hello from the GPU" is printed when the kernel is executed on the GPU.
3. **Main function**:
    - The `helloCPU()` function is called directly from the CPU to print the message.
    - The `helloGPU<<<1, 1>>>()` line launches the `helloGPU` kernel on the GPU. The `<<<1, 1>>>` specifies that the kernel is launched with a grid of 1 block and 1 thread.
    - `cudaDeviceSynchronize()` is called to ensure that the GPU finishes its work before the program exits. This is necessary because kernel execution is asynchronous, and we want to ensure that the "Hello from the GPU" message is printed before the program finishes.

[1] `!nvcc --version`

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
```

```
!python --version
!nvcc --version
!pip install nvcc4jupyter
%load_ext nvcc4jupyter
```

```
Python 3.10.12
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
Collecting nvcc4jupyter
  Downloading nvcc4jupyter-1.2.1-py3-none-any.whl.metadata (5.1 kB)
Downloading nvcc4jupyter-1.2.1-py3-none-any.whl (10 kB)
Installing collected packages: nvcc4jupyter
Successfully installed nvcc4jupyter-1.2.1
Detected platform "Colab". Running its setup...
Source files will be saved in "/tmp/tmp_eyiw199".
```

[3]
```
%%cuda
#include <iostream>
int
main()
{
    std::cout <<"Welcome to CUDA";
    return 0;
}
```

```
Welcome to CUDA
```

[ ]
```
%%cuda
#include <iostream>
int main()
{
    printf("Welcome to CUDA");
    return 0;
}
```

```
Welcome to CUDA
```

[ ]
```
%%cuda
#include <stdio.h>
void helloCPU()
{
    printf("Hello from the CPU\n");
}
__global__ void helloGPU()
{
    printf("Hello from the GPU\n");
}
int main()
{
    helloCPU();
    helloGPU<<<1,1>>>();
    cudaDeviceSynchronize();
    return(0);
}
```

```
Hello from the CPU
Hello from the GPU
```

Conclusion:

This experiment demonstrates the basic structure of a CUDA program, showing how to define and call functions on both the CPU and GPU. We can see that the CPU function executes directly, while the GPU function requires a kernel launch and synchronization to ensure proper execution. This example can be expanded to perform more complex tasks, such as matrix calculations or simulations, on the GPU for faster parallel execution.