**Lab Exercise 3.1 – Basics of Threads and Threading Models**

**Objective:**

* Understand the basics of threads in CUDA programming.
* Learn about the different threading models and how to manage threads in CUDA.
* Implement a simple program to illustrate threading concepts in CUDA.

**1. Key Concepts**

| **Concept** | **Description** |
| --- | --- |
| Threads | Basic units of execution in CUDA. Each thread runs a portion of the kernel. |
| Threading Models | Describes how threads are organized and executed, including SIMT (Single Instruction, Multiple Threads). |
| Kernel | A function executed by multiple threads, where each thread executes in parallel. |
| Thread Hierarchy | Threads are grouped into blocks, and blocks are organized into grids. |
| threadIdx | The index of the thread within a block. |
| blockIdx | The index of the block within the grid. |
| blockDim | The dimensions of a block (number of threads per block). |
| gridDim | The dimensions of the grid (number of blocks). |

**2. CUDA Program: Basic Threading Model**

This program demonstrates the basics of threading and the use of threadIdx, blockIdx, blockDim, and gridDim.

#include <iostream>

#include <cuda\_runtime.h>

\_\_global\_\_ void basicThreadingModel() {

int tid = threadIdx.x + blockIdx.x \* blockDim.x;

int bid = blockIdx.x;

int bdim = blockDim.x;

printf("Thread ID: %d, Block ID: %d, Threads per block: %d\n", tid, bid, bdim);

}

int main() {

int N = 8;

int threadsPerBlock = 4;

int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;

basicThreadingModel<<<blocksPerGrid, threadsPerBlock>>>();

cudaDeviceSynchronize(); // Ensure all threads complete before exiting

return 0;

}

**3. Explanation**

* **Kernel Function** basicThreadingModel: This is a simple kernel that prints the thread ID (threadIdx.x), block ID (blockIdx.x), and the number of threads per block (blockDim.x) for each thread in the grid.
* **Global Indexing**: The variable tid is the global thread ID, calculated using the formula threadIdx.x + blockIdx.x \* blockDim.x, which uniquely identifies each thread in the entire grid.
* **Block and Thread Information**:
  + threadIdx.x: The index of the thread within the current block.
  + blockIdx.x: The index of the block in the grid.
  + blockDim.x: The number of threads in each block.
  + gridDim.x: The number of blocks in the grid (not used in this example but can be referenced similarly to blockDim.x and threadIdx.x).

**4. Compile and Run**

nvcc -o threading\_model threading\_model.cu

./threading\_model

Expected output:

Thread ID: 0, Block ID: 0, Threads per block: 4

Thread ID: 1, Block ID: 0, Threads per block: 4

Thread ID: 2, Block ID: 0, Threads per block: 4

Thread ID: 3, Block ID: 0, Threads per block: 4

Thread ID: 4, Block ID: 1, Threads per block: 4

Thread ID: 5, Block ID: 1, Threads per block: 4

Thread ID: 6, Block ID: 1, Threads per block: 4

Thread ID: 7, Block ID: 1, Threads per block: 4

**5. Summary**

* **Thread Hierarchy**: In CUDA, threads are organized into **blocks**, and blocks are organized into a **grid**.
  + Threads are the basic units of execution.
  + Blocks provide a way to organize threads into groups that can cooperate with shared memory.
  + Grids are a collection of blocks that are executed in parallel.
* **Global Thread ID**: Each thread can access its global ID using threadIdx, blockIdx, and blockDim.
* **Threading Models**: CUDA uses the **SIMT** (Single Instruction, Multiple Thread) model, where multiple threads execute the same instruction but operate on different data.