**Lab Exercise 2.1 – Core Communication and Synchronization in CUDA (Shared and Global Memory Access)**

**Objective:**

* Understand how threads (cores) in a CUDA block communicate and synchronize.
* Learn to use **shared memory** and \_\_syncthreads() for coordination.
* Implement a program that demonstrates intra-block thread collaboration.

**1. Key Concepts**

| **Concept** | **Description** |
| --- | --- |
| Shared Memory | Fast, low-latency memory accessible by all threads in a block. |
| \_\_syncthreads() | Synchronizes all threads in a block to ensure data consistency. |
| Thread Cooperation | Threads can share data via shared memory and coordinate execution steps. |
| Data Race | Occurs when multiple threads access the same memory without synchronization. |

compares vector addition using **global memory** and **shared memory**, and reports the **execution time** of both using **CUDA events**.

**File: vector\_add\_compare.cu**

#include <iostream>

#include <cuda.h>

#define N 1024 \* 1024 // Large enough to see time difference

#define BLOCK\_SIZE 256

// Global memory version

\_\_global\_\_ void vectorAddGlobal(float \*a, float \*b, float \*c) {

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

if (tid < N) {

c[tid] = a[tid] + b[tid];

}

}

// Shared memory version

\_\_global\_\_ void vectorAddShared(float \*a, float \*b, float \*c) {

\_\_shared\_\_ float s\_a[BLOCK\_SIZE];

\_\_shared\_\_ float s\_b[BLOCK\_SIZE];

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

if (tid < N) {

s\_a[threadIdx.x] = a[tid];

s\_b[threadIdx.x] = b[tid];

\_\_syncthreads();

c[tid] = s\_a[threadIdx.x] + s\_b[threadIdx.x];

}

}

int main() {

float \*a, \*b, \*c\_global, \*c\_shared;

float \*d\_a, \*d\_b, \*d\_c\_global, \*d\_c\_shared;

size\_t size = N \* sizeof(float);

// Allocate host memory

a = (float\*)malloc(size);

b = (float\*)malloc(size);

c\_global = (float\*)malloc(size);

c\_shared = (float\*)malloc(size);

// Initialize host vectors

for (int i = 0; i < N; i++) {

a[i] = 1.0f \* i;

b[i] = 2.0f \* i;

}

// Allocate device memory

cudaMalloc(&d\_a, size);

cudaMalloc(&d\_b, size);

cudaMalloc(&d\_c\_global, size);

cudaMalloc(&d\_c\_shared, size);

// Copy data to device

cudaMemcpy(d\_a, a, size, cudaMemcpyHostToDevice);

cudaMemcpy(d\_b, b, size, cudaMemcpyHostToDevice);

// Set up CUDA events

cudaEvent\_t start, stop;

float timeGlobal = 0.0f, timeShared = 0.0f;

int numBlocks = (N + BLOCK\_SIZE - 1) / BLOCK\_SIZE;

// Time the global memory kernel

cudaEventCreate(&start);

cudaEventCreate(&stop);

cudaEventRecord(start);

vectorAddGlobal<<<numBlocks, BLOCK\_SIZE>>>(d\_a, d\_b, d\_c\_global);

cudaEventRecord(stop);

cudaEventSynchronize(stop);

cudaEventElapsedTime(&timeGlobal, start, stop);

// Time the shared memory kernel

cudaEventRecord(start);

vectorAddShared<<<numBlocks, BLOCK\_SIZE>>>(d\_a, d\_b, d\_c\_shared);

cudaEventRecord(stop);

cudaEventSynchronize(stop);

cudaEventElapsedTime(&timeShared, start, stop);

// Copy results back to host

cudaMemcpy(c\_global, d\_c\_global, size, cudaMemcpyDeviceToHost);

cudaMemcpy(c\_shared, d\_c\_shared, size, cudaMemcpyDeviceToHost);

// Verify correctness (first 5 elements)

std::cout << "Sample results:\n";

for (int i = 0; i < 5; i++) {

std::cout << "c\_global[" << i << "] = " << c\_global[i]

<< ", c\_shared[" << i << "] = " << c\_shared[i] << std::endl;

}

// Print execution times

std::cout << "\nExecution Time (Global Memory): " << timeGlobal << " ms" << std::endl;

std::cout << "Execution Time (Shared Memory): " << timeShared << " ms" << std::endl;

// Cleanup

free(a); free(b); free(c\_global); free(c\_shared);

cudaFree(d\_a); cudaFree(d\_b); cudaFree(d\_c\_global); cudaFree(d\_c\_shared);

cudaEventDestroy(start);

cudaEventDestroy(stop);

return 0;

}

**How to Compile and Run**

nvcc -o vector\_add\_compare vector\_add\_compare.cu

./vector\_add\_compare

**Output (Example)**

Sample results:

c\_global[0] = 0, c\_shared[0] = 0

c\_global[1] = 3, c\_shared[1] = 3

c\_global[2] = 6, c\_shared[2] = 6

c\_global[3] = 9, c\_shared[3] = 9

c\_global[4] = 12, c\_shared[4] = 12

Execution Time (Global Memory): 211.795 ms

Execution Time (Shared Memory): 0.182272 ms