**Lab Exercise 11.3 – Optimizing Memory using Coalesced and Non-Coalesced Memory Accesses in CUDA**

**Objective:**

To measure and understand the performance differences between coalesced and non-coalesced global memory access patterns in CUDA kernels.

**Background:**

* In CUDA, **coalesced memory access** means that threads in a warp access consecutive memory addresses.
* **Non-coalesced memory access** occurs when threads in a warp access memory locations that are not adjacent.
* Coalescing reduces memory transactions, improving performance.

**Exercise Code: coalescing.cu**

#include <iostream>

#include <cuda\_runtime.h>

#define N 1024

#define BLOCK\_SIZE 32

\_\_global\_\_ void nonCoalescedKernel(float \*input, float \*output, int width) {

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

int tx = threadIdx.x;

int ty = threadIdx.y;

if (tid < width \* width) {

int row = tid % width;

int col = tid / width;

output[tid] = input[col + row \* width]; // non-coalesced

}

}

\_\_global\_\_ void coalescedKernel(float \*input, float \*output, int width) {

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

if (tid < width \* width) {

output[tid] = input[tid]; // coalesced

}

}

int main() {

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

float \*h\_input = new float[N \* N];

float \*h\_output = new float[N \* N];

// Initialize input

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

h\_input[i] = static\_cast<float>(i);

}

float \*d\_input, \*d\_output;

cudaMalloc(&d\_input, size);

cudaMalloc(&d\_output, size);

cudaMemcpy(d\_input, h\_input, size, cudaMemcpyHostToDevice);

dim3 block(BLOCK\_SIZE);

dim3 grid((N \* N + BLOCK\_SIZE - 1) / BLOCK\_SIZE);

// Non-coalesced access

cudaEvent\_t start1, stop1;

cudaEventCreate(&start1);

cudaEventCreate(&stop1);

cudaEventRecord(start1);

nonCoalescedKernel<<<grid, block>>>(d\_input, d\_output, N);

cudaEventRecord(stop1);

cudaEventSynchronize(stop1);

float time1 = 0;

cudaEventElapsedTime(&time1, start1, stop1);

// Coalesced access

cudaEvent\_t start2, stop2;

cudaEventCreate(&start2);

cudaEventCreate(&stop2);

cudaEventRecord(start2);

coalescedKernel<<<grid, block>>>(d\_input, d\_output, N);

cudaEventRecord(stop2);

cudaEventSynchronize(stop2);

float time2 = 0;

cudaEventElapsedTime(&time2, start2, stop2);

std::cout << "Non-Coalesced Kernel Time: " << time1 << " ms\n";

std::cout << "Coalesced Kernel Time: " << time2 << " ms\n";

cudaFree(d\_input);

cudaFree(d\_output);

delete[] h\_input;

delete[] h\_output;

return 0;

}

**Compilation Command (Windows CMD or PowerShell):**

Make sure you're in the same folder as your .cu file and run:

nvcc -o coalescing coalescing.cu

./coalescing

**Expected Output (Example):**

Non-Coalesced Kernel Time: 0.354 ms

Coalesced Kernel Time: 0.226 ms

**Takeaway:**

* Coalesced memory access is significantly faster.
* Efficient memory access patterns are crucial in optimizing CUDA kernels.