**Lab Exercise 9.2 – CPU vs GPU Matrix Operation**

Here's a complete **CUDA C++ program (.cu)** that multiplies two large matrices on both the **CPU and the GPU**, measures the time taken for each, and prints the difference.

**Filename: matrix\_mul\_compare.cu**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <cuda\_runtime.h>

#define N 1024 // Matrix size: N x N

// CUDA kernel for matrix multiplication

\_\_global\_\_ void matrixMulCUDA(float \*A, float \*B, float \*C, int n) {

int row = blockIdx.y \* blockDim.y + threadIdx.y;

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

float sum = 0;

if (row < n && col < n) {

for (int k = 0; k < n; ++k) {

sum += A[row \* n + k] \* B[k \* n + col];

}

C[row \* n + col] = sum;

}

}

// CPU function for matrix multiplication

void matrixMulCPU(float \*A, float \*B, float \*C, int n) {

for (int row = 0; row < n; ++row) {

for (int col = 0; col < n; ++col) {

float sum = 0;

for (int k = 0; k < n; ++k) {

sum += A[row \* n + k] \* B[k \* n + col];

}

C[row \* n + col] = sum;

}

}

}

int main() {

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

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

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

float \*C\_cpu = (float \*)malloc(size);

float \*C\_gpu = (float \*)malloc(size);

// Initialize input matrices with random values

srand(time(NULL));

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

A[i] = static\_cast<float>(rand()) / RAND\_MAX;

B[i] = static\_cast<float>(rand()) / RAND\_MAX;

}

// ---------- CPU TIME ----------

clock\_t start\_cpu = clock();

matrixMulCPU(A, B, C\_cpu, N);

clock\_t end\_cpu = clock();

double cpu\_time = (double)(end\_cpu - start\_cpu) / CLOCKS\_PER\_SEC;

std::cout << "CPU time: " << cpu\_time << " seconds\n";

// ---------- GPU MEMORY ALLOCATION ----------

float \*d\_A, \*d\_B, \*d\_C;

cudaMalloc(&d\_A, size);

cudaMalloc(&d\_B, size);

cudaMalloc(&d\_C, size);

cudaMemcpy(d\_A, A, size, cudaMemcpyHostToDevice);

cudaMemcpy(d\_B, B, size, cudaMemcpyHostToDevice);

dim3 threadsPerBlock(16, 16);

dim3 blocksPerGrid((N + 15) / 16, (N + 15) / 16);

// ---------- GPU TIME ----------

cudaEvent\_t start\_gpu, stop\_gpu;

cudaEventCreate(&start\_gpu);

cudaEventCreate(&stop\_gpu);

cudaEventRecord(start\_gpu);

matrixMulCUDA<<<blocksPerGrid, threadsPerBlock>>>(d\_A, d\_B, d\_C, N);

cudaDeviceSynchronize();

cudaEventRecord(stop\_gpu);

cudaEventSynchronize(stop\_gpu);

float gpu\_time = 0;

cudaEventElapsedTime(&gpu\_time, start\_gpu, stop\_gpu);

std::cout << "GPU time: " << gpu\_time / 1000.0f << " seconds\n";

cudaMemcpy(C\_gpu, d\_C, size, cudaMemcpyDeviceToHost);

// ---------- CLEANUP ----------

cudaFree(d\_A);

cudaFree(d\_B);

cudaFree(d\_C);

free(A);

free(B);

free(C\_cpu);

free(C\_gpu);

return 0;

}

**To Compile and Run:**

nvcc matrix\_mul\_compare.cu -o matrix\_mul\_compare

./matrix\_mul\_compare

**Expected Output (Varies with hardware):**

CPU time: 3.45 seconds

GPU time: 0.12 seconds