//4.1 MartrixMul

#include <cmath>

#include <cstdlib>

#include <iostream>

#define checkCudaErrors(call) \

do { \

cudaError\_t err = call; \

if (err != cudaSuccess) { \

printf("CUDA error at %s %d: %s\n", \_\_FILE\_\_, \_\_LINE\_\_, cudaGetErrorString(err)); \

exit(EXIT\_FAILURE); \

} \

} while (0)

using namespace std;

// Matrix multiplication Cuda

\_\_global\_\_ void matrixMultiplication(int \*a, int \*b, int \*c, int n) {

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

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

int sum = 0;

if (row < n && col < n)

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

sum = sum + a[row \* n + j] \* b[j \* n + col];

}

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

}

int main() {

int \*a, \*b, \*c;

int \*a\_dev, \*b\_dev, \*c\_dev;

int n = 10;

a = new int[n \* n];

b = new int[n \* n];

c = new int[n \* n];

int \*d = new int[n \* n];

int size = n \* n \* sizeof(int);

checkCudaErrors(cudaMalloc(&a\_dev, size));

checkCudaErrors(cudaMalloc(&b\_dev, size));

checkCudaErrors(cudaMalloc(&c\_dev, size));

// Array initialization

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

a[i] = rand() % 10;

b[i] = rand() % 10;

}

cout << "Given matrix A is =>\n";

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

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

cout << a[row \* n + col] << " ";

}

cout << "\n";

}

cout << "\n";

cout << "Given matrix B is =>\n";

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

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

cout << b[row \* n + col] << " ";

}

cout << "\n";

}

cout << "\n";

cudaEvent\_t start, end;

checkCudaErrors(cudaEventCreate(&start));

checkCudaErrors(cudaEventCreate(&end));

checkCudaErrors(cudaMemcpy(a\_dev, a, size, cudaMemcpyHostToDevice));

checkCudaErrors(cudaMemcpy(b\_dev, b, size, cudaMemcpyHostToDevice));

dim3 threadsPerBlock(n, n);

dim3 blocksPerGrid(1, 1);

// GPU Multiplication

checkCudaErrors(cudaEventRecord(start));

matrixMultiplication<<<blocksPerGrid, threadsPerBlock>>>(a\_dev, b\_dev, c\_dev, n);

checkCudaErrors(cudaEventRecord(end));

checkCudaErrors(cudaEventSynchronize(end));

float time = 0.0;

checkCudaErrors(cudaEventElapsedTime(&time, start, end));

checkCudaErrors(cudaMemcpy(c, c\_dev, size, cudaMemcpyDeviceToHost));

// CPU matrix multiplication

int sum = 0;

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

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

sum = 0;

for (int k = 0; k < n; k++) sum = sum + a[row \* n + k] \* b[k \* n + col];

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

}

}

cout << "CPU product is =>\n";

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

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

cout << d[row \* n + col] << " ";

}

cout << "\n";

}

cout << "\n";

cout << "GPU product is =>\n";

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

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

cout << c[row \* n + col] << " ";

}

cout << "\n";

}

cout << "\n";

int error = 0;

int \_c, \_d;

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

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

\_c = c[row \* n + col];

\_d = d[row \* n + col];

error += \_c - \_d;

if (0 != (\_c - \_d)) {

cout << "Error at (" << row << ", " << col << ") => GPU: " << \_c << ", CPU: " << \_d

<< "\n";

}

}

}

cout << "\n";

cout << "Error : " << error;

cout << "\nTime Elapsed: " << time;

return 0;

}