#include <iostream>

#include <cuda\_runtime.h>

#define BLOCK\_SIZE 16

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

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

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

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

int sum = 0;

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

sum += a[row \* size + k] \* b[k \* size + col]; }

c[row \* size + col] = sum; }}

int main() {

int size = 1024;

int matrixSize = size \* size;

int\* hostC = new int[matrixSize];

int\* deviceA, \* deviceB, \* deviceC;

cudaMalloc((void\*\*)&deviceA, matrixSize \* sizeof(int));

cudaMalloc((void\*\*)&deviceB, matrixSize \* sizeof(int));

cudaMalloc((void\*\*)&deviceC, matrixSize \* sizeof(int));

cudaMemcpy(deviceA, hostA, matrixSize \* sizeof(int), cudaMemcpyHostToDevice);

cudaMemcpy(deviceB, hostB, matrixSize \* sizeof(int), cudaMemcpyHostToDevice);

dim3 gridSize((size + BLOCK\_SIZE - 1) / BLOCK\_SIZE, (size + BLOCK\_SIZE - 1) / BLOCK\_SIZE);

dim3 blockSize(BLOCK\_SIZE, BLOCK\_SIZE);

matrixMultiplication<<<gridSize, blockSize>>>(deviceA, deviceB, deviceC, size);

cudaMemcpy(hostC, deviceC, matrixSize \* sizeof(int), cudaMemcpyDeviceToHost);

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

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

std::cout << hostC[i \* size + j] << " ";}

std::cout << std::endl;}

delete[] hostC;

cudaFree(deviceA);

cudaFree(deviceB);

cudaFree(deviceC);

return 0;}