#include <iostream>

#include <cuda\_runtime.h>

\_\_global\_\_ void matrixMul(float \*A, float \*B, float \*C, int N) {

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

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

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

float value = 0;

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

value += A[row \* N + k] \* B[k \* N + col];

}

C[row \* N + col] = value;

}

}

int main() {

int N = 3;

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

float \*A, \*B, \*C, \*d\_A, \*d\_B, \*d\_C;

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

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

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

cudaMalloc(&d\_A, size);

cudaMalloc(&d\_B, size);

cudaMalloc(&d\_C, size);

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

A[i] = (i % N) + 1;

B[i] = ((i % N) + 1) \* 2;

}

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

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

dim3 threadsPerBlock(16, 16);

dim3 blocksPerGrid((N + threadsPerBlock.x - 1) / threadsPerBlock.x,

(N + threadsPerBlock.y - 1) / threadsPerBlock.y);

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

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

std::cout << "Matrix A:" << std::endl;

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

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

std::cout << A[i \* N + j] << " ";

}

std::cout << std::endl;

}

std::cout << "Matrix B:" << std::endl;

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

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

std::cout << B[i \* N + j] << " ";

}

std::cout << std::endl;

}

std::cout << "Calculations (C[i][j] = A[i][k] \* B[k][j]):" << std::endl;

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

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

float value = 0;

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

value += A[i \* N + k] \* B[k \* N + j];

}

std::cout << "C[" << i << "][" << j << "] = ";

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

std::cout << A[i \* N + k] << "\*" << B[k \* N + j];

if (k < N - 1) std::cout << " + ";

}

std::cout << " = " << C[i \* N + j] << std::endl;

}

}

free(A);

free(B);

free(C);

cudaFree(d\_A);

cudaFree(d\_B);

cudaFree(d\_C);

return 0;

}