#include <iostream>

#include <cuda\_runtime.h>

#define N 1024 // Size of the matrix (NxN)

// Kernel function for matrix multiplication

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

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

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

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

int sum = 0;

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

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

}

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

}

}

// Error checking macro for CUDA

#define CHECK\_CUDA\_CALL(call) { \

cudaError\_t err = call; \

if (err != cudaSuccess) { \

std::cerr << "CUDA error: " << cudaGetErrorString(err) << std::endl; \

exit(-1); \

} \

}

int main() {

int \*A, \*B, \*C; // Host matrices

int \*d\_A, \*d\_B, \*d\_C; // Device matrices

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

// Allocate memory on the host

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

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

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

// Allocate memory on the device

CHECK\_CUDA\_CALL(cudaMalloc(&d\_A, size));

CHECK\_CUDA\_CALL(cudaMalloc(&d\_B, size));

CHECK\_CUDA\_CALL(cudaMalloc(&d\_C, size));

// Initialize matrices with random values

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

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

A[i \* N + j] = rand() % 10;

B[i \* N + j] = rand() % 10;

}

}

// Copy matrices from host to device

CHECK\_CUDA\_CALL(cudaMemcpy(d\_A, A, size, cudaMemcpyHostToDevice));

CHECK\_CUDA\_CALL(cudaMemcpy(d\_B, B, size, cudaMemcpyHostToDevice));

// Define block and grid sizes

dim3 threadsPerBlock(16, 16); // Number of threads per block

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

(N + threadsPerBlock.y - 1) / threadsPerBlock.y); // Calculate number of blocks

// Launch kernel for matrix multiplication

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

// Check for kernel launch errors

CHECK\_CUDA\_CALL(cudaGetLastError());

// Synchronize device

CHECK\_CUDA\_CALL(cudaDeviceSynchronize());

// Copy result back to host

CHECK\_CUDA\_CALL(cudaMemcpy(C, d\_C, size, cudaMemcpyDeviceToHost));

// Display input matrices A and B (first 5x5 elements)

std::cout << "First 5x5 elements of Matrix A (Input):" << std::endl;

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

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

}

std::cout << std::endl;

}

std::cout << "\nFirst 5x5 elements of Matrix B (Input):" << std::endl;

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

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

}

std::cout << std::endl;

}

// Display the resulting matrix C (first 5x5 elements)

std::cout << "\nFirst 5x5 elements of Matrix C (Output):" << std::endl;

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

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

}

std::cout << std::endl;

}

// Free memory

free(A);

free(B);

free(C);

CHECK\_CUDA\_CALL(cudaFree(d\_A));

CHECK\_CUDA\_CALL(cudaFree(d\_B));

CHECK\_CUDA\_CALL(cudaFree(d\_C));

return 0;

}