#include <iostream>

#include <cuda\_runtime.h>

#define N 1000000 // Size of the vectors

// Kernel function for vector addition

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

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

if (idx < n) {

C[idx] = A[idx] + B[idx];

}

}

// 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 vectors

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

size\_t size = 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 host vectors with random values

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

A[i] = rand() % 100;

B[i] = rand() % 100;

}

// Copy data 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

int blockSize = 256; // Number of threads per block

int gridSize = (N + blockSize - 1) / blockSize; // Calculate number of blocks

// Launch kernel for vector addition

vectorAdd<<<gridSize, blockSize>>>(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 the input vectors (A and B) and the result vector (C)

std::cout << "First 10 elements of Vector A (Input):" << std::endl;

for (int i = 0; i < 10; i++) {

std::cout << "A[" << i << "] = " << A[i] << std::endl;

}

std::cout << "\nFirst 10 elements of Vector B (Input):" << std::endl;

for (int i = 0; i < 10; i++) {

std::cout << "B[" << i << "] = " << B[i] << std::endl;

}

std::cout << "\nFirst 10 elements of Vector C (Output):" << std::endl;

for (int i = 0; i < 10; i++) {

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

}