Write a CUDA program for addition of two large vector

#include <iostream>

#include <cstdlib>

#include <cuda\_runtime.h>

#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;

// CUDA kernel for vector addition

\_\_global\_\_ void vectorAdd(int \*a, int \*b, int \*result, int n) {

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

if (tid < n) {

result[tid] = a[tid] + b[tid];

}

}

int main() {

int \*a, \*b, \*c; // Host arrays

int \*a\_dev, \*b\_dev, \*c\_dev; // Device arrays

int n = 1 << 4; // Vector size = 16

int size = n \* sizeof(int);

a = new int[n];

b = new int[n];

c = new int[n];

int \*d = new int[n]; // For serial result

// Initialize vectors with random values

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

a[i] = rand() % 1000;

b[i] = rand() % 1000;

d[i] = a[i] + b[i]; // CPU result

}

// Display array A

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

for (int i = 0; i < n; i++) cout << a[i] << ", ";

cout << "\n\n";

// Display array B

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

for (int i = 0; i < n; i++) cout << b[i] << ", ";

cout << "\n\n";

// Allocate device memory

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

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

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

// Create CUDA events for timing

cudaEvent\_t start, end;

checkCudaErrors(cudaEventCreate(&start));

checkCudaErrors(cudaEventCreate(&end));

// Copy host data to device

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

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

int threads = 1024;

int blocks = (n + threads - 1) / threads;

// Record start time

checkCudaErrors(cudaEventRecord(start));

// Launch CUDA kernel

vectorAdd<<<blocks, threads>>>(a\_dev, b\_dev, c\_dev, n);

checkCudaErrors(cudaGetLastError()); // Catch kernel launch errors

// Record end time

checkCudaErrors(cudaEventRecord(end));

checkCudaErrors(cudaEventSynchronize(end));

float time = 0.0;

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

// Copy result from device to host

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

// Print CPU sum

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

for (int i = 0; i < n; i++) cout << d[i] << ", ";

cout << "\n\n";

// Print GPU sum

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

for (int i = 0; i < n; i++) cout << c[i] << ", ";

cout << "\n\n";

// Check for differences

int error = 0;

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

error += abs(d[i] - c[i]);

if (d[i] != c[i]) {

cout << "Error at (" << i << ") => GPU: " << c[i] << ", CPU: " << d[i] << "\n";

}

}

cout << "\nError : " << error;

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

// Cleanup

delete[] a;

delete[] b;

delete[] c;

delete[] d;

cudaFree(a\_dev);

cudaFree(b\_dev);

cudaFree(c\_dev);

return 0;

}