#include <iostream>

#include <cuda\_runtime.h>

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

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

if (idx < N) {

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

}

}

int main() {

int N = 10;

size\_t size = 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; i++) {

A[i] = i + 1;

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

}

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

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

int threadsPerBlock = 256;

int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;

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

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

std::cout << "Vector A: ";

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

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

}

std::cout << std::endl;

std::cout << "Vector B: ";

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

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

}

std::cout << std::endl;

std::cout << "Calculations (A[i] + B[i]):" << std::endl;

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

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

}

free(A);

free(B);

free(C);

cudaFree(d\_A);

cudaFree(d\_B);

cudaFree(d\_C);

return 0;

}