HPC Lab Exp No.4

1.Addition of 2 large vectors:

#include <iostream>

#include <cuda\_runtime.h>

#include <cstdlib>

\_\_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];

}

}

int main() {

int N = 10;

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

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

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

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

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

cudaMalloc(&d\_A, size);

cudaMalloc(&d\_B, size);

cudaMalloc(&d\_C, size);

srand(time(NULL));

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

A[i] = rand() % 100;

B[i] = rand() % 100;

}

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 << "Result Vector C: ";

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

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

}

std::cout << std::endl;

std::cout << "Calculations of Matrix C: " << 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);

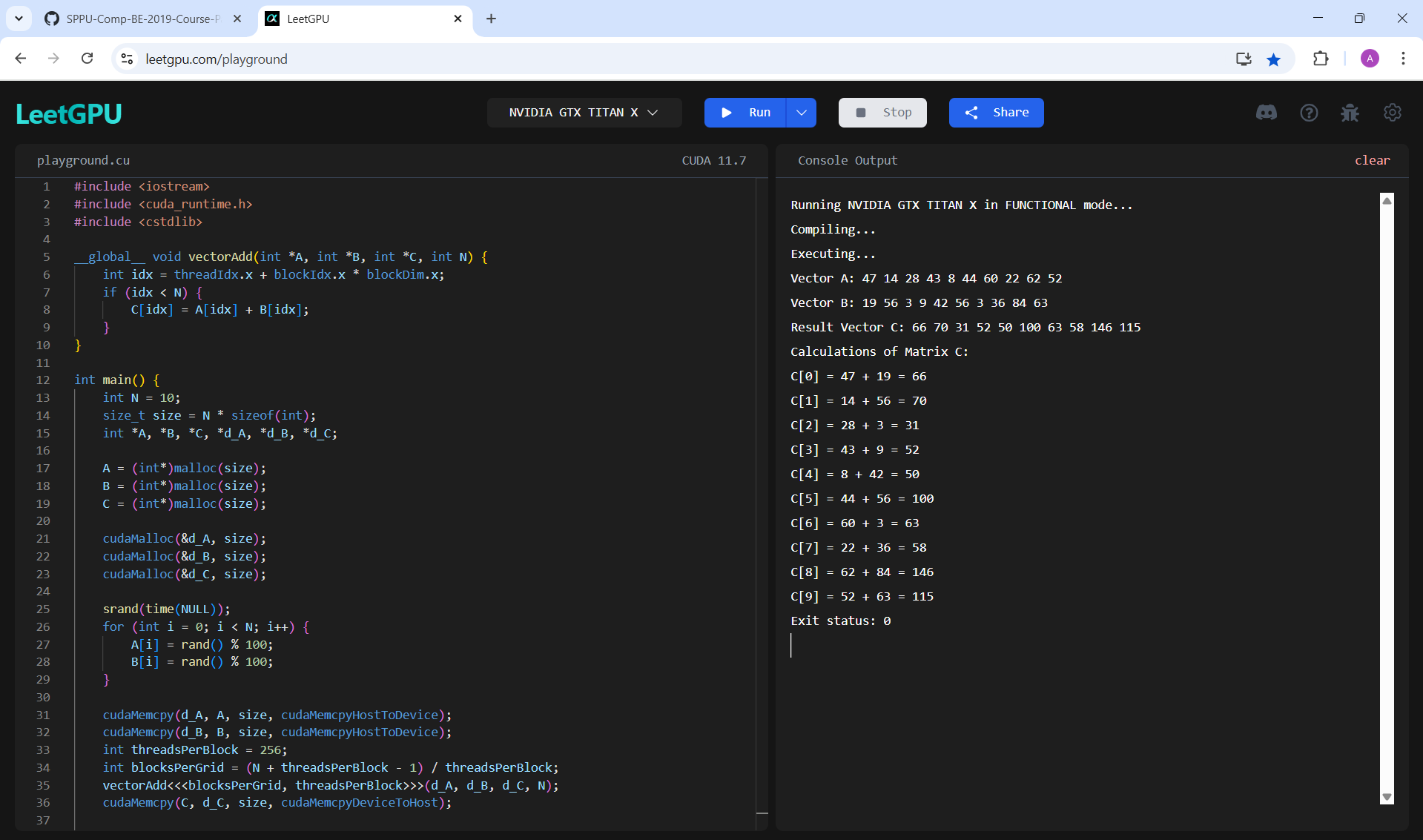
cudaFree(d\_B);

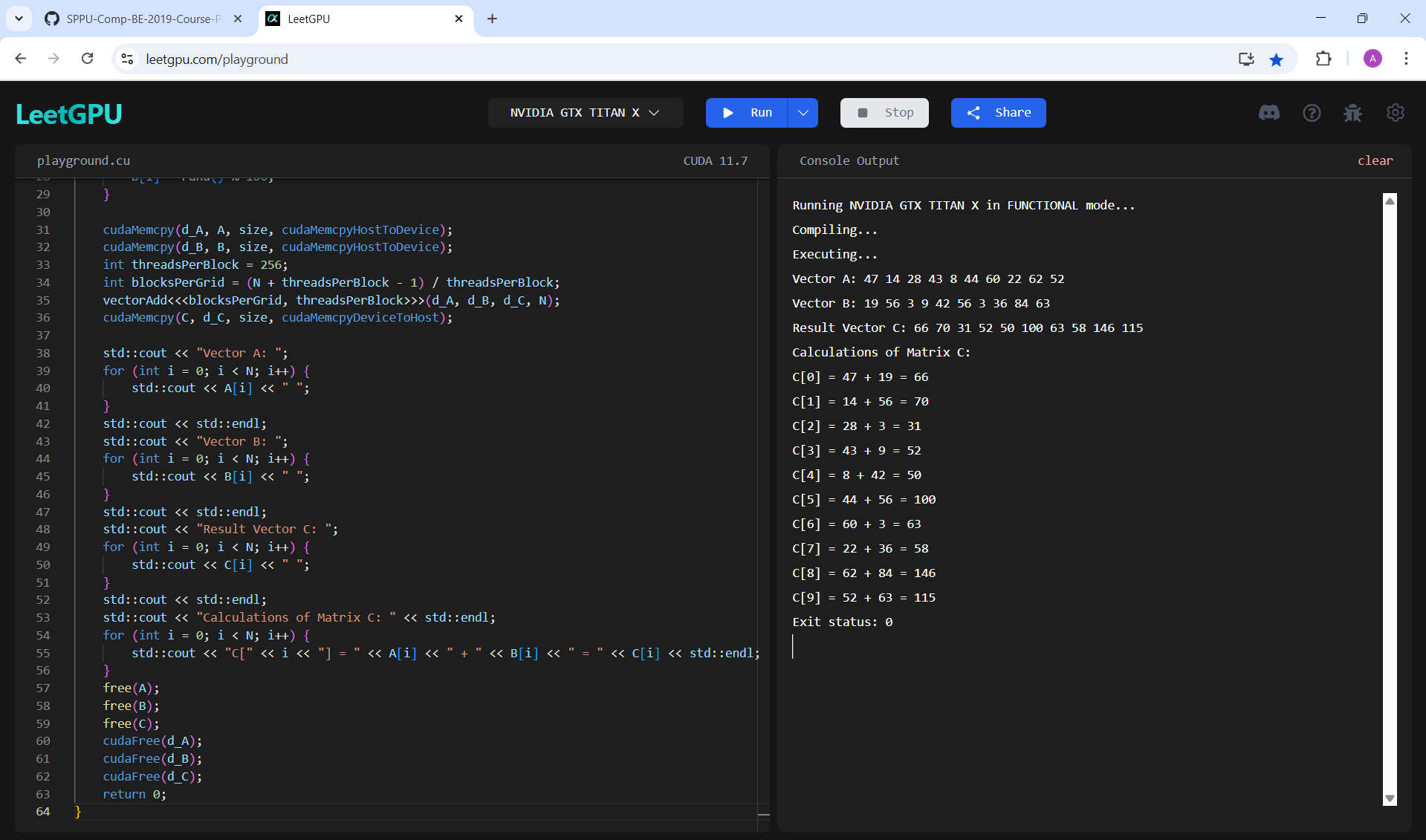
cudaFree(d\_C);

return 0;

}

OUTPUT:





2. Matrix Multiplication:

#include <iostream>

#include <cstdlib>

#include <ctime>

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

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

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

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

int 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(int);

int \*A = (int\*)malloc(size), \*B = (int\*)malloc(size), \*C = (int\*)malloc(size);

int \*d\_A, \*d\_B, \*d\_C;

cudaMalloc(&d\_A, size); cudaMalloc(&d\_B, size); cudaMalloc(&d\_C, size);

srand(time(NULL));

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

B[i] = rand() % 10;

A[i] = rand() % 10;

}

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

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

dim3 threadsPerBlock(16, 16), blocksPerGrid((N + 15) / 16, (N + 15) / 16);

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 << "Result Matrix C : " << std::endl;

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

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

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

}

std::cout << std::endl;

}

std::cout << "Calculation of Matrix C: " << std::endl;

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

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

int value = 0;

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

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

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

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

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

}

std::cout << " = " << value << std::endl;

}

}

free(A); free(B); free(C);

cudaFree(d\_A); cudaFree(d\_B); cudaFree(d\_C);

return 0;

}

OUTPUT:

