HPC Lab Exp No.4

1.Addition of 2 large vectors:

#include <iostream>

#include <cuda\_runtime.h>s

\_\_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);

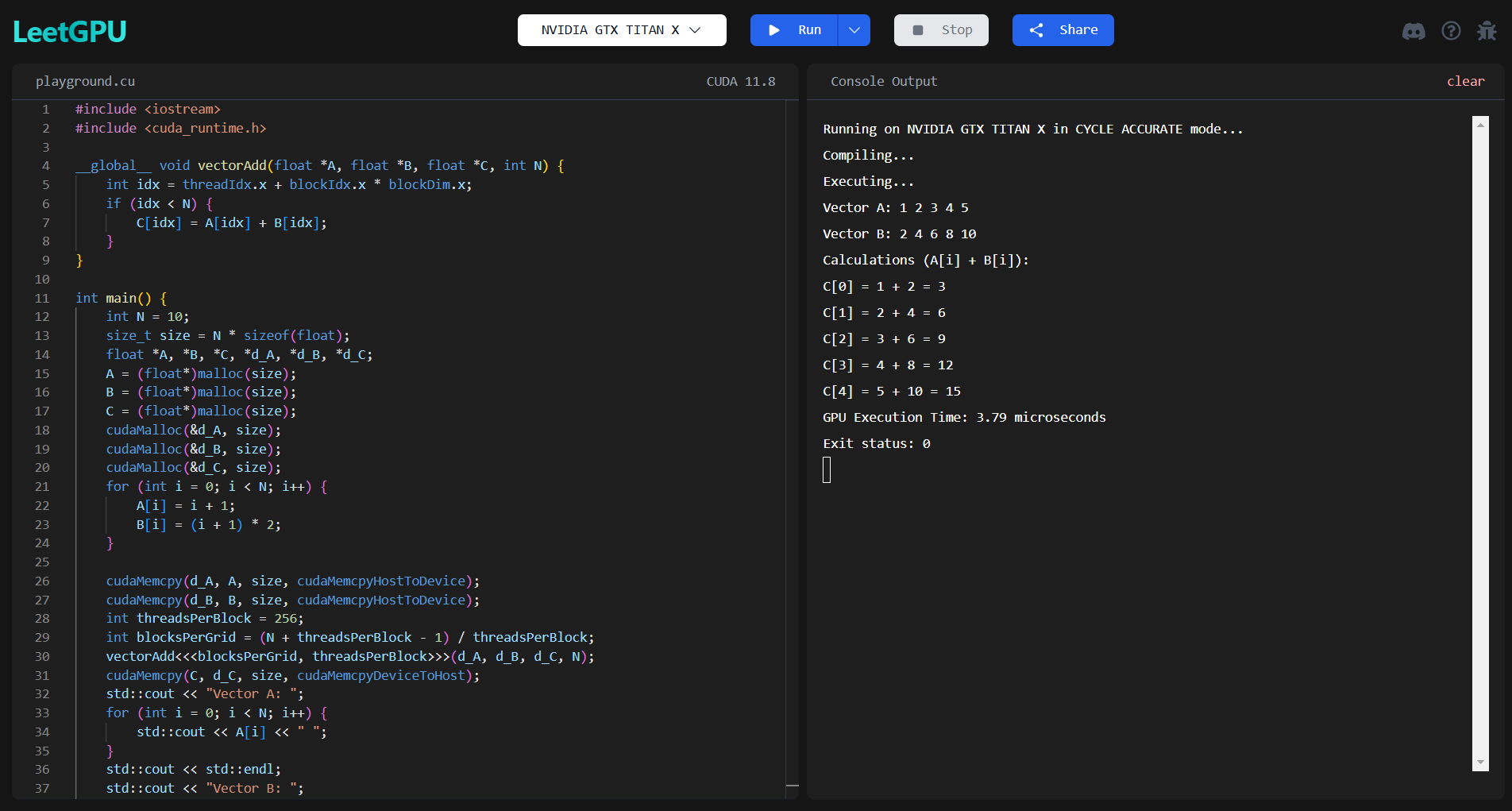
cudaFree(d\_B);

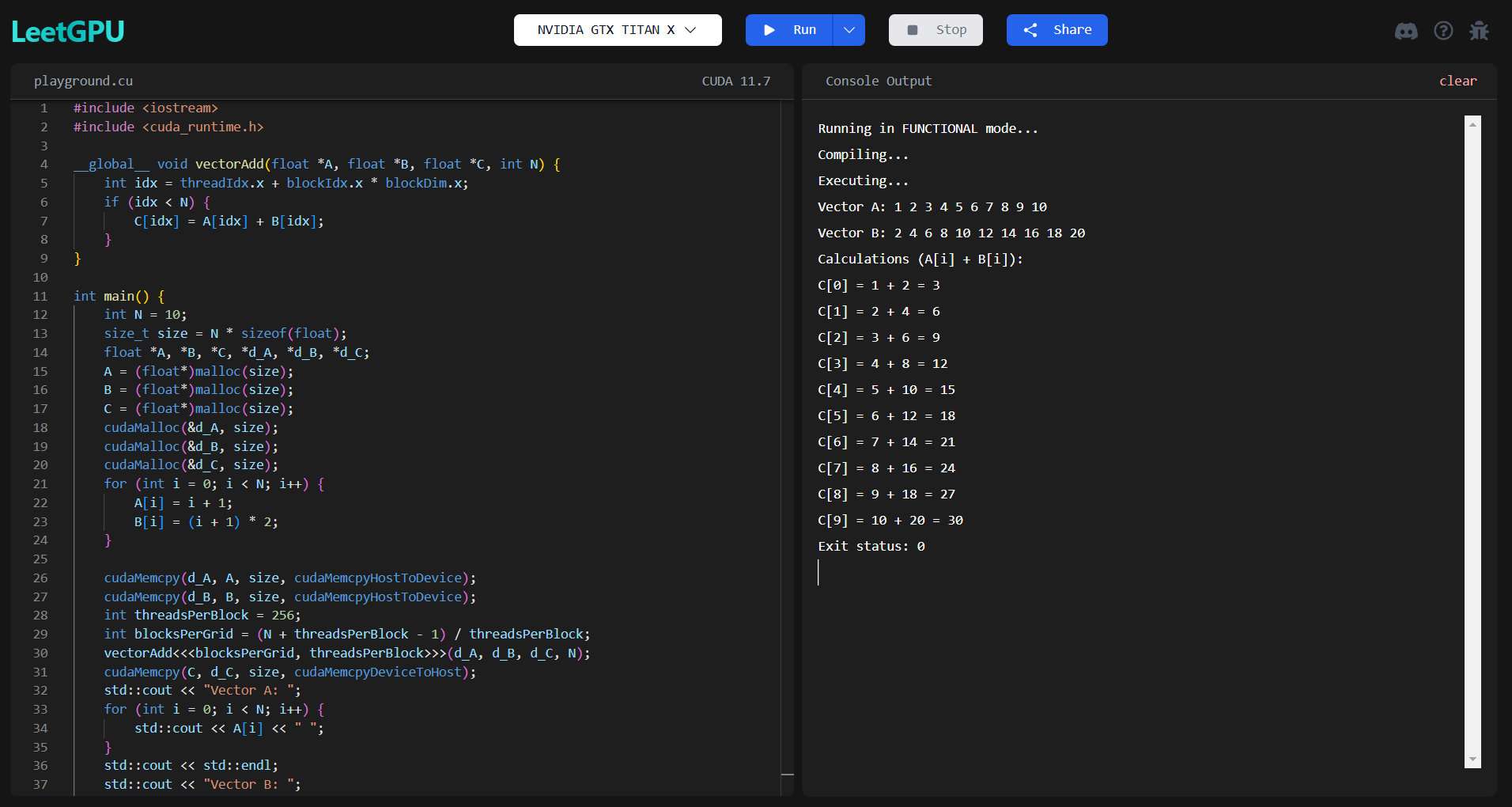
cudaFree(d\_C);

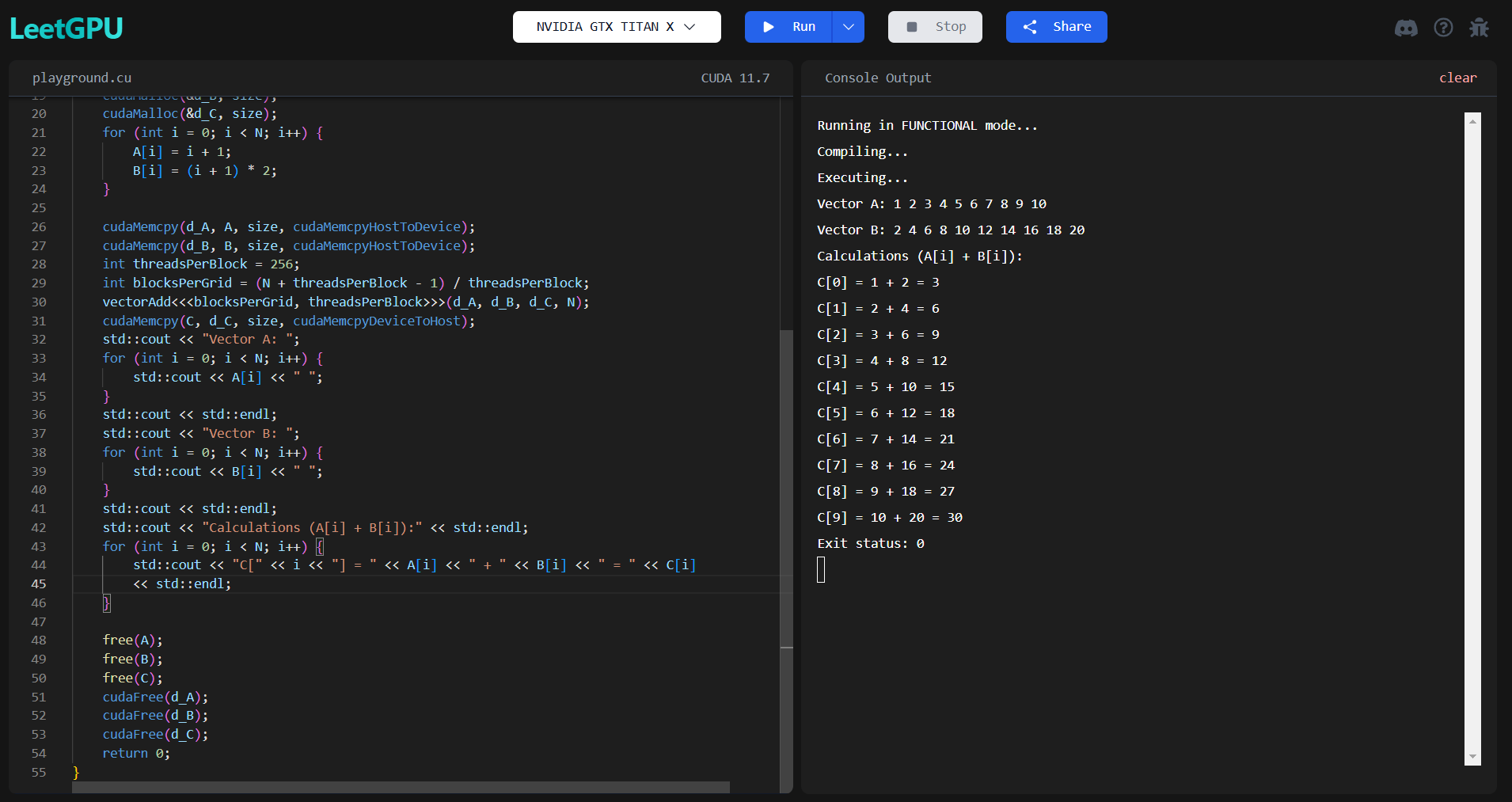
return 0;

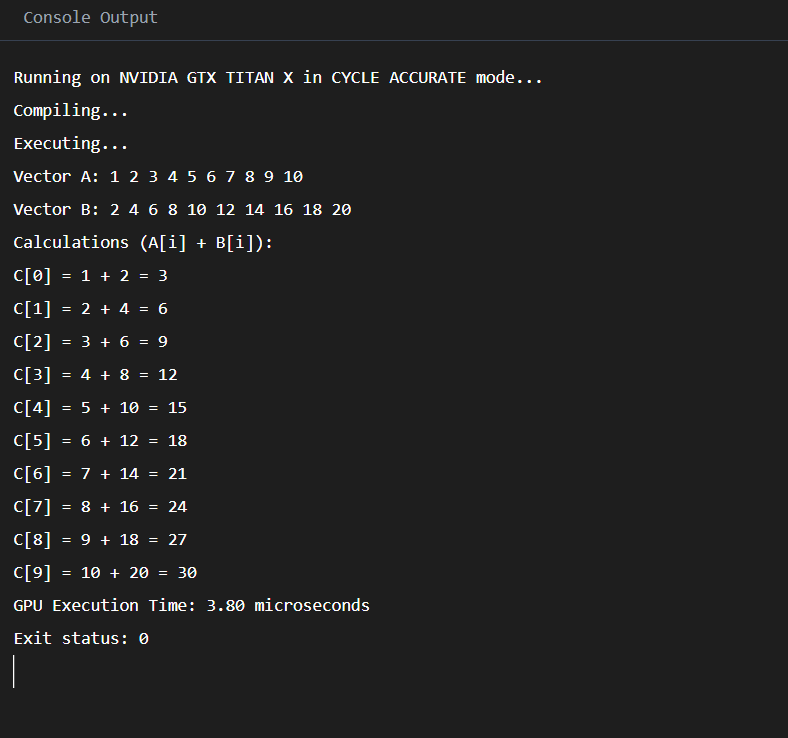
}

OUTPUT:









2. Matrix Multiplication:

#include <iostream>

#include <cuda\_runtime.h>

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

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

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

    if (row < N && col < N) {

        float 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(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 \* N; i++) {

        A[i] = (i % N) + 1;

        B[i] = ((i % N) + 1) \* 2;

    }

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

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

    dim3 threadsPerBlock(16, 16);

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

                         (N + threadsPerBlock.y - 1) / threadsPerBlock.y);

    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 << "Calculations (C[i][j] = A[i][k] \* B[k][j]):" << std::endl;

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++) {

            float value = 0;

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

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

            }

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

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

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

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

            }

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

        }

    }

    free(A);

    free(B);

    free(C);

    cudaFree(d\_A);

    cudaFree(d\_B);

    cudaFree(d\_C);

    return 0;

}

OUTPUT:

