Practical No: 4

CUDA program for addition and multiplication

* These are the steps for parallel CUDA code execution

1. **Visit the Website:**  
Go to [**leetgpu.com**](https://leetgpu.com).

2. **Open Playground:**  
Click on the **“Playground”** option from the menu.

3. **Write or Paste Your CUDA Code:**  
In the editor window, **write** or **paste** your .cu file (CUDA code).

4. **Click on the Run Button:**  
After writing your code, simply click on the **“Run”** button

5 **View the Output:**  
Your code will be **compiled and executed** on a real GPU, and the **output** will appear below

* These are the steps for sequential code execution

For the sequential code, I used Visual Studio Code on my system. I wrote the program in a .cpp file, compiled it using the 'Run' button provided by the C++ extension, and executed it through the terminal. It worked well and gave me the CPU execution time accurately.

Parallel CUDA code :

#include <iostream>

#include <cstdlib>

using namespace std;

\_\_global\_\_ void addKernel(int\* A, int\* B, int\* C, int size) {

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

    if (tid < size) {

        C[tid] = A[tid] + B[tid];

    }

}

\_\_global\_\_ void mulKernel(int\* A, int\* B, int\* C, int size) {

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

    if (tid < size) {

        C[tid] = A[tid] \* B[tid];

    }

}

void initialize(int\* vector, int size) {

    for (int i = 0; i < size; i++) {

        vector[i] = rand() % 10;

    }

}

void print(int\* vector, int size) {

    for (int i = 0; i < size; i++) {

        cout << vector[i] << " ";

    }

    cout << endl;

}

void checkCudaError(cudaError\_t err, const char\* msg) {

    if (err != cudaSuccess) {

        cerr << "CUDA Error (" << msg << "): " << cudaGetErrorString(err) << endl;

        exit(EXIT\_FAILURE);

    }

}

int main() {

    int N = 1000000;

    size\_t bytes = N \* sizeof(int);

    int \*A = new int[N];

    int \*B = new int[N];

    int \*C\_add = new int[N];

    int \*C\_mul = new int[N];

    initialize(A, N);

    initialize(B, N);

    int \*d\_A, \*d\_B, \*d\_C;

    checkCudaError(cudaMalloc(&d\_A, bytes), "cudaMalloc d\_A");

    checkCudaError(cudaMalloc(&d\_B, bytes), "cudaMalloc d\_B");

    checkCudaError(cudaMalloc(&d\_C, bytes), "cudaMalloc d\_C");

    checkCudaError(cudaMemcpy(d\_A, A, bytes, cudaMemcpyHostToDevice), "Memcpy A");

    checkCudaError(cudaMemcpy(d\_B, B, bytes, cudaMemcpyHostToDevice), "Memcpy B");

    int threadsPerBlock = 256;

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

    cudaEvent\_t startAdd, stopAdd;

    cudaEventCreate(&startAdd);

    cudaEventCreate(&stopAdd);

    cudaEventRecord(startAdd);

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

    cudaEventRecord(stopAdd);

    cudaEventSynchronize(stopAdd);

    float timeAdd = 0;

    cudaEventElapsedTime(&timeAdd, startAdd, stopAdd);

    checkCudaError(cudaMemcpy(C\_add, d\_C, bytes, cudaMemcpyDeviceToHost), "Memcpy C\_add");

    cudaEvent\_t startMul, stopMul;

    cudaEventCreate(&startMul);

    cudaEventCreate(&stopMul);

    cudaEventRecord(startMul);

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

    cudaEventRecord(stopMul);

    cudaEventSynchronize(stopMul);

    float timeMul = 0;

    cudaEventElapsedTime(&timeMul, startMul, stopMul);

    checkCudaError(cudaMemcpy(C\_mul, d\_C, bytes, cudaMemcpyDeviceToHost), "Memcpy C\_mul");

    cout << "Vector A: "; print(A, 10);

    cout << "Vector B: "; print(B, 10);

    cout << "Addition : "; print(C\_add, 10);

    cout << "Multiply : "; print(C\_mul, 10);

    cout << "Time taken for addition kernel: " << timeAdd << " ms" << endl;

    cout << "Time taken for multiplication kernel: " << timeMul << " ms" << endl;

    delete[] A;

    delete[] B;

    delete[] C\_add;

    delete[] C\_mul;

    cudaFree(d\_A);

    cudaFree(d\_B);

    cudaFree(d\_C);

    return 0;

}

Output:

Running NVIDIA GTX TITAN X in FUNCTIONAL mode...

Compiling...

Executing...

Vector A: 3 6 7 5 3 5 6 2 9 1

Vector B: 9 3 4 3 1 5 6 3 1 0

Addition : 12 9 11 8 4 10 12 5 10 1

Multiply : 27 18 28 15 3 25 36 6 9 0

Time taken for addition kernel: 166.47 ms

Time taken for multiplication kernel: 197.892 ms

Exit status: 0

For sequential code:

#include <iostream>

#include <cstdlib>

#include <chrono>

using namespace std;

using namespace chrono;

void initialize(int\* v, int n) {

    for (int i = 0; i < n; i++) v[i] = rand() % 10;

}

void add(int\* A, int\* B, int\* C, int n) {

    for (int i = 0; i < n; i++) C[i] = A[i] + B[i];

}

void mul(int\* A, int\* B, int\* C, int n) {

    for (int i = 0; i < n; i++) C[i] = A[i] \* B[i];

}

void print(int\* v, int n) {

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

    cout << endl;

}

int main() {

    int N = 100000000;

    int\* A = new int[N];

    int\* B = new int[N];

    int\* C\_add = new int[N];

    int\* C\_mul = new int[N];

    initialize(A, N);

    initialize(B, N);

    auto startAdd = high\_resolution\_clock::now();

    add(A, B, C\_add, N);

    auto endAdd = high\_resolution\_clock::now();

    double timeAdd = duration<double, milli>(endAdd - startAdd).count();

    auto startMul = high\_resolution\_clock::now();

    mul(A, B, C\_mul, N);

    auto endMul = high\_resolution\_clock::now();

    double timeMul = duration<double, milli>(endMul - startMul).count();

    cout << "Vector A: "; print(A, 10);

    cout << "Vector B: "; print(B, 10);

    cout << "Addition : "; print(C\_add, 10);

    cout << "Multiply : "; print(C\_mul, 10);

    cout << "Time taken for addition: " << timeAdd << " ms" << endl;

    cout << "Time taken for multiplication: " << timeMul << " ms" << endl;

    delete[] A;

    delete[] B;

    delete[] C\_add;

    delete[] C\_mul;

    return 0;

}

Output:

Vector A: 1 7 4 0 9 4 8 8 2 4

Vector B: 9 1 9 2 8 1 4 4 8 8

Addition : 10 8 13 2 17 5 12 12 10 12

Multiply : 9 7 36 0 72 4 32 32 16 32

Time taken for addition: 495.245 ms

Time taken for multiplication: 518.518 ms