Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«**СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**»

Институт космических и информационных технологий
Кафедра «Вычислительной техники»

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1

Изучение модели программирования CUDA

Преподаватель _____ С.А. Тарасов
подпись, дата инициалы, фамилия

Студент КИ20-07Б, 032049287 _____ А.С. Базаров
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2023

ВВЕДЕНИЕ

Цель работы: ознакомление с языком программирования CUDA

Задание:

1. Написать программу на языке CUDA C/C++ для CPU и GPU;
2. Добавить в программу функциональность для сравнения результатов работы этих реализаций по сложности вычисления и возвращаемым значениям;
3. Выполнить вычислительный эксперимент, результатом которого должны быть графики реальных вычислительных сложностей двух реализаций. Интерпретировать результаты эксперимента;
4. Подготовить отчет, который должен содержать исходный код программы (1), названия GPU и CPU (2), графики вычислительных сложностей (3), описание этих графиков (4).

Вариант: Сумма векторов.

Ход работы

1 Реализация программы

Код программы программы пресдтавлен на листинге 1.

Листинг 1 – Код программы

```
%%writefile laba1.cu
#include <cuda_runtime.h>
#include <curand_kernel.h>

extern "C" {

#include <stdio.h>

__host__ void h_add(float* a, float* b, float* c, int n) {
    for (int i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
    }
}


__global__ void d_add(float *a, float *b, float *c, int n) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < n) {
        c[i] = a[i] + b[i];
    }
}


__global__ void d_fill_uniform(
    float *a, float *b, int n, float r, unsigned long long seed) {

    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < n) {
        curandState_t state;
        curand_init(seed, i, 0, &state);
```

```
        a[i] = -r + 2 * r * curand_uniform(&state);
        b[i] = -r + 2 * r * curand_uniform(&state);
    }
}


float compare(float *a, float *b, int n, float eps) {
    float diff = 0;

    for (int i = 0; i < n; i++) {
        diff = fabs(a[i] - b[i]);
        if (diff >= eps) {
            return diff;
        }
    }

    return diff;
}

}

#ifndef REDEFINE
    #define VEC_LEN 51200000
    #define VEC_LEN_INC 512000
    #define CHECK_FIRST 51200
    #define BLOCK_SIZE 128
    #define FNAME_STAMPS "timings.stmp"
    #define PRECISION 10e-10
    #define SEED 27
    #define VEC_MAX_ABS_VAL 101
#endif

#define VEC_MEM_SIZE (VEC_LEN * sizeof(float))
#define ts_to_ms(ts) (ts.tv_sec * 10e3 + ts.tv_nsec * 10e-6)
#define calc_grid_size(m) ((m + BLOCK_SIZE - 1) / BLOCK_SIZE)


int main() {
    float *h_a __attribute__ ((aligned (64)));
    float *h_b __attribute__ ((aligned (64)));
    float *h_c __attribute__ ((aligned (64)));
    float *h_d __attribute__ ((aligned (64)));

    h_a = (float*)malloc(VEC_MEM_SIZE);
    h_b = (float*)malloc(VEC_MEM_SIZE);
    h_c = (float*)malloc(VEC_MEM_SIZE);
    h_d = (float*)malloc(VEC_MEM_SIZE);

    float *d_a, *d_b, *d_c;
```

```
    cudaMalloc((void**)&d_a, VEC_MEM_SIZE);
    cudaMalloc((void**)&d_b, VEC_MEM_SIZE);
    cudaMalloc((void**)&d_c, VEC_MEM_SIZE);

    d_fill_uniform<<<calc_grid_size(VEC_LEN), BLOCK_SIZE>>>(
        d_a, d_b, VEC_LEN, VEC_MAX_ABS_VAL, SEED);
    cudaMemcpy(h_a, d_a, VEC_MEM_SIZE, cudaMemcpyDeviceToHost);
    cudaMemcpy(h_b, d_b, VEC_MEM_SIZE, cudaMemcpyDeviceToHost);

    h_add(h_a, h_b, h_c, CHECK_FIRST);
    d_add<<<calc_grid_size(CHECK_FIRST), BLOCK_SIZE>>>(d_a, d_b, d_c, CHECK_FIRST);
    cudaMemcpy(h_d, d_c, CHECK_FIRST * sizeof(float), cudaMemcpyDeviceToHost);

    if (compare(h_c, h_d, CHECK_FIRST, PRECISION) > PRECISION) {
        printf("Panic!\n");
        return -1;
    }

    float h_time;
    timespec h_start, h_stop;

    float d_time;
    cudaEvent_t d_start, d_stop;
    cudaEventCreate(&d_start);
    cudaEventCreate(&d_stop);

    FILE* file = fopen(FNAME_STAMPS, "w");
    fprintf(file, "Vector Length, CPU Time, GPU Time\n");

    for (int m = VEC_LEN_INC; m <= VEC_LEN; m += VEC_LEN_INC) {
        clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &h_start);
        h_add(h_a, h_b, h_c, m);
        clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &h_stop);
        h_time = (ts_to_ms(h_stop) - ts_to_ms(h_start)); // time in ms

        cudaEventRecord(d_start);
        d_add<<<calc_grid_size(m), BLOCK_SIZE>>>(d_a, d_b, d_c, m);
        cudaEventRecord(d_stop);
        cudaEventSynchronize(d_stop);
        cudaEventElapsedTime(&d_time, d_start, d_stop); // time in ms

        fprintf(file, "%d, %f, %f\n", m, h_time, d_time);
    }

    free(h_a);
    free(h_b);
    free(h_c);
    free(h_d);
```

```
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    fclose(file);

    return 0;
}
```

```
!nvcc laba1.cu -o prog
```

```
!./prog
```

```
import numpy as np
import matplotlib.pyplot as plt

# Compile and run the CUDA program to generate timing data

# Read the timing data from the file
data = np.genfromtxt('timings.stmp', delimiter=',', skip_header=1)
vector_lengths = data[:, 0]
cpu_times = data[:, 1]
gpu_times = data[:, 2]

# Plot the timings
plt.figure(figsize=(10, 6))
plt.plot(vector_lengths, cpu_times, label='CPU Time', marker='o')
plt.plot(vector_lengths, gpu_times, label='GPU Time', marker='o')
plt.xlabel('Vector Length')
plt.ylabel('Time (ms)')
plt.title('CPU vs GPU Time for Vector Addition')
plt.legend()
plt.grid()
plt.show()
```

2 График результатов

График работы программы представлен на рисунке 2.

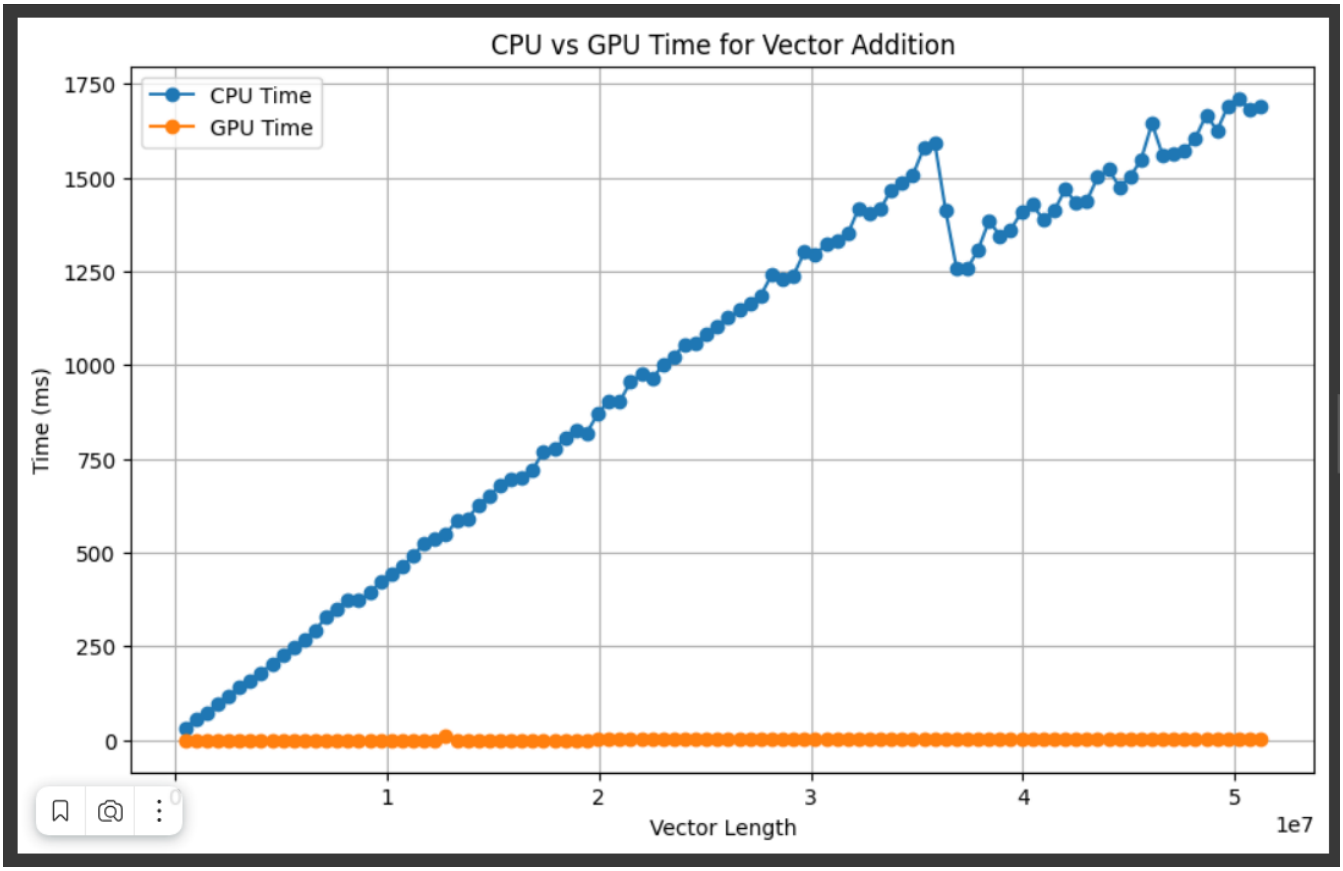

Рисунок 2 – Результат работы