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

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

Студент КИ22-06Б, 032212840 _____ С.Д. Жадан
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Студент КИ22-06Б, 032212840 _____ А.Д. Черненко
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2025

1. Цель работы:

Изучить основы модели параллельного программирования CUDA, реализовать и сравнить производительность функции на центральном (CPU) и графическом (GPU) процессорах, а также проанализировать полученные результаты для понимания преимуществ и особенностей GPU-вычислений.

2. Задание

Сумма векторов.

3. Ход работы

3.1 Аппаратная и программная конфигурация

Для проведения вычислительного эксперимента была использована следующая конфигурация:

- **Центральный процессор (CPU):** Intel Core i3-8130U
- **Графический процессор (GPU):** NVIDIA GeForce MX 130

3.2 Исходный код программы

На рисунке 1 представлен исходный код программы:

```cpp
#include "../includes/vector_add.cuh"

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

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

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

__host__ int vector_add(const float* a, const float* b, float* c, int n) {
  float *d_a = nullptr, *d_b = nullptr, *d_c = nullptr;
  size_t size = n * sizeof(float);

  cudaMalloc(&d_a, size);
  cudaMalloc(&d_b, size);
  cudaMalloc(&d_c, size);

  cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
  cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);

  int blockSize = 256;
  int gridSize = (n + blockSize - 1) / blockSize;

  vector_add_kernel<<<gridSize, blockSize>>>(d_a, d_b, d_c, n);

  cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);

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

  return 0;
}
```

Рисунок 1 – Исходный код программы

3.3 Результаты эксперимента и их анализ

По результатам выполнения программы были построены графики зависимости времени выполнения от размера входных данных (вектора).

На рисунке 2 представлен график зависимости времени выполнения от размера вектора:

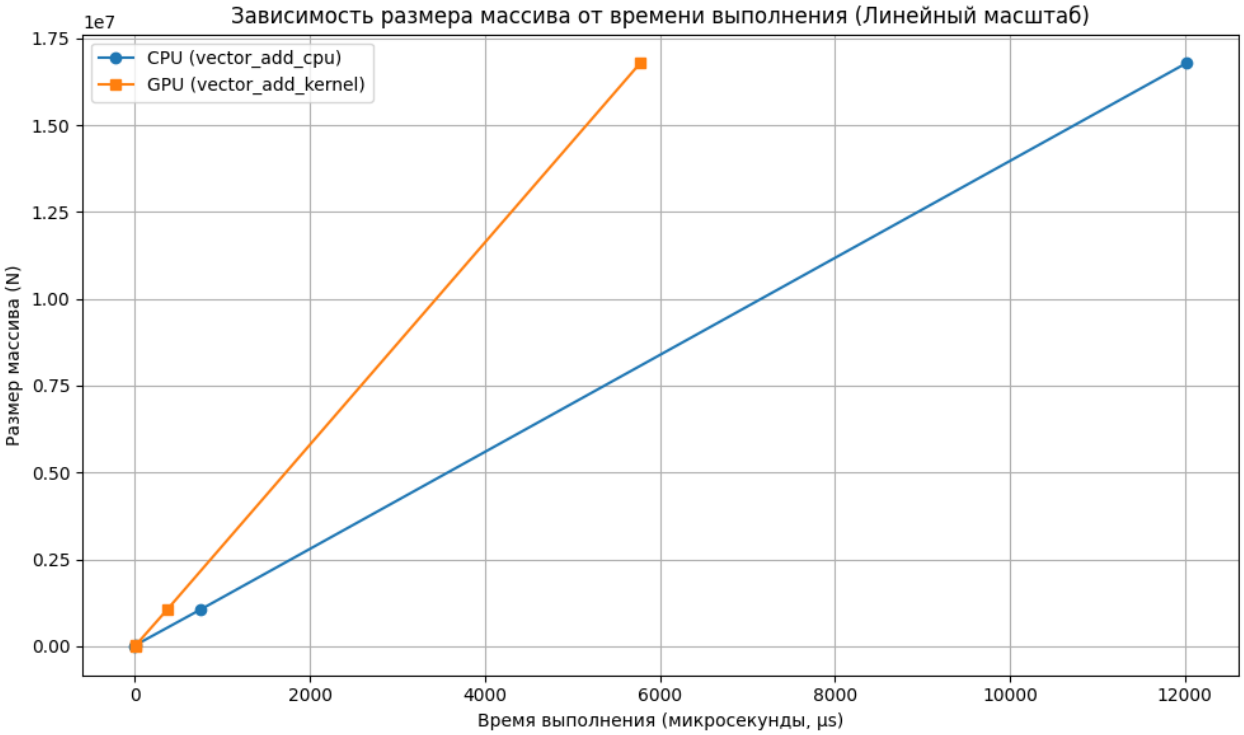

Рисунок 2 – График зависимости времени выполнения от размера вектора

Описание:

На этом графике по оси X отложен размер вектора (количество элементов), а по оси Y — время выполнения в микросекундах.

- **Линия CPU (синяя)** - Представляет собой прямую линию, выходящую из начала координат. Это демонстрирует **линейную вычислительную сложность O(N)**. Время работы напрямую и предсказуемо зависит от количества элементов, так как процессор обрабатывает их строго последовательно в цикле.
- **Линия GPU (оранжевая)** - при малых размерах вектора время выполнения на GPU может быть больше, чем на CPU. Это связано с тем, что время тратится не только на вычисления, но и на выделение памяти на GPU, а также на копирование данных с хоста на устройство и обратно (cudaMemcpy). Однако с увеличением размера вектора (N) наклон этой линии становится меньше, чем у CPU. Это показывает, что время выполнения на GPU растет гораздо медленнее с ростом объема данных.