МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра вычислительной математики и программирования

**спецкурс «Параллельные и распределенные вычисления»**

**ОТЧЕТ**

**Лабораторная работа № 3**

**«****Использование библиотек CUDA»**

Выполнил: Зыкова Д. И.

Группа: М80-114М-21

Преподаватель: Семенов С. А.

Москва, 2021

**Содержание**

[1. Постановка задачи 2](#_heading=h.gjdgxs)

[2. Описание решения 2](#_heading=h.30j0zll)

[3. Аппаратное обеспечение и ПО 2](#_heading=h.1fob9te)

[4. Основные моменты кода 2](#_heading=h.3znysh7)

[5. Результат работы программы 2](#_heading=h.2et92p0)

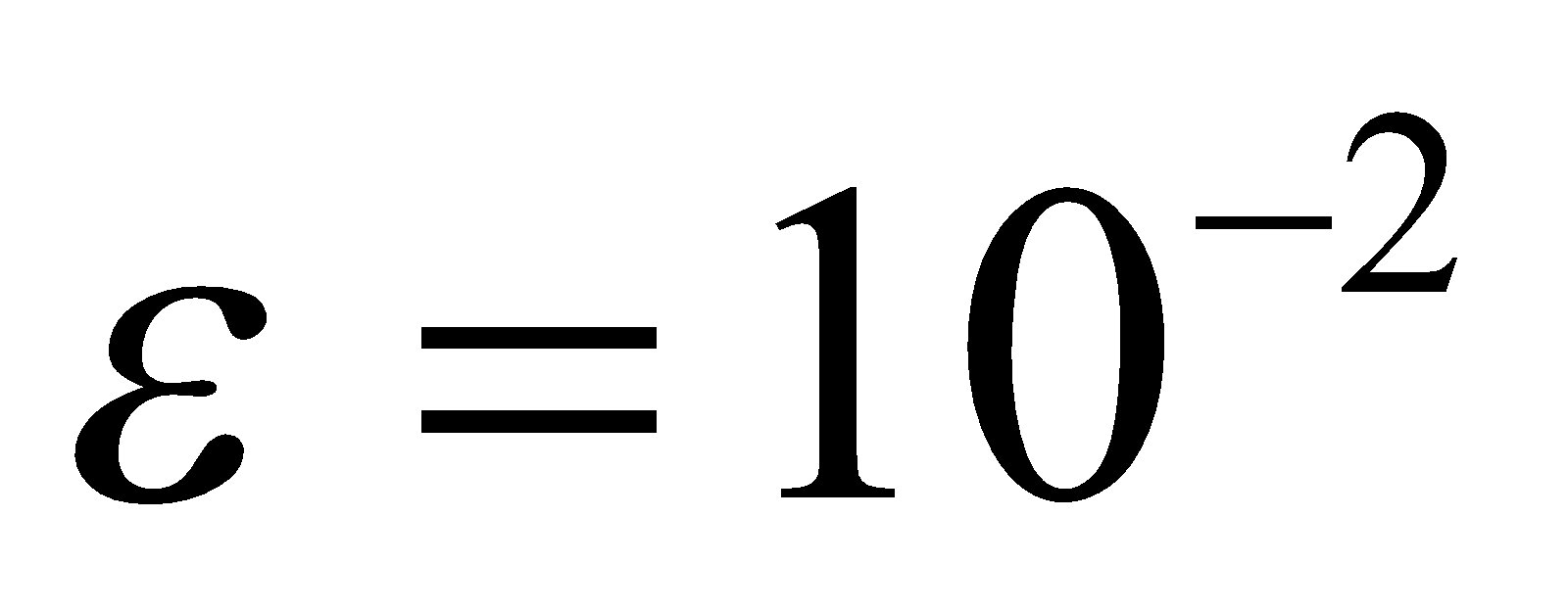
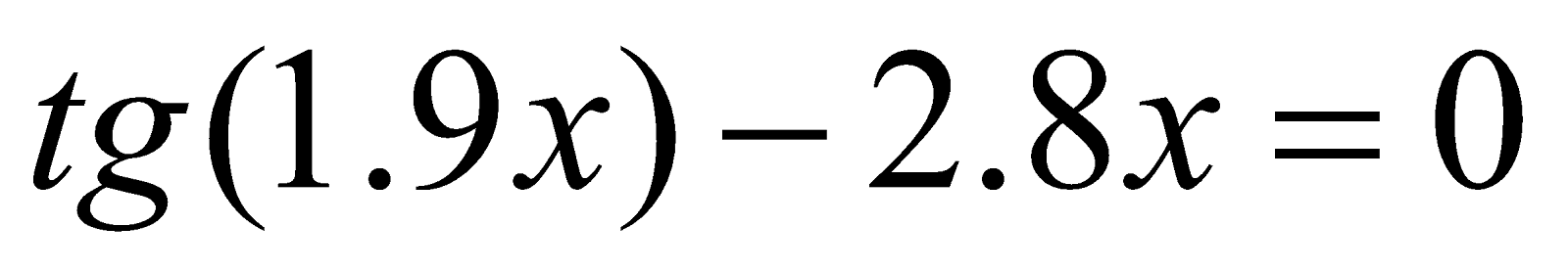
[6. Сравнение скорости выполнения на CPU и GPU 2](#_heading=h.tyjcwt)

[7. Выводы 3](#_heading=h.1t3h5sf)

[8. Приложения 3](#_heading=h.4d34og8)

# **1. Постановка задачи**

Вариант 5

Методом половинного деления с точностью  найти корни уравнения 

# 2. Описание решения

Перед применением метода для поиска корней функции необходимо отделить корни одним из известных способов, например, графическим методом. Отделение корней необходимо в случае, если неизвестно на каком отрезке нужно искать корень.

Рассмотрим отрезок [a, b]. Пусть f(a) имеет отрицательный знак, а f(b) — положительный. Поскольку функция непрерывна, то это означает, что на данном отрезке есть корень. Рассмотрим следующий алгоритм:

Возьмем середину данного отрезка . Если f(с)=0, то мы нашли корень. В противном слечае, допустим, что f(с)>0. Тогда делаем замену b=c. Иначе, если f(c)<0 => делаем замену a=c.

Продолжаем дальше, пока не найдем корень или пока [a, b]>E.

# 3. Аппаратное обеспечение и ПО

Операционная система:

Linux 5.11.0-34-generic x86\_64, Ubuntu 20.04

Аппаратное обеспечение:

1) Процессор: 16\*AMD Ryzen 7 5800X 8-Core Processor

2) Оперативная память: 32Гб

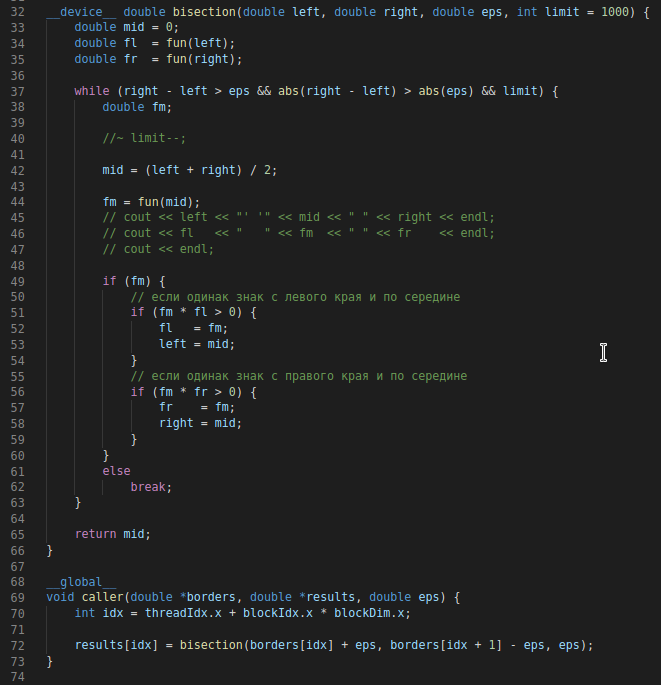
3) Видеокарта: NVIDIA Corporation TU117 [GeForce GTX 1650]

(ядра NVIDIA CUDA 896, 4 ГБ памяти GDDR5, ширина шины данных 128-бит, номинальная частота GPU 1485 МГц)

Программное обеспечение:

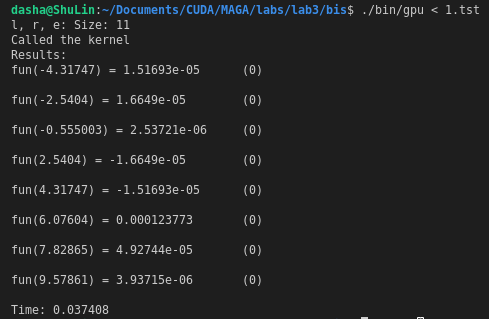
Visual Studio Code, nvcc, gcc

# 4. Основные моменты кода

Для того, чтобы быстро сгенерировать границы отрезков, с которыми мы будем работать, воспользуемся библиотекой thrust. Генерируем вектор, исплодьзуем sequence(), чтобы заполнить его значениями границ.

# 

# 5. Результат работы программы



# 6. Сравнение скорости выполнения на CPU и GPU

Время выполнения программы:

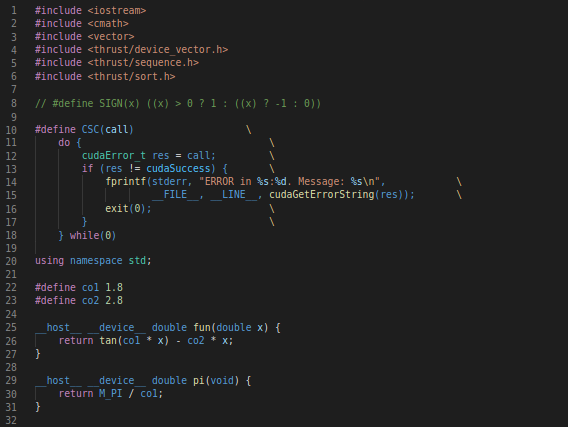
|  |  |  |
| --- | --- | --- |
|  | **GPU**  **время выполнения, мс** | **CPU**  **время выполнения, мс** |
|  | 0.037408 | 0.742432 |
|  |  |  |

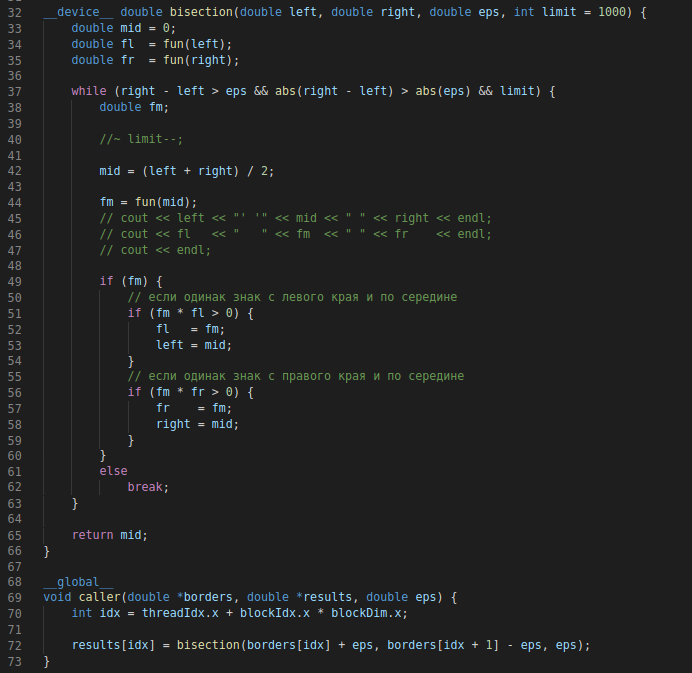
# 7. Выводы

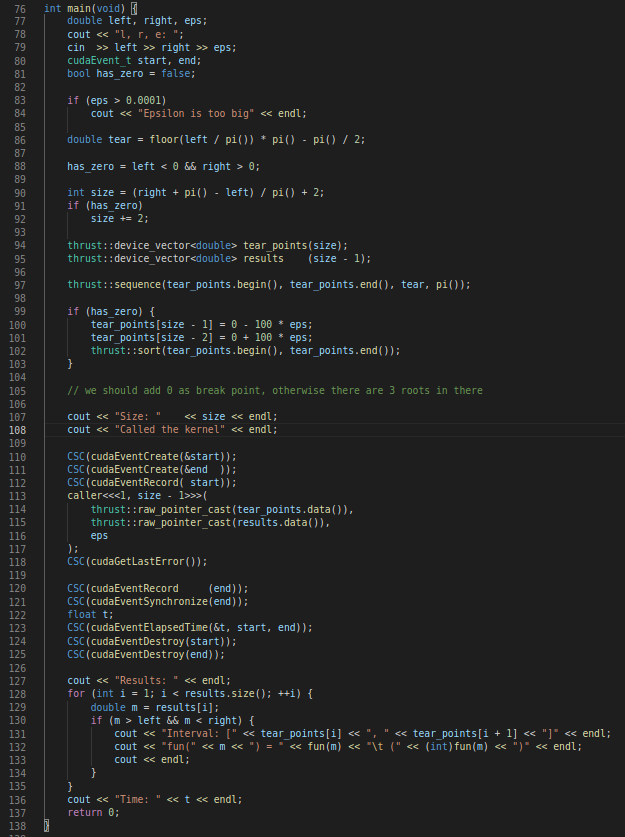
Использование библиотеки thrust оказалось очень удобным потому что можно избежать ручного вызова методов cudaMalloc(), cudaMemcpy().

Это позволяет выполнить на gpu некоторые задачи в том же стиле, в каком они происходят на cpu.

# 8. Приложения





Cсылка на GitHub:

https://github.com/Dashazyk/MAI\_labs/tree/master/cuda