Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное

образовательное учреждение высшего образования

«Южно-Уральский государственный университет

(национальный исследовательский университет)»

Высшая школы электроники и компьютерных наук

Кафедра системного программирования

Итоговое задание  
  
по дисциплине «Технологии параллельного программирования»

Проверил, доцент  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Маковецкая Т. Ю.   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.  
  
Автор работы  
студент группы КЭ-220  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Голенищев А. Б.   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.   
   
Работа защищена с оценкой (прописью, цифрой)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

**Содержание**

[Теоретическое описание алгоритма БПФ 3](#_Toc185228361)

[Практическая реализация 6](#_Toc185228362)

[Синтетические данные 6](#_Toc185228363)

[Данные из набора 7](#_Toc185228364)

[Последовательные вычисления на процессоре 9](#_Toc185228365)

[Использование FFTW 11](#_Toc185228366)

[Собственная реализация алгоритма с параллельными вычислениями 12](#_Toc185228367)

[Установка необходимых стандартов и библиотек 13](#_Toc185228368)

[Настройка параметров сборки проекта 14](#_Toc185228369)

[Основной код программы 15](#_Toc185228370)

[Визуализация выходных данных 17](#_Toc185228371)

[Оценка эффективности параллельного алгоритма 19](#_Toc185228372)

[Расчет эффективности алгоритмов 21](#_Toc185228373)

[Выводы 24](#_Toc185228374)

[Библиографический список 25](#_Toc185228375)

## Теоретическое описание алгоритма БПФ

После второй мировой войны началась гонка вооружений, которая сопровождалась большим количеством испытаний ядреного оружия. После проведения испытаний водородных бомб, представители СССР и США - двух крупнейших ядерных держав заключили Договор о нераспространении ядерного оружия в 1968 г. [1], предполагающий запрет испытаний ядерного оружия под водой, в воздухе, на земле и в космосе. Данный договор не предусматривал подземные испытания ядерного оружия, т.к. существующие технические средства сейсмического контроля не позволяли отслеживать и контролировать такие испытания. Международной группе ученых, включая советских и американских, было поручено разработать техническое решение по мониторингу ядерных подземных испытаний в режиме реального времени. Группа математиков для анализа сигналов сейсмографов пробовала применить алгоритм дискретного преобразования Фурье. Исходные данные - временной ряд (сейсмограмма). Физический смысл разложения временного ряда сигнала в ряд Фурье - получении информации об амплитудном и частотном составе сигнала. Для преобразования в ряд Фурье мы умножаем сложную функцию сигнала на простые гармонические функции, затем интегрируем (считаем результирующую площадь под графиками результирующей функции), Рисунок 1. Так делаем для каждого значения умножаемой частоты.

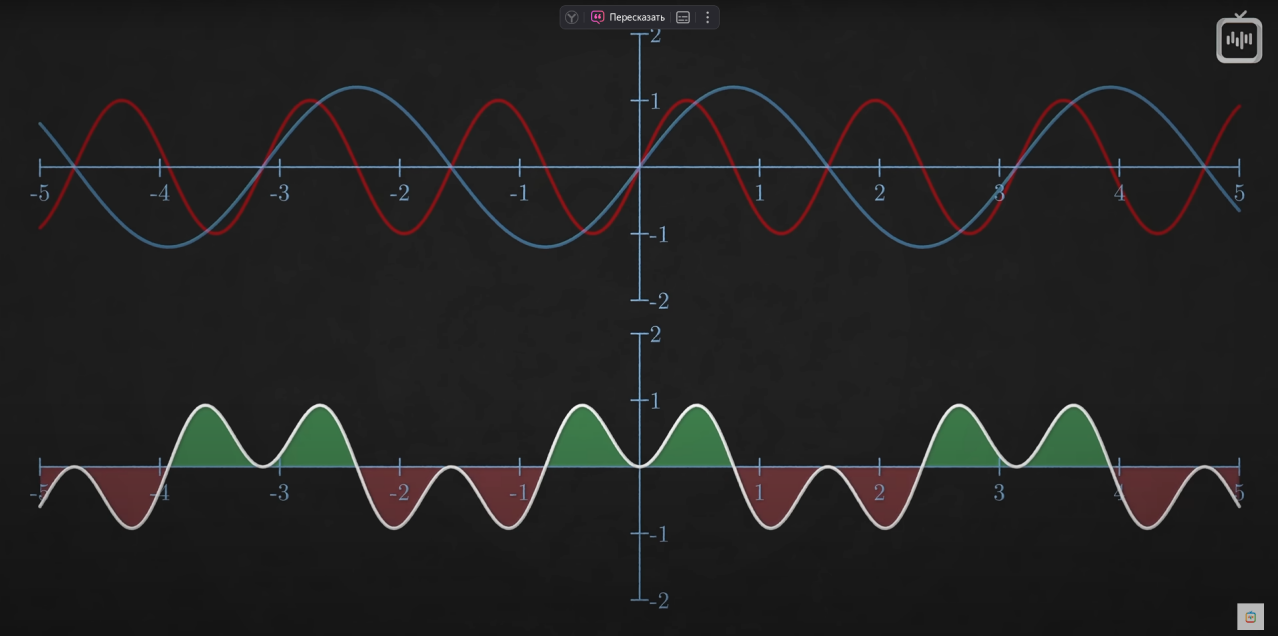


Рисунок 1. Умножение функции исследуемого сигнала на гармоническую

Реальные сигналы являются аналоговыми непрерывными величинами, поэтому мы можем обрабатывать с помощью вычислительных устройств только временные ряды. Для временных рядов применятся дискретное преобразование Фурье (ДПФ) и быстрое преобразование Фурье (БПФ). Преобразование Фурье отображает периодическую функцию c временной в частотную область в дискретной форме по следующей формуле (1). Дискретное преобразование Фурье предполагает умножение временного ряда из N значений на N гармонических функций, что требует высоких вычислительных затрат.

Американский математик Джон Кьюти, а также исследователь компании IBM Джеймс Кули в 1965 году опубликовали статью с решением проблемы производительности дискретного преобразования Фурье [2]. Быстрое преобразование Фурье является алгоритмом, полученным в ходе наблюдений Дж. Кьюти, который зметил, что значения периодических функций повторяются. Оно предполагает значительно меньшие вычислительные затраты, т.к. требуется перемножить не гармонических функций, а , где N - количество элементов выборки временного ряда. Это дает значительное преимущество, когда количество точек велико, Рисунок 2.

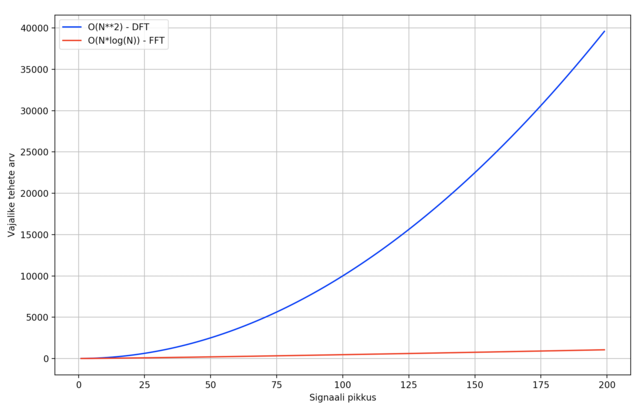


Рисунок 2. Количество вычислений от размера выборки

Для языка C++ существует хорошая реализация библиотеки для дискретного преобразования Фурье, которую разработали в 1999 г. исследователи из Массачусетского Технологического института [3]. Она носит название FFTW (Самое быстрое преобразование Фурье на Западе). Данная библиотека поддерживает MPI, а также OpenMP, Cilk, Rthreads. В данной работе будет показана реализация алгоритма БПФ с использованием OpenACC и OpenMP.

## Практическая реализация

В качестве набора данных для расчетов выбрали 2:

1. Синтетический набор данных (сами рассчитали выборку значений)

2. Взяли один файл dart.csv из набора данных [4], описание которого приведено в статье [5].

### Синтетические данные

using namespace std;

// Глобальные векторы для данных (std::vector)

vector<double> t, u, f, a;

// Искусственный сигнал

double my\_signal(double t) {

    return 3 \* cos(2 \* M\_PI \* 3 \* t + M\_PI / 4) + 2 \* sin(2 \* M\_PI \* 7 \* t - M\_PI / 6) +

           1.5 \* cos(2 \* M\_PI \* 12 \* t) + 0.8 \* sin(2 \* M\_PI \* 20 \* t + M\_PI / 3);

}

// Функция для генерации сигнала

void sample\_signal(double (\*func)(double), int m, vector<double> &x, vector<double> &y) {

    x.clear();

    y.clear();

    double dt = 1.0 / m;

    for (int i = 0; i < m; ++i) {

        double time = i \* dt;

        x.push\_back(time);

        y.push\_back(func(time));

    }

}

Листнинг 1. Реализация функций для создания искусственного временного ряда

Файл содержит следующие данные: время измерения, напряжение канала 1, напряжение канала 2 осциллографа. Осциллограф имеет частоту дискретизации   
1 ГВыб/с, количество измерений в файле – 1000000 значений для обоих каналов   
(все замеры за 1 с проведены). Для констант и тригонометрических функций использована библиотека math.h.

### Данные из набора

Рассмотрим набор данных в текстовом редакторе, Рисунок 3.

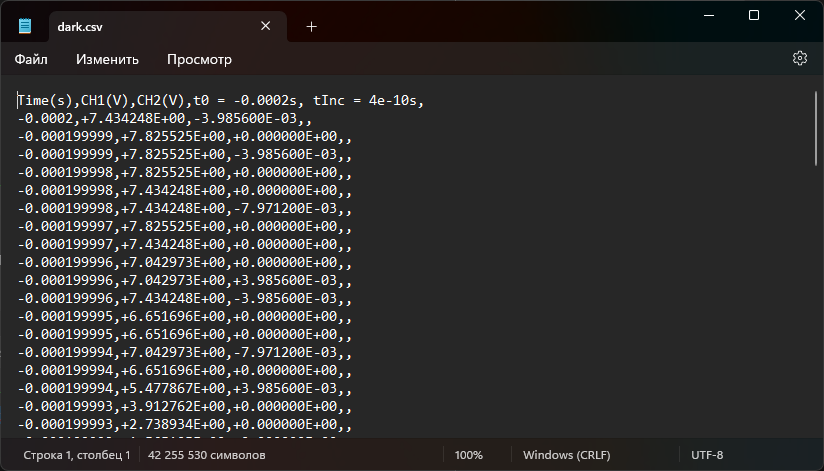


Рисунок 3. Содержание файла dark.csv

Реализуем функцию чтения файла с указанием количества строк, листнинг 2. Данная функция работает с файлом благодаря библиотеке fstream, при чтении функция открывает файл с проверкой его открытия без ошибок (на случай, если путь к файлу указан неверно, например). Удаляет все значения из глобальных векторов, затем построчно считывает и заполняет их, переводя string в double. Чтение данных осуществляется пока line\_count не достигнет max\_lines.

bool read\_csv(const string &filename, vector<double> &x, vector<double> &y, int max\_lines = -1)

{

    ifstream file(filename);

    if (!file.is\_open()) {

        cerr << "Ошибка: не удалось открыть файл " << filename << endl;

        return false;

    }

    x.clear();

    y.clear();

    string line;

    bool header\_skipped = false;

    int line\_count = 0;  // Счётчик обработанных строк

    while (getline(file, line)) {

        if (!header\_skipped) {

            header\_skipped = true;

            continue;  // Пропускаем первую строку (заголовок)

        }

        if (max\_lines != -1 && line\_count >= max\_lines) {

            break;  // Прекращаем чтение, если достигнут лимит строк

        }

        stringstream ss(line);

        string time\_str, ch1\_str;

        if (getline(ss, time\_str, ',') && getline(ss, ch1\_str, ',')) {

            try {

                x.push\_back(stod(time\_str));

                y.push\_back(stod(ch1\_str));

            } catch (const invalid\_argument &e) {

                cerr << "Ошибка: некорректные данные в строке: " << line << endl;

                continue;

            }

        }

        ++line\_count;  // Увеличиваем счётчик строк

    }

    file.close();

    return true;

}

Листнинг 2. Функция чтения данных из набора

### Последовательные вычисления на процессоре

Показана реализация функции для вычисления ДПФ без параллельных стандартов, листнинг 3. Данная функция обрабатывает указанные ей векторы (в аргумент передаем глобальные u – с заполненными данными, f, a – с пустыми, а также шаг дискретизации в секундах). Для вычисления комплексных значений используем тип std::complex. Представлена формула вычисления на каждом шаге (2).

В формуле (2) – значение входного сигнала из временного ряда осциллограммы, – индекс вычисляемой частоты, – комплексный коэффициент вращения (По формуле Эйлера ),  
 – мнимая единица. В цикле для каждого значения вычислим угол фазу сигнала angle, как аргумент тригонометрических функций. Каждое умножаем на комплексное выражение, затем складываем частичные суммы («интегрируем») – находим значение амплитуды для данной частоты. Шаг частоты сигнала вычислется по формуле (3), где N – количество шагов во временном интервале, – шаг по времени, его можно посмотреть в файле с данными или вычислить из частоты дискретизации осциллографа как обратную ей величину. Все вычисленные значения добавляем в конец векторов (динамических массивов).

void serial\_compute\_fft(const vector<double> &u, vector<double> &frequencies, vector<double> &amplitudes, double delta\_t) {

    int N = u.size();

    vector<complex<double>> fft\_result(N);

    for (int k = 0; k < N; ++k) {

        complex<double> sum(0.0, 0.0);

        for (int n = 0; n < N; ++n) {

            double angle = -2.0 \* M\_PI \* k \* n / N;

            sum += u[n] \* complex<double>(cos(angle), sin(angle));

        }

        fft\_result[k] = sum;

    }

    double freq\_step = 1.0 / (N \* delta\_t);

    for (int k = 0; k < N; ++k) {

        frequencies.push\_back(k \* freq\_step);

        amplitudes.push\_back(abs(fft\_result[k]) / N);

}

Листнинг 3. Реализация последовательного алгоритма ДПФ

### Использование FFTW

Напишем функцию с теми же входными данными, что и последовательная. Нужно подготовить динамические массивы сперва для использования с этой библиотекой – используются собственные классы и функции этой библиотеки. Запуск вычислений происходит с помощью функции fftw\_execute(), аргументов которой служит fftw\_plan – поля объекта являются параметрами для расчетов. В данном случае мы указываем конструктор для одномерного ДПФ (задаем в нем количество наших точек, входной и выходной м массивы, тип преобразования (прямое/обратное) и флаг оптимизации (FFTW\_ESTIMATE – план создается быстро без оптимизации)).

void fftw\_compute\_fft(const vector<double> &u, vector<double> &frequencies, vector<double> &amplitudes, double delta\_t) {

    int N = u.size();

    // Выделяем память для FFTW

    fftw\_complex \*in = (fftw\_complex\*) fftw\_malloc(sizeof(fftw\_complex) \* N);

    fftw\_complex \*out = (fftw\_complex\*) fftw\_malloc(sizeof(fftw\_complex) \* N);

    fftw\_plan plan = fftw\_plan\_dft\_1d(N, in, out, FFTW\_FORWARD, FFTW\_ESTIMATE);

    // Инициализируем входные данные

    for (int i = 0; i < N; ++i) {

        in[i][0] = u[i]; // Реальная часть

        in[i][1] = 0.0;  // Мнимая часть

    }

    // Выполняем FFT

    fftw\_execute(plan);

    double freq\_step = 1.0 / (N \* delta\_t);

    for (int k = 0; k < N; ++k) {

        frequencies.push\_back(k \* freq\_step); // Частота

        amplitudes.push\_back(sqrt(out[k][0] \* out[k][0] + out[k][1] \* out[k][1]) / N);

    }

    // Освобождаем ресурсы FFTW

    fftw\_destroy\_plan(plan);

    fftw\_free(in);

    fftw\_free(out);

}

Листнинг 4. Функция, которая использует реализацию вычислений БПФ в FFTW

### Собственная реализация алгоритма с параллельными вычислениями

Идея распараллеливания в собственной реализации состоит в следующем: нам необходимо использовать те инструменты, которые предполагают использование CPU, поскольку во встраиваемых системах и многих ПК отсутствует дискретная видеокарта. По этой причине решено использовать два стандарта распараллеливания программ – OpenACC и OpenMP. Представлен код, листнинг 5.

void my\_compute\_fft(const vector<double> &u, vector<double> &frequencies, vector<double> &amplitudes, double time\_step) {

    int N = u.size();

    vector<complex<double>> fft\_result(N);

#pragma acc data copyin(u[0:N]) copyout(fft\_result[0:N])

#pragma omp parallel for

    for (int k = 0; k < N; ++k) {

        complex<double> sum(0.0, 0.0);

        // Параллельная обработка по n с использованием OpenACC

#pragma acc parallel loop reduction(+:sum)

        for (int n = 0; n < N; ++n) {

            double angle = -2.0 \* M\_PI \* k \* n / N;

            sum += u[n] \* complex<double>(cos(angle), sin(angle));

        }

        fft\_result[k] = sum;

    }

    // Рассчитываем частоты и амплитуды

    double freq\_step = 1.0 / (N \* time\_step);

    for (int k = 0; k < N; ++k) {

        frequencies.push\_back(k \* freq\_step);

        amplitudes.push\_back(abs(fft\_result[k]) / N);

    }

}

Листнинг 5. Код собственной распараллеливающей функции для вычисления ДПФ

В моей реализации OpenMP используется для распараллеливания цикла вычислений на процессоре – задействуя по умолчанию максимальное число его потоков (АЛУ процессора выполняет математические операции быстро – поэтому для распараллеливания цикла является оптимальным решением). OpenACC предполагает, что данные могут быть загружены в видеокарту, количество процессоров в которой значительно больше. Вычислить сумму сразу проще. Мы загружаем исходные данные в ускорительное устройство (видеокарту, которую использует OpenACC). Затем мы с в потоках OpenMP распараллеливаем вычисления внутреннего цикла с помощью OpenACC. Таким образом – основные вычисления выборки мы выполняем на графическом ускорителе NVIDIA, а запись обработку рассчитанных значений (суммы) сохраняют потоки CPU.

### Установка необходимых стандартов и библиотек

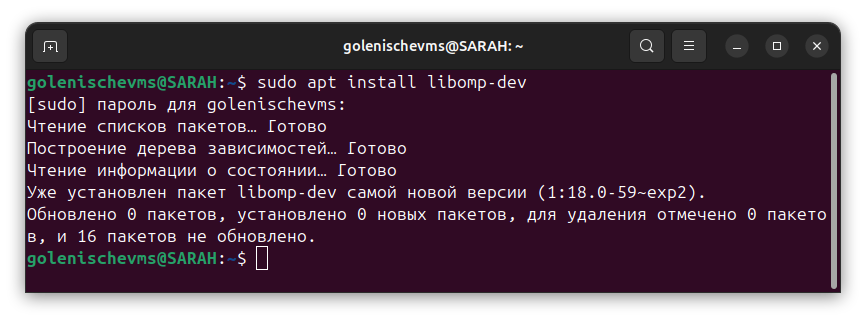
Установим OpenMP в Ubuntu, Рисунок 4. Установим также OpenACC (входит в Cuda Toolkit), Рисунок 5. Установим библиотеку FFTW, Рисунок 6.

Рисунок 4. Установка OpenMP с помощью apt

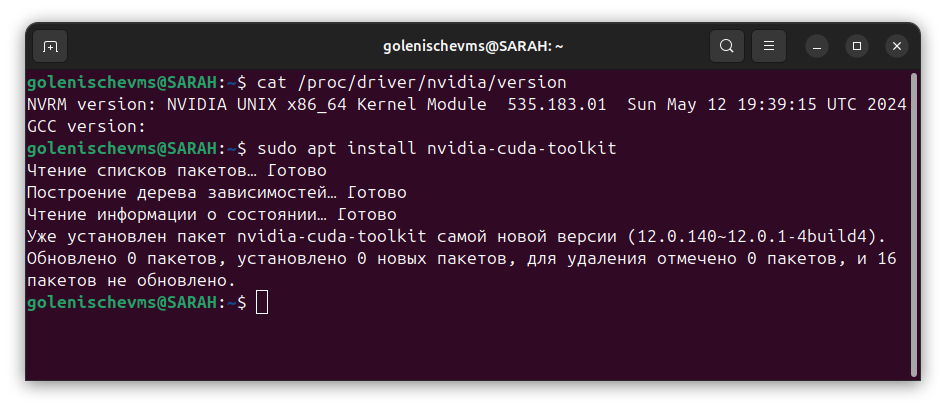


Рисунок 5. Установка инструментов для вычислений на GPU

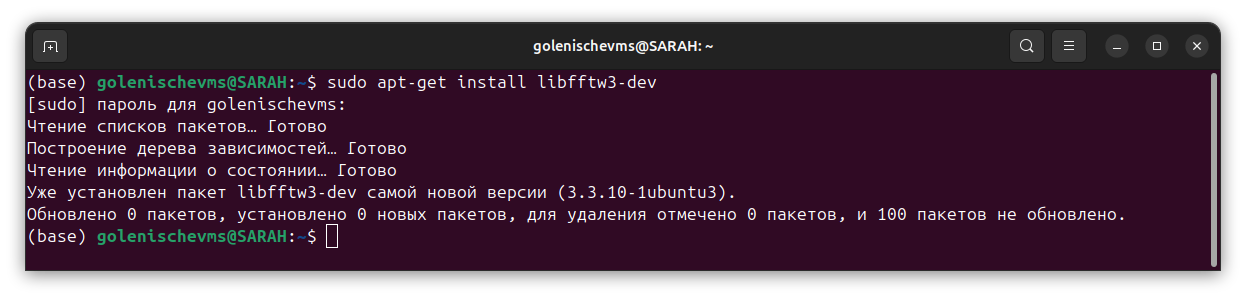


Рисунок 6. Установка FFTW с помощью apt

### Настройка параметров сборки проекта

Для запуска проекта с OpenMP и OpenACC необходимо добавить флаги компилятора в \*.pro файл настроек сборки проекта qmake (Qt), а также добавить пути к библиотекам компилятора с поддержкой OpenACC, флаги библиотеки, листнинг 6.

TEMPLATE = app

CONFIG += console c++17

CONFIG -= app\_bundle

CONFIG -= qt

# Флаги компилятора и линковки для OpenMP

QMAKE\_CXXFLAGS += -fopenmp

QMAKE\_LFLAGS += -fopenmp

# Пути к заголовкам и библиотекам OpenACC

INCLUDEPATH += /snap/freecad/1202/usr/lib/gcc/x86\_64-linux-gnu/11/include

LIBS += -L/snap/freecad/1202/usr/lib/gcc/x86\_64-linux-gnu/11/lib64

# Пути к заголовкам и библиотекам FFTW

INCLUDEPATH += /path/to/fftw/include

LIBS += -L/path/to/fftw/lib -lfftw3 -lm

# Источник кода

SOURCES += \

        main.cpp

Листнинг 6. Содержимое \*.pro файла проекта с OpenMP и OpenACC в QT

### Основной код программы

В основном коде программы мы реализовали как возможность использования как синтетических данных, так и выборки из файла с фиксированным количеством значений. Выборка обрабатывается тремя исследуемыми методами, затем результаты записываются в CSV файлы, для дальнейшей обработки/визуализации данных. Подсчитываем временные затраты на выполнения каждой функции вычислений ДПФ.

Показан пример вывода программы, Рисунок 7. Представлен код программы, листнинг 7.

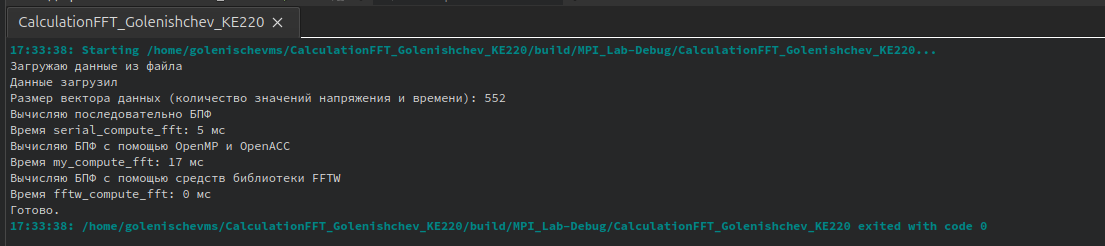


Рисунок 7. Вывод программы с фиксацией временных затрат на вычисление ДПФ каждым исследуемым методом

Проведены несколько тестов с различным количеством значений временного ряда от 500 до 500 000 точек (в файле 1 000 000 точек), Рисунок 8.

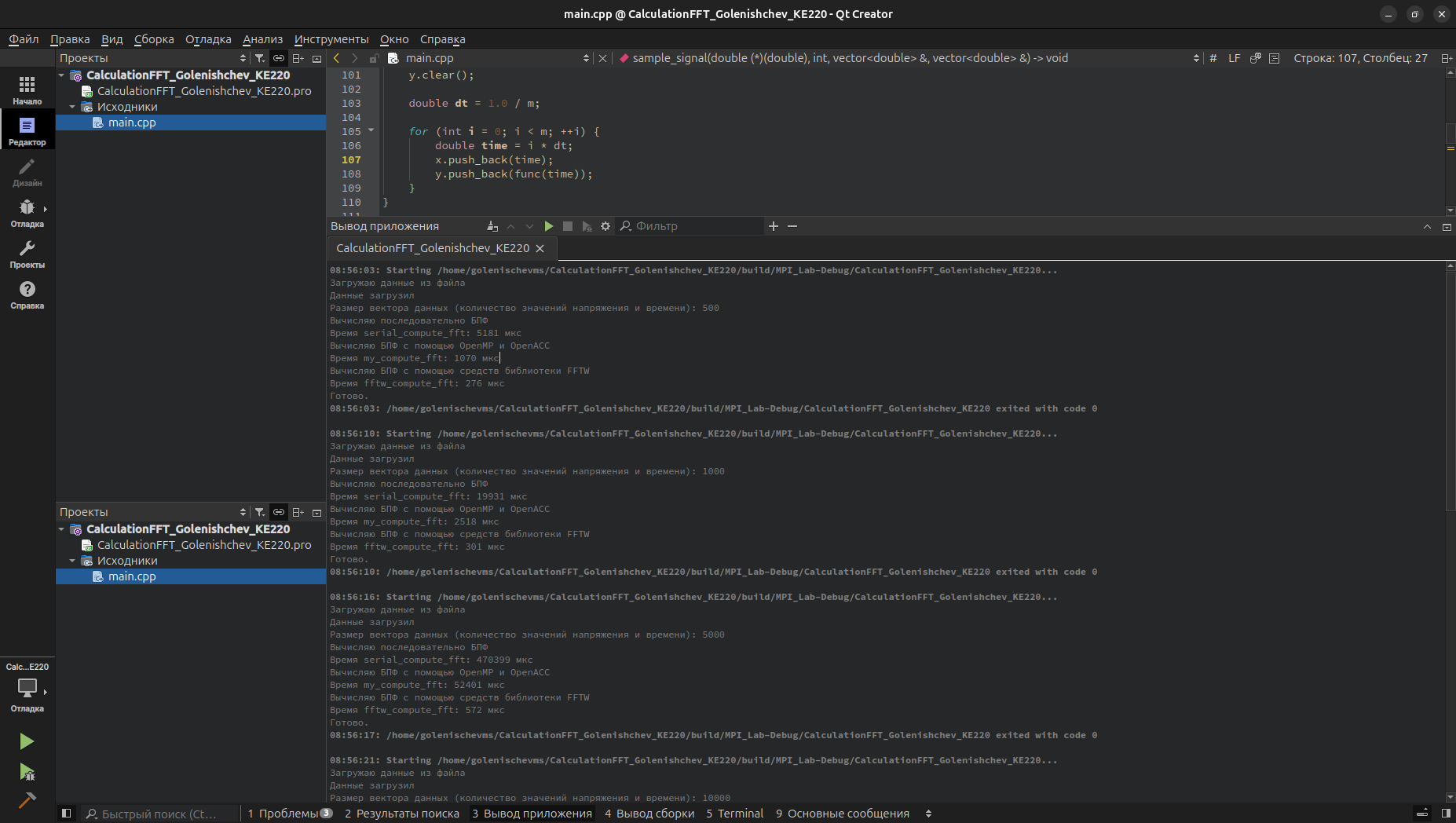


Рисунок 8. Результаты тестовых запусков с временными рядами различной длины

int main() {

    // Тестирование на синтетических данных

    // cout << "Генерирую данные" << endl;

    // const int m = 1000;

    // sample\_signal(my\_signal, m, t, u);

    // Тестирование на данных из файла

    cout << "Загружаю данные из файла" << endl;

    read\_csv("/home/golenischevms/CalculationFFT\_Golenishchev\_KE220/input\_data/dark.csv", t, u, 500000);

    cout << "Данные загрузил" << endl;

    double time\_step = 4e-10;

    vector<double> f\_serial, a\_serial;

    vector<double> f\_my, a\_my;

    vector<double> f\_fftw, a\_fftw;

    cout << "Размер вектора данных (количество значений напряжения и времени): " << u.size() << endl;

    cout << "Вычисляю последовательно БПФ" << endl;

    auto start = chrono::high\_resolution\_clock::now();

    serial\_compute\_fft(u, f\_serial, a\_serial, time\_step);

    auto end = chrono::high\_resolution\_clock::now();

    cout << "Время serial\_compute\_fft: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " мкс" << endl;

    write\_csv("/home/golenischevms/CalculationFFT\_Golenishchev\_KE220/output\_data/serial\_results.csv", t, u, f\_serial, a\_serial);

    cout << "Вычисляю БПФ с помощью OpenMP и OpenACC" << endl;

    start = chrono::high\_resolution\_clock::now();

    my\_compute\_fft(u, f\_my, a\_my, time\_step);

    end = chrono::high\_resolution\_clock::now();

    cout << "Время my\_compute\_fft: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " мкс" << endl;

    write\_csv("/home/golenischevms/CalculationFFT\_Golenishchev\_KE220/output\_data/my\_results.csv", t, u, f\_my, a\_my);

    cout << "Вычисляю БПФ с помощью средств библиотеки FFTW" << endl;

    start = chrono::high\_resolution\_clock::now();

    fftw\_compute\_fft(u, f\_fftw, a\_fftw, time\_step);

    end = chrono::high\_resolution\_clock::now();

    cout << "Время fftw\_compute\_fft: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " мкс" << endl;

    write\_csv("/home/golenischevms/CalculationFFT\_Golenishchev\_KE220/output\_data/fftw\_results.csv", t, u, f\_fftw, a\_fftw);

    cout << "Готово." << endl;

    return 0;

}

Листнинг 7. Основной код программы

### Визуализация выходных данных

На C++ разработано приложение QtWidgets, которое позволяет обработанные выборки визуализировать с помощью библиотеки QCustomPlot, поддерживающей OpenGL, Рисунки 9-11.

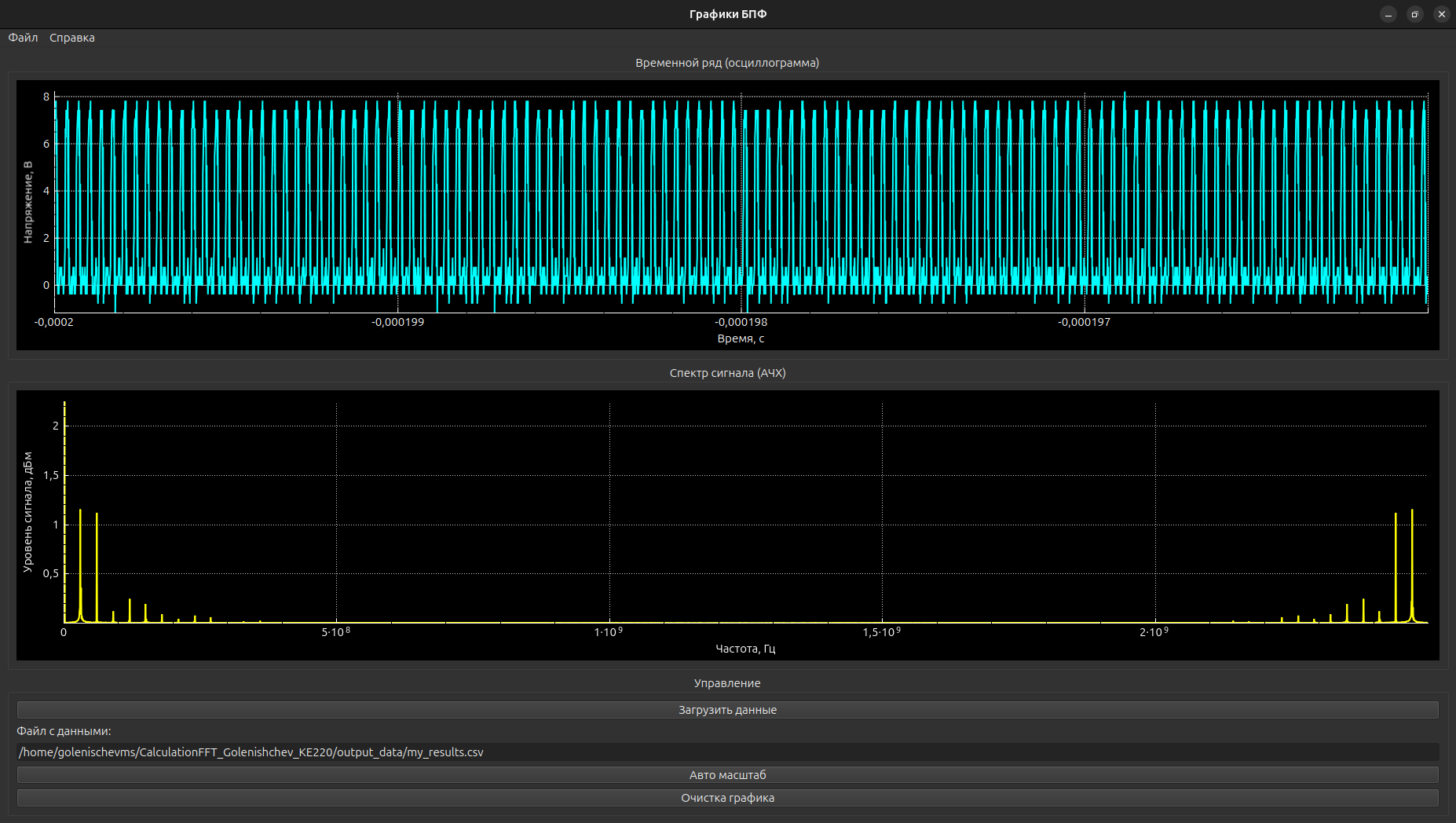


Рисунок 9. Визуализация результатов обработки OpenACC + OpenMP

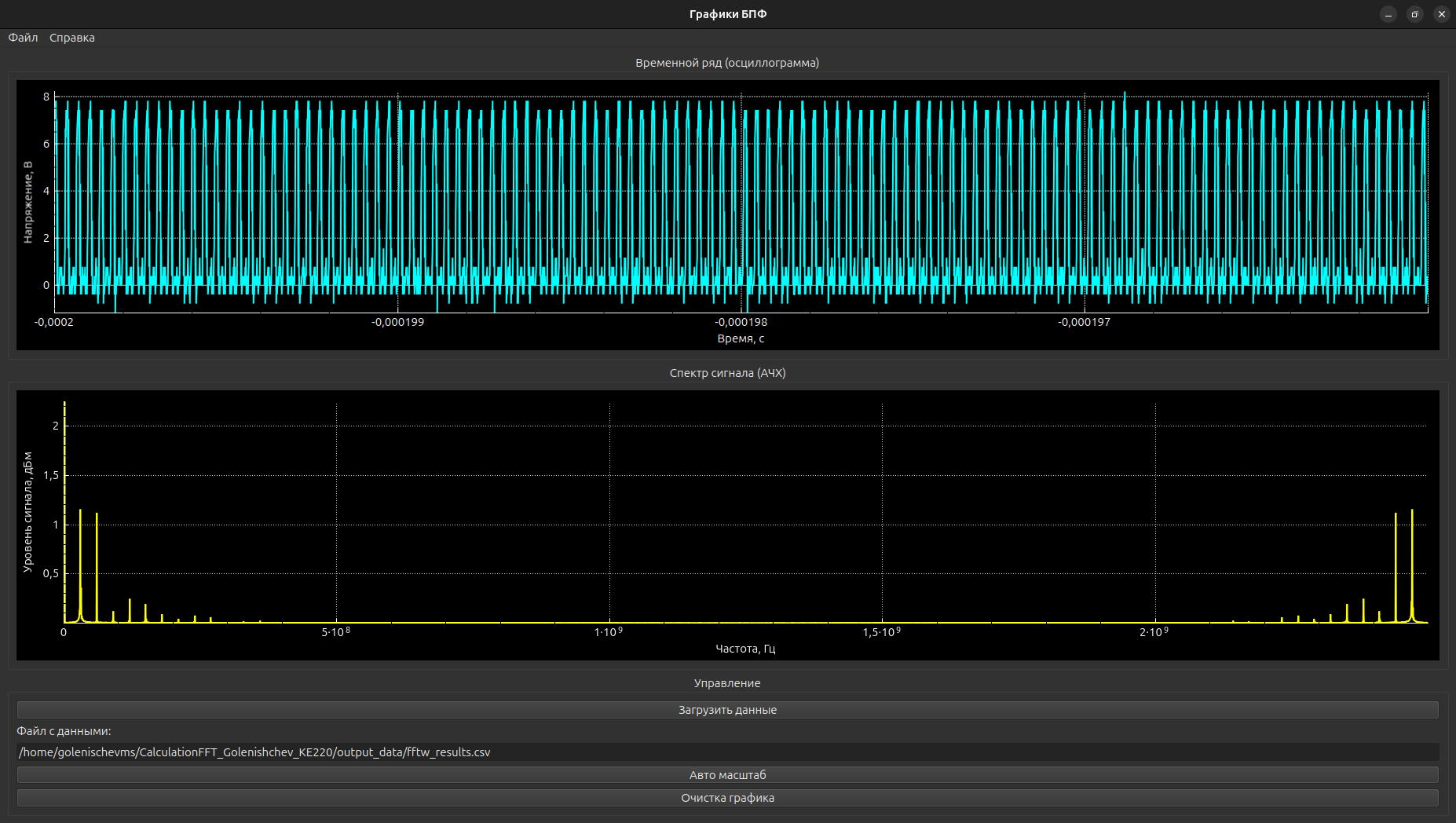


Рисунок 10. Визуализация результатов обработки FFTW



Рисунок 11. Визуализация результатов обработки последовательных вычислений

Построенные графики позволяют оценить результаты работы алгоритмов.

## Оценка эффективности параллельного алгоритма

Представлена таблица, состоящая из рассчитанных значений временных затрат в ходе вычислений тремя методами, Таблица 1.

Таблица 1 – Время расчета БПФ (в мкс) исследуемыми стандартами

|  |  |  |  |
| --- | --- | --- | --- |
| Количество выборок | Последовательные вычисления на CPU | OpenACC + OpenMP | FFTW |
| 500 | 5181 | 1070 | 276 |
| 1000 | 19931 | 2518 | 301 |
| 5000 | 470399 | 52401 | 572 |
| 10000 | 1862332 | 196433 | 725 |
| 50000 | 46004596 | 3789500 | 2506 |
| 100000 | 187269098 | 16958167 | 4788 |
| 500000 | 7473433645 | 649621625 | 22190 |

Визуализируем результаты с помощью matplotlib, Рисунок 12.

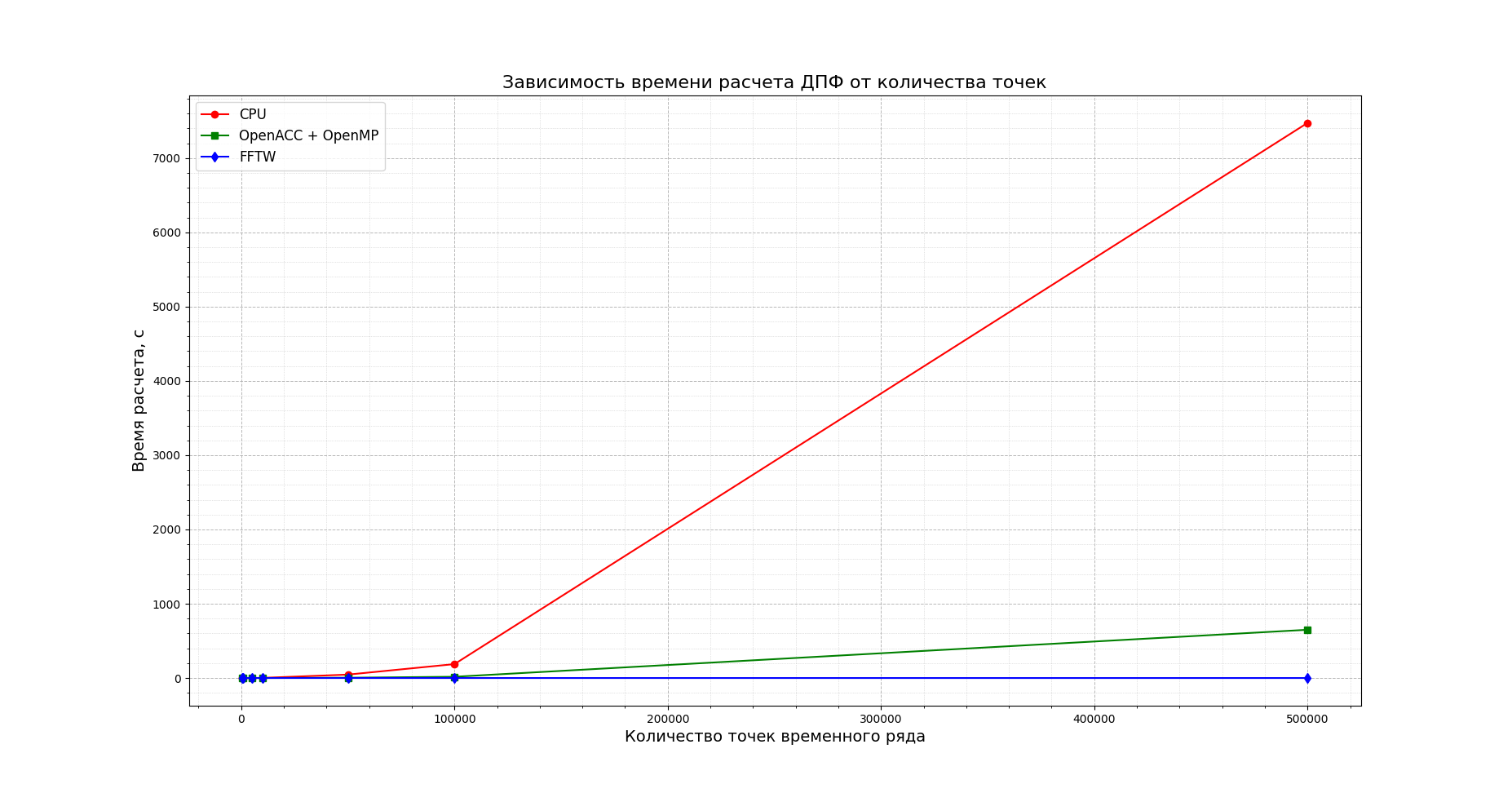


Рисунок 12. График затраченного времени при вычислениях

Представлен код программы для построения графика сравнения результатов временных затрат, листнинг 8.

import matplotlib.pyplot as plt

import numpy as np

# Данные

samples = np.array([500, 1000, 5000, 10000, 50000, 100000, 500000])

cpu\_times = np.array([5181, 19931, 470399, 1862332, 46004596, 187269098, 7473433645]) / 1000000  # Перевод в секунды

openacc\_times = np.array([1070, 2518, 52401, 196433, 3789500, 16958167, 649621625]) / 1000000

fftw\_times = np.array([276, 301, 572, 725, 2506, 4788, 22190]) / 1000000

# Построение графика

plt.plot(samples, cpu\_times, 'o-r', label='CPU')  # Красный кружок

plt.plot(samples, openacc\_times, 's-g', label='OpenACC + OpenMP')  # Зеленый квадрат

plt.plot(samples, fftw\_times, 'd-b', label='FFTW')  # Синий ромб

# Настройка осей

plt.xlabel('Количество точек временного ряда', fontsize=14)

plt.ylabel('Время расчета, с', fontsize=14)

plt.title('Зависимость времени расчета ДПФ от количества точек', fontsize=16)

# Сетка

plt.grid(which='major', linestyle='--', linewidth=0.7, alpha=0.9)  # Основная сетка

plt.grid(which='minor', linestyle=':', linewidth=0.5, alpha=0.7)  # Дополнительная сетка

plt.minorticks\_on()  # Включить дополнительные деления

# Легенда

plt.legend(fontsize=12, loc='upper left')

# Показ графика

plt.show()

Листнинг 8. Код программы для построения графика на Python

Представлены характеристики компьютера, на котором проводились расчеты, Рисунок 13. Процессор – Intel Core i7-13700K (16 ядер 24 потока с тактовой частотой 3,4 ГГц и до 5,4 ГГц в режиме турбо), видеокарта NVIDIA GEFORCE RTX4060TI (2580 МГц, DLSS 3, шейдерные ядра Ada Lovelace проивзодительностью 22 TFLOPS, ядра тарссировки лучей 3th Gen проиводительностью 51 TFLOPS, тензорные ядра 4th Gen производительностью 353 AI TOPS, 16 ГБ GDDR6 видеопамяти).

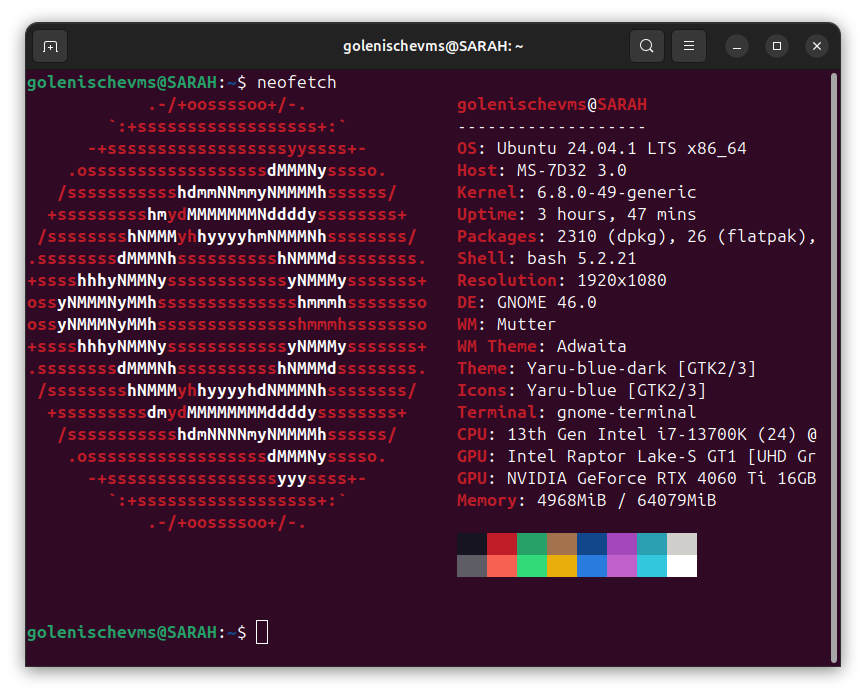


Рисунок 13. Характеристики ПК для тестирования алгоритмов.

### Расчет эффективности алгоритмов

Проведем расчет эффективности алгоритма: вычислим значения ускорения по формуле (4), где – параметр стоимости, – время выполнения последовательного алгоритма для выборок, – время выполнения параллельного алгоритма для n выборок (OpenACC + OpenMP или FFTW).

Проведем расчет стоимости по формуле (5), – стоимость параллельного алгоритма, - время выполнения параллельного алгоритма для n выборок, – число потоков/процессов (расчет проведу для 8 процессов).

Расчеты проведем с помощью программы на языке Python, листнинг 9.

import numpy as np

from prettytable import PrettyTable

import matplotlib.pyplot as plt

# Данные

samples = np.array([500, 1000, 5000, 10000, 50000, 100000, 500000])

time\_cpu = np.array([5181, 19931, 470399, 1862332, 46004596, 187269098, 7473433645])

time\_openacc = np.array([1070, 2518, 52401, 196433, 3789500, 16958167, 649621625])

time\_fftw = np.array([276, 301, 572, 725, 2506, 4788, 22190])

# Число процессоров

p\_openacc = 8  # Количество процессов для OpenACC + OpenMP

p\_fftw = 8     # Количество процессов для FFTW

# Ускорение (Speedup)

speedup\_openacc = time\_cpu / time\_openacc

speedup\_fftw = time\_cpu / time\_fftw

# Стоимость (Cost)

cost\_openacc = time\_openacc \* p\_openacc

cost\_fftw = time\_fftw \* p\_fftw

# Формирование таблицы результатов

results\_table = PrettyTable()

results\_table.field\_names = [

    "Количество выборок", "Ускорение OpenACC + OpenMP", "Ускорение FFTW", "Стоимость OpenACC + OpenMP", "Стоимость FFTW"

]

for i in range(len(samples)):

    results\_table.add\_row([

        samples[i],

        f"{speedup\_openacc[i]:.2f}",

        f"{speedup\_fftw[i]:.2f}",

        f"{cost\_openacc[i]:.2e}",

        f"{cost\_fftw[i]:.2e}"

    ])

print("Таблица 2 - Результаты расчетов эффективности алгоритма:")

print(results\_table)

Листнинг 9. Код программы для расчета ускорения и стоимости

Приведены результаты расчетов, программыы Таблица 2.

Таблица 2 – Результаты расчетов эффективности и стоимости алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество выборок | Ускорение OpenACC + OpenMP | Ускорение FFTW | Стоимость OpenACC + OpenMP | Стоимость FFTW |
| 500 | 4.84 | 18.77 | 8.56e+03 | 1.10e+03 |
| 1000 | 7.92 | 66.22 | 2.01e+04 | 1.20e+03 |
| 5000 | 8.98 | 822.38 | 4.19e+05 | 2.29e+03 |
| 10000 | 9.48 | 2568.73 | 1.57e+06 | 2.90e+03 |
| 50000 | 12.14 | 18357.78 | 3.03e+07 | 1.00e+04 |
| 100000 | 11.04 | 39112.18 | 1.36e+08 | 1.92e+04 |
| 500000 | 11.50 | 336792.86 | 9.02e+08 | 8.88e+04 |

Визуализируем результаты вычислений, Рисунки 14-15.

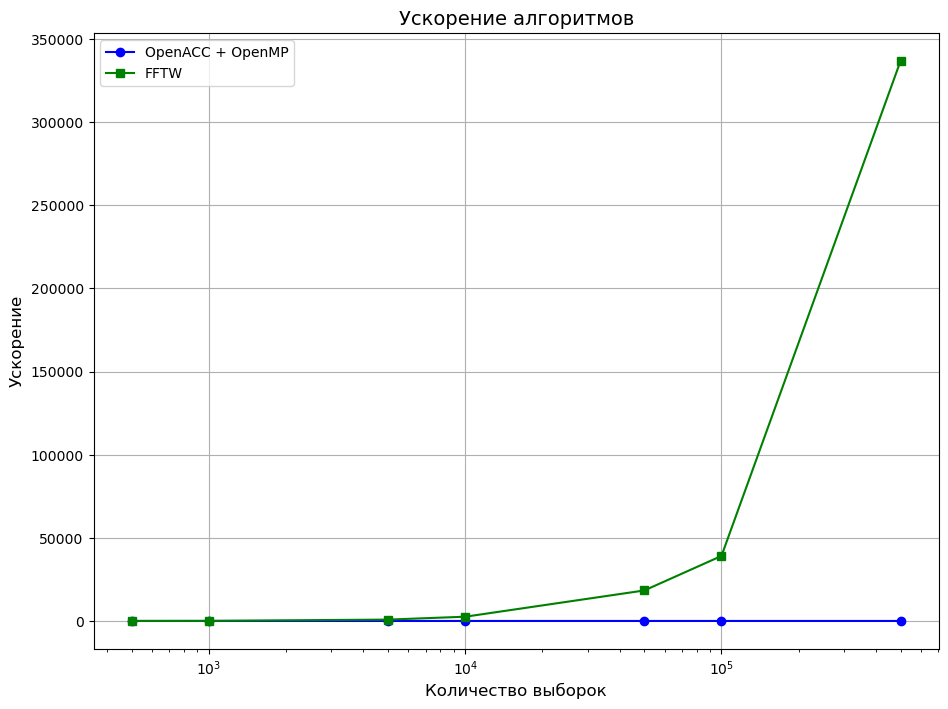


Рисунок 14. График сравнения ускорения алгоритмов для временного ряда различной длины

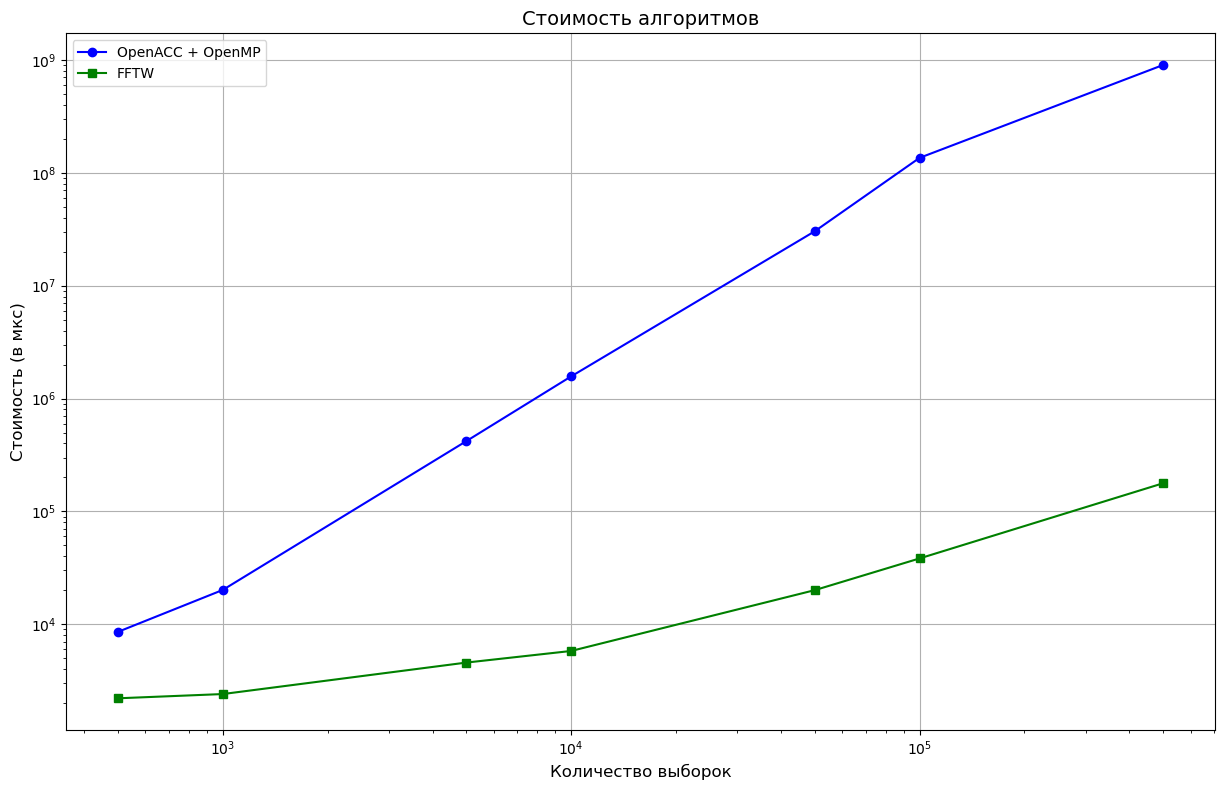


Рисунок 15. График сравнения стоимости алгоритмов для временного ряда различной длины

## Выводы

Расчет эффективности показал, что БПФ, реализованный в FFTW значительно превышает ускорение OpenACC + OpenMP для ДПФ на всех объемах данных, чем больше данных – тем больше разрыв.

Использование параллельных многопоточных вычислений на CPU и GPU позволило значительно ускорить вычисления.

## Библиографический список

1. Договор о нераспространении ядерного оружия [Электронный ресурс] URL: <https://ru.wikipedia.org/wiki/Договор_о_нераспространении_ядерного_оружия>   
   (дата обращения 14.12.2024 г.)
2. Cooley J.W. and Tukey J.W. An algorithm for the machine calculation of the complex fourier series. Mathematics Computation, 19:297-301, 1965.
3. FFTW Home Page [Электронный ресурс] URL: <https://www.fftw.org/> (дата обращения 14.12.2024 г.)
4. Oscilloscope data for SPAD quenched by 100 kohm [Электронный ресурс] URL: <https://figshare.com/articles/dataset/Oscilloscope_data_for_SPAD_quenched_by_100_kohm/19092761/1> (дата обращения: 30.11.2024 г.).
5. Zheng J. et al. Dynamic-quenching of a single-photon avalanche photodetector using an adaptive resistive switch //Nature Communications. – 2022. – Т. 13. – №. 1. – С. 1517.