

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.

Челябинск
2024

Содержание

Теоретическое описание алгоритма БПФ	3
Практическая реализация	6
Синтетические данные.....	6
Данные из набора	7
Последовательные вычисления на процессоре	9
Использование FFTW	11
Собственная реализация алгоритма с параллельными вычислениями	12
Установка необходимых стандартов и библиотек.....	13
Настройка параметров сборки проекта.....	14
Основной код программы.....	15
Визуализация выходных данных	18
Расчет эффективности алгоритмов.....	21
Выводы	21
Библиографический список.....	22
Приложение А.....	23

Теоретическое описание алгоритма БПФ

После второй мировой войны началась гонка вооружений, которая сопровождалась большим количеством испытаний ядерного 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.

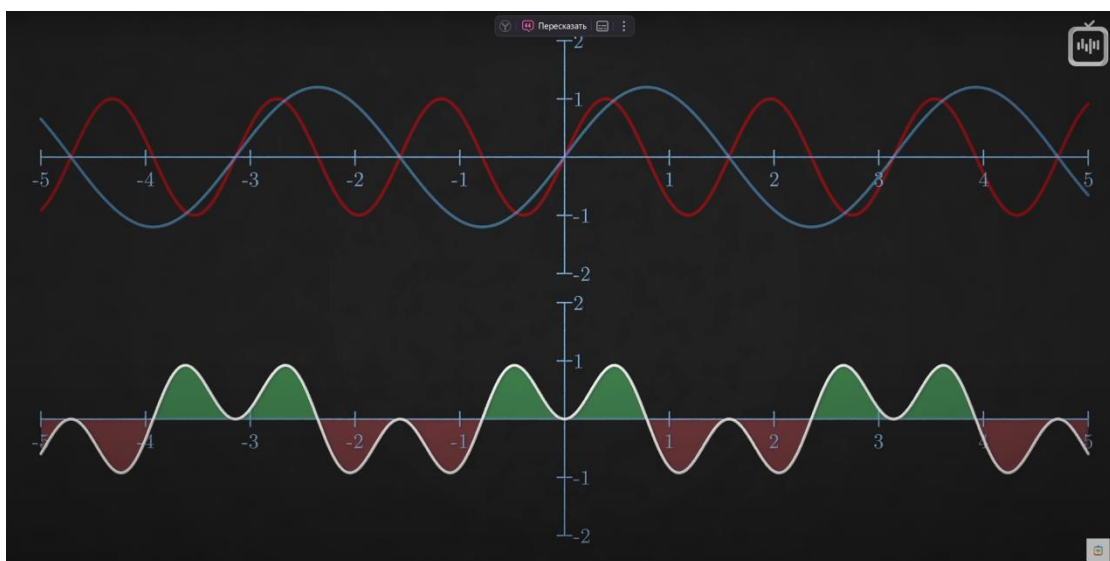


Рисунок 1. Умножение функции исследуемого сигнала на гармоническую

Реальные сигналы являются аналоговыми непрерывными величинами, поэтому мы можем обрабатывать с помощью вычислительных устройств только временные ряды. Для временных рядов применяется дискретное преобразование Фурье (ДПФ) и быстрое преобразование Фурье (БПФ). Преобразование Фурье отображает периодическую функцию $f(t)$ с 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.

$$(1) F_k = \frac{1}{n} \sum_{j=0}^{n-1} f_j e^{-2\pi i \left(\frac{jk}{n}\right)}, k = 0, 1, \dots, n-1, f_k = f(x_k)$$

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 заметил, что значения периодических функций повторяются. Оно предполагает значительно меньшие вычислительные затраты, т.к. требуется перемножить не N^2 гармонических функций, а $N \log_2 N$, где N - количество элементов выборки временного ряда. Это дает значительное преимущество, когда количество точек велико, Рисунок 2.

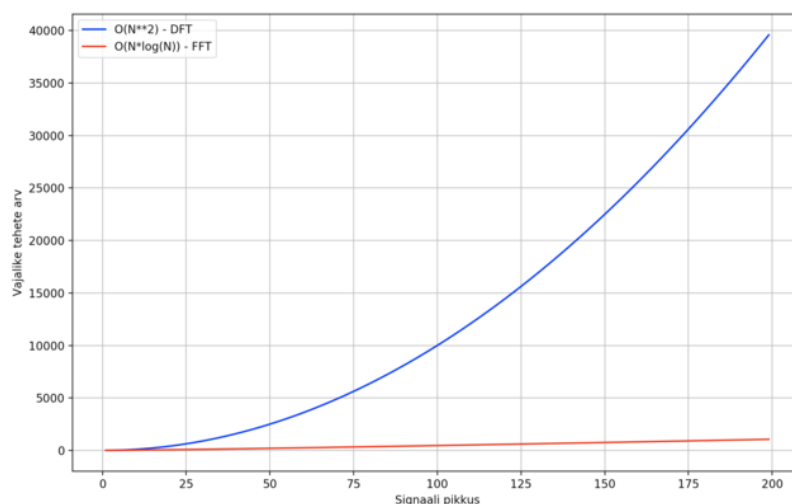


Рисунок 2. Количество вычислений от размера выборки

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;

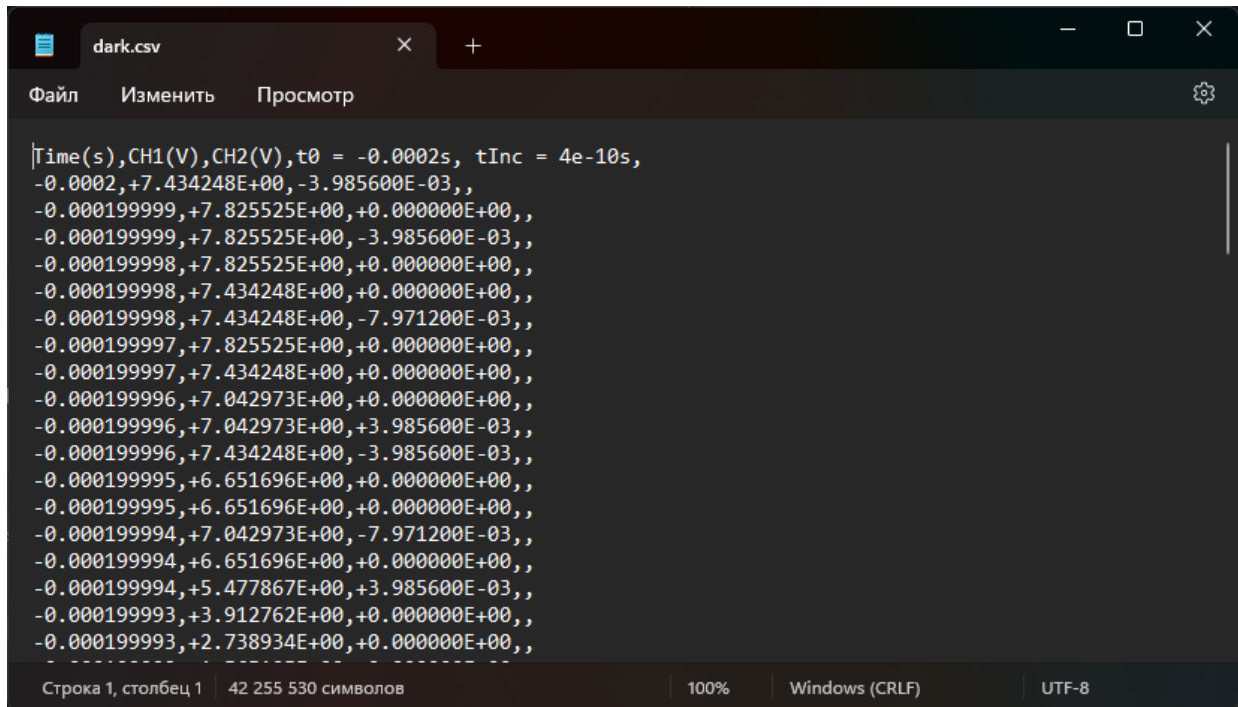
    for (int i = 0; i < m; ++i) {
        double time = i * dt;
        x.push_back(time);
        y.push_back(func(time));
    }
}
```

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.



```
|Time(s),CH1(V),CH2(V),t0 = -0.0002s, tInc = 4e-10s,  
-0.0002,+7.434248E+00,-3.985600E-03,,  
-0.000199999,+7.825525E+00,+0.000000E+00,,  
-0.000199999,+7.825525E+00,-3.985600E-03,,  
-0.000199998,+7.825525E+00,+0.000000E+00,,  
-0.000199998,+7.434248E+00,+0.000000E+00,,  
-0.000199998,+7.434248E+00,-7.971200E-03,,  
-0.000199997,+7.825525E+00,+0.000000E+00,,  
-0.000199997,+7.434248E+00,+0.000000E+00,,  
-0.000199996,+7.042973E+00,+0.000000E+00,,  
-0.000199996,+7.042973E+00,+3.985600E-03,,  
-0.000199996,+7.434248E+00,-3.985600E-03,,  
-0.000199995,+6.651696E+00,+0.000000E+00,,  
-0.000199995,+6.651696E+00,+0.000000E+00,,  
-0.000199994,+7.042973E+00,-7.971200E-03,,  
-0.000199994,+6.651696E+00,+0.000000E+00,,  
-0.000199994,+5.477867E+00,+3.985600E-03,,  
-0.000199993,+3.912762E+00,+0.000000E+00,,  
-0.000199993,+2.738934E+00,+0.000000E+00,,  
-----  
Строка 1, столбец 1 | 42 255 530 символов | 100% | Windows (CRLF) | UTF-8
```

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;
}

```

Листинг 2. Функция чтения данных из набора

Последовательные вычисления на процессоре

Показана реализация функции для вычисления ДПФ без параллельных стандартов, листинг 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).

$$(2) X_k = \sum_{n=0}^{N-1} u[n] \cdot e^{-\frac{i2\pi kn}{N}}$$

В формуле (2) $u[n]$ – значение входного сигнала из временного ряда осциллограммы, k – индекс вычисляемой частоты, $e^{-\frac{i2\pi kn}{N}}$ – комплексный коэффициент вращения (По формуле Эйлера $e^{-\frac{i2\pi kn}{N}} = \cos\left(-\frac{2\pi kn}{N}\right) + i\sin\left(-\frac{2\pi kn}{N}\right)$), i – мнимая единица. В цикле для каждого значения $u[n]$ вычислим угол фазу сигнала angle , как аргумент тригонометрических функций. Каждое $u[n]$ 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, Δt – 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).

$$(3) \Delta f = \frac{1}{N \cdot \Delta t}$$

```

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 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);
}
```

Листинг 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, листинг 5.

```
void my_compute_fft(const vector<double> &u, vector<double> &frequencies,
vector<double> &amplitudes, double time_step, int threads_omp, int gangs_acc) {
    int N = u.size();
    vector<complex<double>> fft_result(N);

    // Основная обработка с использованием OpenACC и OpenMP
    #pragma acc data copyin(u[0:N]) copyout(fft_result[0:N])
    omp_set_num_threads(threads_omp);
    #pragma omp parallel for
        for (int k = 0; k < N; ++k) {
            complex<double> sum(0.0, 0.0);

            // Параллельная обработка по элементам внутри OpenACC
            #pragma acc parallel loop num_gangs(gangs_acc) num_workers(32) vector_length(32)
            #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));
                }

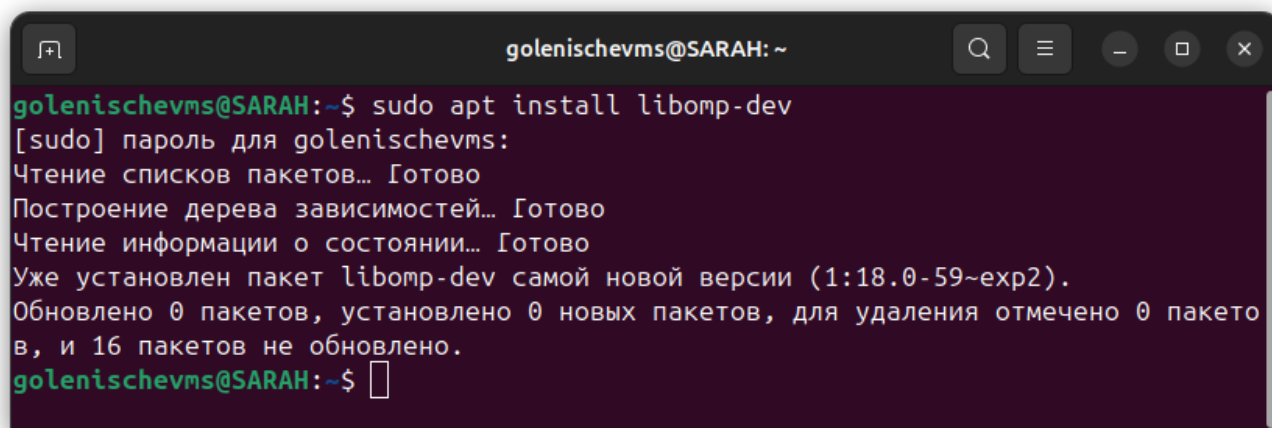
            fft_result[k] = sum;
        }
    // Инициализация векторов частот и амплитуд
    frequencies.resize(N);
    amplitudes.resize(N);
    // Расчет шага частоты
    double freq_step = 1.0 / (N * time_step);
    // Построение массивов частот и амплитуд
    for (int k = 0; k < N; ++k) {
        frequencies[k] = k * freq_step;
        amplitudes[k] = abs(fft_result[k]) / N;
    }
}
```

Листинг 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 потоках OpenMP распараллеливаем вычисления внутреннего цикла с помощью OpenACC. Таким образом – основные вычисления выборки мы выполняем на графическом ускорителе NVIDIA, а запись обработки 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.

A terminal window with a dark purple background and white text. The window title is 'golenischevms@SARAH: ~'. The terminal shows the command 'sudo apt install libomp-dev' being executed. The output includes prompts for the password, progress messages for reading package lists and building dependency trees, and a confirmation that the package is installed. The prompt returns to the user's shell.

```
golenischevms@SARAH:~$ sudo apt install libomp-dev
[sudo] пароль для golenischevms:
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Уже установлен пакет libomp-dev самой новой версии (1:18.0-59~exp2).
Обновлено 0 пакетов, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 16 пакетов не обновлено.
golenischevms@SARAH:~$
```

Рисунок 4. Установка OpenMP с помощью apt

```
golenischevms@SARAH: ~  
golenischevms@SARAH:~$ cat /proc/driver/nvidia/version  
NVRM version: NVIDIA UNIX x86_64 Kernel Module  535.183.01  Sun May 12 19:39:15 UTC 2024  
GCC version:  
golenischevms@SARAH:~$ sudo apt install nvidia-cuda-toolkit  
Чтение списков пакетов... Готово  
Построение дерева зависимостей... Готово  
Чтение информации о состоянии... Готово  
Уже установлен пакет nvidia-cuda-toolkit самой новой версии (12.0.140-1ubuntu4).  
Обновлено 0 пакетов, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 16  
пакетов не обновлено.  
golenischevms@SARAH:~$
```

Рисунок 5. Установка инструментов для вычислений на GPU

```
(base) golenischevms@SARAH:~$ sudo apt-get install libfftw3-dev  
[sudo] пароль для golenischevms:  
Чтение списков пакетов... Готово  
Построение дерева зависимостей... Готово  
Чтение информации о состоянии... Готово  
Уже установлен пакет libfftw3-dev самой новой версии (3.3.10-1ubuntu3).  
Обновлено 0 пакетов, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 100 пакетов не обновлено.  
(base) golenischevms@SARAH:~$
```

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, листинг 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

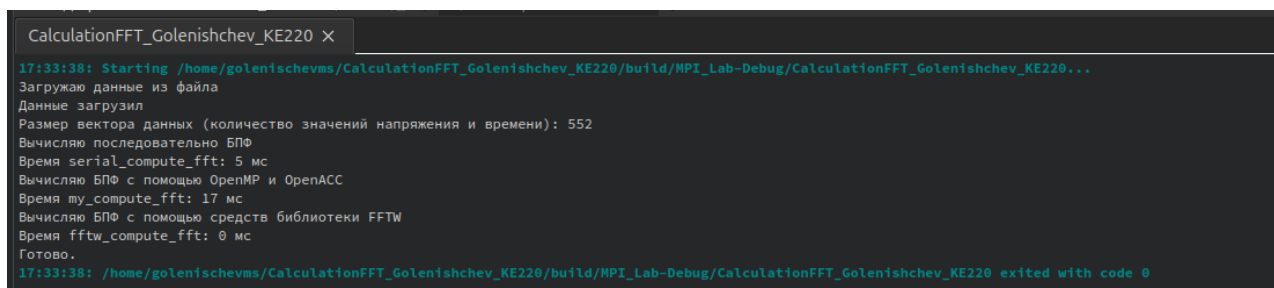
```

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.



```

CalculationFFT_Golenishchev_KE220 X
17:33:38: Starting /home/golenishchevms/CalculationFFT_Golenishchev_KE220/build/MPI_Lab-Debug/CalculationFFT_Golenishchev_KE220...
Загружаю данные из файла
Данные загрузил
Размер вектора данных (количество значений напряжения и времени): 552
Вычисляю последовательно БПФ
Время serial_compute_fft: 5 мс
Вычисляю БПФ с помощью OpenMP и OpenACC
Время mp_compute_fft: 17 мс
Вычисляю БПФ с помощью средств библиотеки FFTW
Время fftw_compute_fft: 0 мс
Готово.
17:33:38: /home/golenishchevms/CalculationFFT_Golenishchev_KE220/build/MPI_Lab-Debug/CalculationFFT_Golenishchev_KE220 exited with code 0

```

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

Проведены несколько тестов с увеличением количества потоков, gangs, а также тестирование с различным размеров выборки, Рисунок 8.

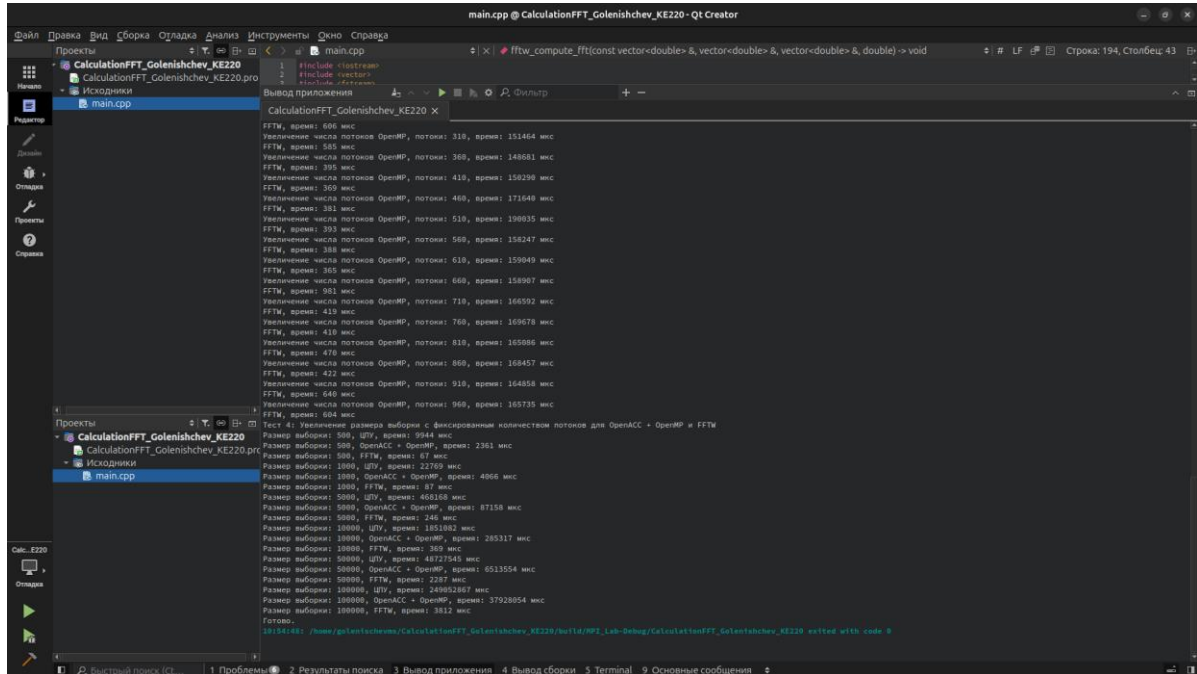


Рисунок 8. Результаты тестовых запусков

Описание тестов:

1. Количество точек: 10 000, параметры OpenACC num_workers(32) vector_length(32), увеличиваем num_gangs и omp_set_num_threads от 10 до 1000 с шагом 50. Представлены результаты теста, Таблица 1.
2. Количество точек: 10 000, параметры OpenACC num_workers(32) vector_length(32), увеличиваем num_gangs от 10 до 1000 с шагом 50, omp_set_num_threads(10). Представлены результаты теста, Таблица 2.
3. Количество точек: 10 000, параметры OpenACC num_workers(32) vector_length(32), увеличиваем omp_set_num_threads от 10 до 1000 с шагом 50, num_gangs(10). Представлены результаты теста, Таблица 3.
4. Количество точек: 500, 1000, 5000, 10000, 50000, 100000, параметры OpenACC num_workers(32) vector_length(32), num_gangs(10) и omp_set_num_threads(10). Представлены результаты теста, Таблица 4.

Приведены результаты вычислений, Приложение А.

```
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, 10000); // Используем 10 000 значений  
    cout << "Данные загружены" << endl;  
  
    double time_step = 4e-10; // Шаг времени  
    //  
    cout << "-----" << endl;  
    cout << "Тестируется собственная реализация ДПФ и БПФ в FFTW" << endl;  
    cout << "-----" << endl;  
    // Проведение тестов  
    test_openacc_omp_fftw_fixed_threads(u, time_step);  
    test_openacc_omp_increase_openacc_threads(u, time_step);  
    test_openacc_omp_increase_omp_threads(u, time_step);  
    test_increase_sample_size(u, time_step);  
  
    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.

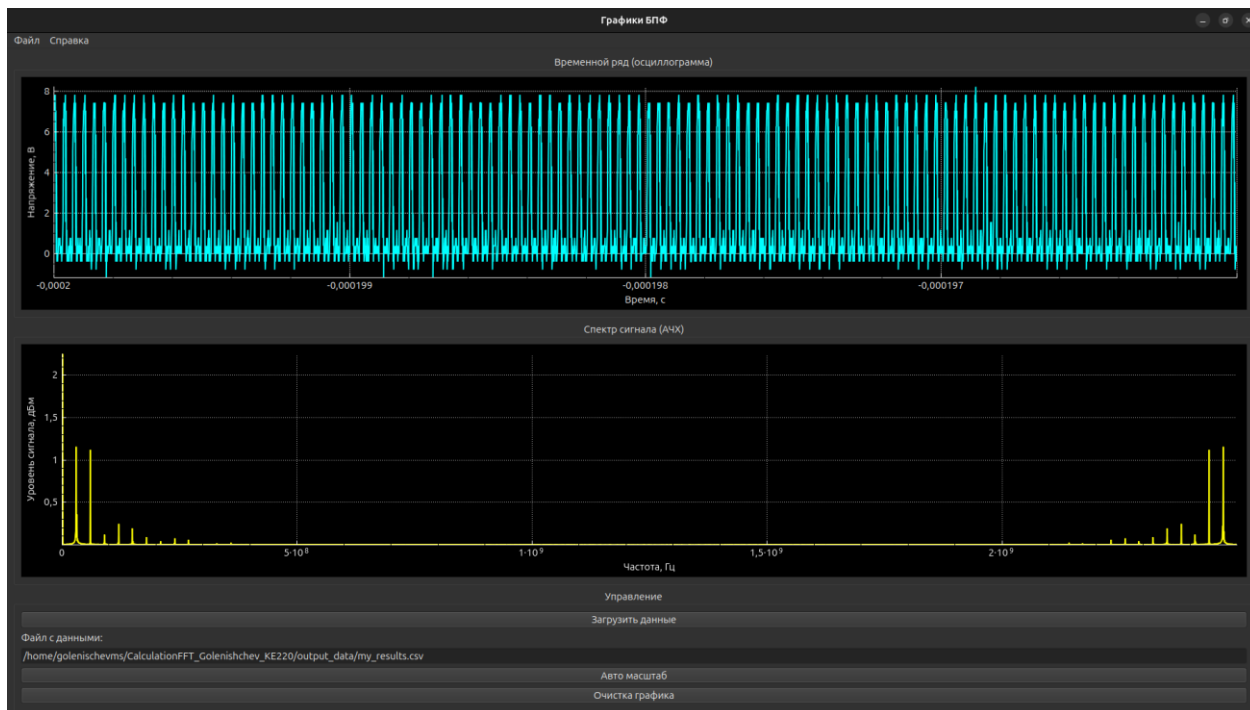


Рисунок 9. Визуализация результатов обработки OpenACC + OpenMP

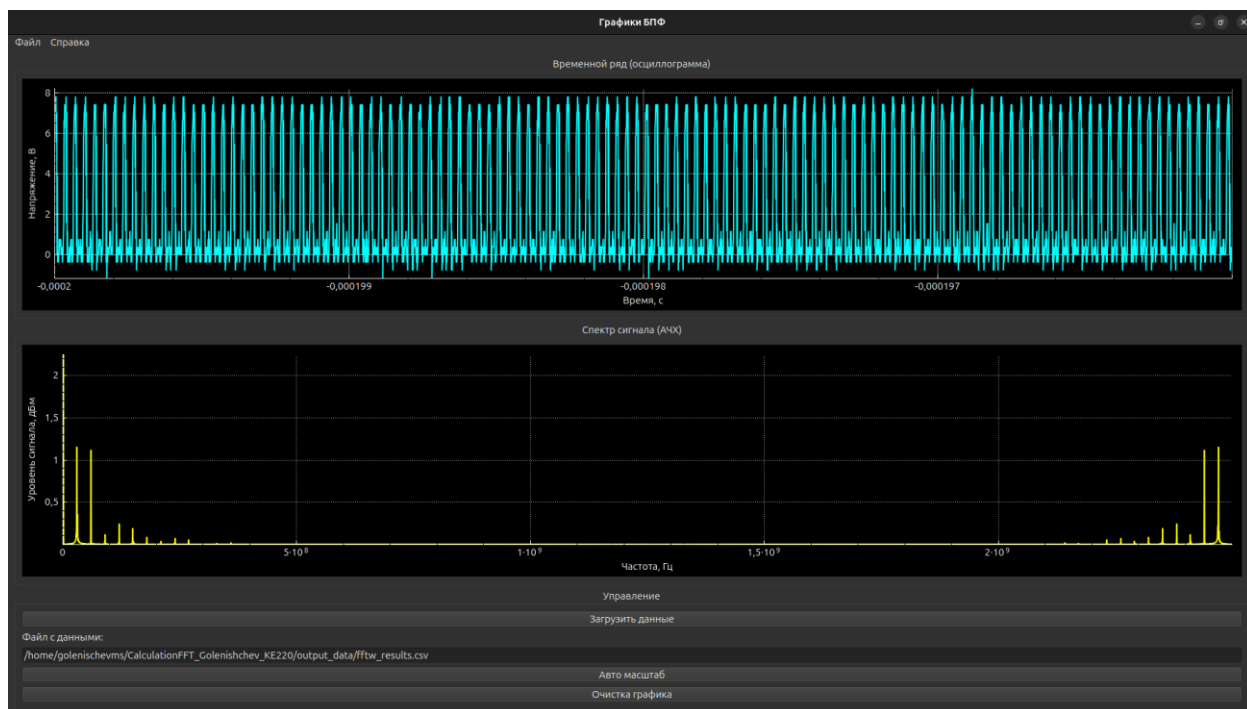


Рисунок 10. Визуализация результатов обработки FFTW

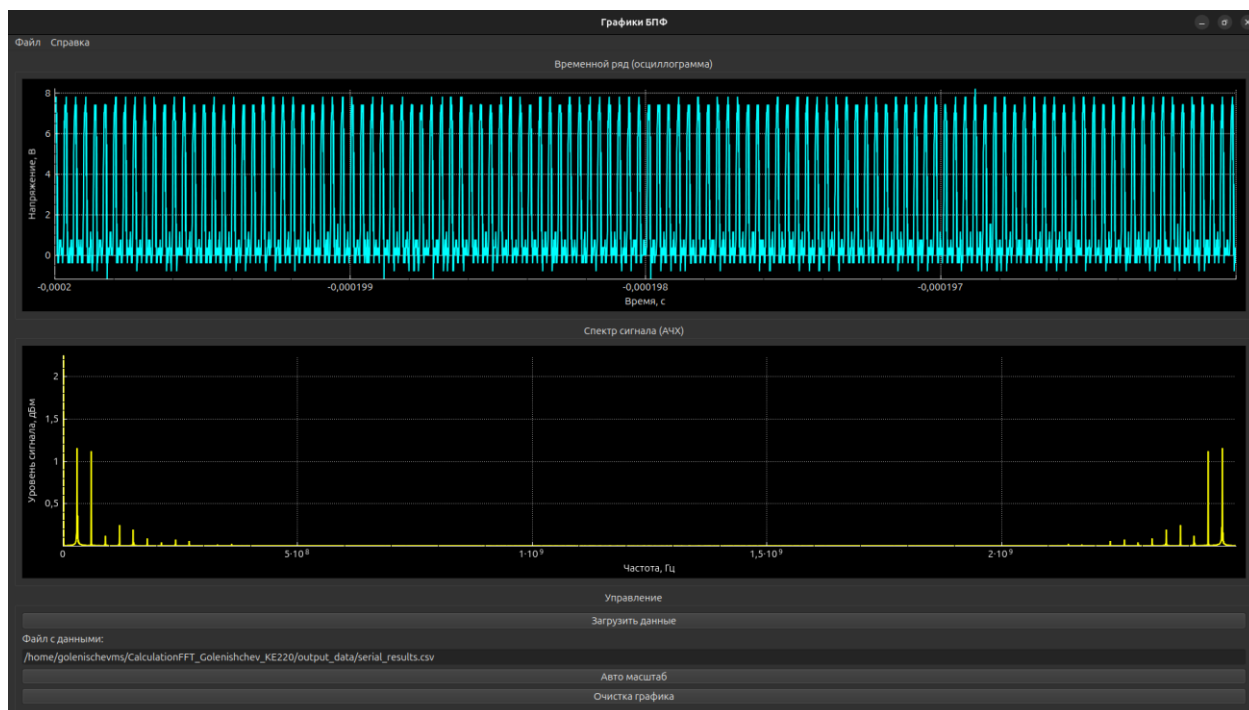


Рисунок 11. Визуализация результатов обработки последовательных вычислений

Построенные графики позволяют оценить результаты работы алгоритмов.

Представлены характеристики компьютера, на котором проводились расчеты,

Рисунок 12. 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 производительностью 22 TFLOPS, ядра тарсировки лучей 3th Gen производительностью 51 TFLOPS, тензорные ядра 4th Gen производительностью 353 AI TOPS, 16 Гб GDDR6 видеопамяти).

```
golenischevms@SARAH:~$ neofetch

      .-/+oossssoo+/-.
      `:+ssssssssssssssss+:`
      -+ssssssssssssssssyyssss+-
      .ossssssssssssssssdMMMMyssso.
      /ssssssssssshdmmNNmmyNMMMMhssssss/
      +ssssssssshmydMMMMMMMMNdddyssssssss+
      /ssssssssshNMMMyhhyyyyhmNMMMMhssssssss/
      .sssssssssdMMMNhssssssssshNMMMdssssssss.
      +ssssshhhyNMMNysssssssssssyNMMMyssssssss+
      ossyNMMMNyMMhssssssssssshmmhssssssso
      ossyNMMMNyMMhssssssssssshmmhssssssso
      +ssssshhhyNMMNysssssssssssyNMMMyssssssss+
      .sssssssssdMMMNhssssssssshNMMMdssssssss.
      /ssssssssshNMMMyhhyyyyhdNMMMMhssssssss/
      +sssssssssdmydMMMMMMMMNdddyssssssss+
      /ssssssssshdmmNNNmyNMMMMhssssss/
      .ossssssssssssssssdMMMMyssso.
      -+ssssssssssssssssyyssss+-
      `:+ssssssssssssssss+:`
      .-/+oossssoo+/-.

golenischevms@SARAH
-----
OS: Ubuntu 24.04.1 LTS x86_64
Host: MS-7D32 3.0
Kernel: 6.8.0-49-generic
Uptime: 3 hours, 47 mins
Packages: 2310 (dpkg), 26 (flatpak),
Shell: bash 5.2.21
Resolution: 1920x1080
DE: GNOME 46.0
WM: Mutter
WM Theme: Adwaita
Theme: Yaru-blue-dark [GTK2/3]
Icons: Yaru-blue [GTK2/3]
Terminal: gnome-terminal
CPU: 13th Gen Intel i7-13700K (24) @
GPU: Intel Raptor Lake-S GT1 [UHD Gr
GPU: NVIDIA GeForce RTX 4060 Ti 16GB
Memory: 4968MiB / 64079MiB

golenischevms@SARAH:~$
```

Рисунок 12. Характеристики ПК для тестирования алгоритмов.

Расчет эффективности алгоритмов

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

$$(4) S_p(n) = \frac{T_{seq}(n)}{T_p(n)}$$

Проведем расчет стоимости по формуле (5), $C_p(n)$ – стоимость параллельного алгоритма, T_p – время выполнения параллельного алгоритма для n выборок, p – число потоков/процессов (расчет проведем для 8 процессов).

$$(5) C_p(n) = T_p \cdot p$$

Расчеты проведем с помощью программы на языке Python, Приложение А.

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.

Приложение А

Визуализация данных и построение таблиц к результатам тестов Итоговое задание, Голенищев А. Б., КЭ-220

Исходные данные из коснсольного вывода

10:48:50: Starting /home/golenishevms/CalculationFFT_Golenishchev_KE220/build/MPI_Lab-Debug/CalculationFFT_Golenishchev_KE220... Загружаю данные из файла Данные загружены -----
Тестируется собственная реализация ДПФ и БПФ в FFTW ----- Тест 1: Увеличение числа потоков для OpenACC + OpenMP и FFTW с фиксированным размером выборки OpenACC + OpenMP, потоки: 10, время: 303804 мкс FFTW, потоки: 10, время: 743 мкс OpenACC + OpenMP, потоки: 60, время: 162738 мкс FFTW, потоки: 60, время: 493 мкс OpenACC + OpenMP, потоки: 110, время: 150007 мкс FFTW, потоки: 110, время: 400 мкс OpenACC + OpenMP, потоки: 160, время: 149461 мкс FFTW, потоки: 160, время: 494 мкс OpenACC + OpenMP, потоки: 210, время: 148981 мкс FFTW, потоки: 210, время: 387 мкс OpenACC + OpenMP, потоки: 260, время: 147864 мкс FFTW, потоки: 260, время: 379 мкс OpenACC + OpenMP, потоки: 310, время: 148581 мкс FFTW, потоки: 310, время: 908 мкс OpenACC + OpenMP, потоки: 360, время: 160341 мкс FFTW, потоки: 360, время: 434 мкс OpenACC + OpenMP, потоки: 410, время: 160634 мкс FFTW, потоки: 410, время: 386 мкс OpenACC + OpenMP, потоки: 460, время: 146756 мкс FFTW, потоки: 460, время: 382 мкс OpenACC + OpenMP, потоки: 510, время: 148308 мкс FFTW, потоки: 510, время: 387 мкс OpenACC + OpenMP, потоки: 560, время: 149473 мкс FFTW, потоки: 560, время: 366 мкс OpenACC + OpenMP, потоки: 610, время: 149331 мкс FFTW, потоки: 610, время: 591 мкс OpenACC + OpenMP, потоки: 660, время: 149823 мкс FFTW, потоки: 660, время: 984 мкс OpenACC + OpenMP, потоки: 710, время: 151405 мкс FFTW, потоки: 710, время: 420 мкс OpenACC + OpenMP, потоки: 760, время: 152542 мкс FFTW, потоки: 760, время: 383 мкс OpenACC + OpenMP, потоки: 810, время: 153801 мкс FFTW, потоки: 810, время: 375 мкс OpenACC + OpenMP, потоки: 860, время: 152010 мкс FFTW, потоки: 860, время: 480 мкс OpenACC + OpenMP, потоки: 910, время: 150808 мкс FFTW, потоки: 910, время: 382 мкс OpenACC + OpenMP, потоки: 960, время: 150816 мкс FFTW, потоки: 960, время: 445 мкс Тест 2: Увеличение числа потоков для OpenACC, фиксированное количество потоков для OpenMP и FFTW Увеличение gsngs OpenACC, gangs: 10, время: 291724 мкс FFTW, время: 571 мкс Увеличение gsngs OpenACC, gangs: 60, время: 289334 мкс FFTW, время: 411 мкс Увеличение gsngs OpenACC, gangs: 110, время: 295173 мкс FFTW, время: 390 мкс Увеличение gsngs OpenACC, gangs: 160, время: 286332 мкс FFTW, время: 508 мкс Увеличение gsngs OpenACC, gangs: 210, время: 305266 мкс FFTW, время: 568 мкс Увеличение gsngs OpenACC, gangs: 260, время: 295021 мкс FFTW, время: 359 мкс Увеличение gsngs OpenACC, gangs: 310, время: 293375 мкс FFTW, время: 708 мкс Увеличение gsngs OpenACC, gangs: 360, время: 298900 мкс FFTW, время: 393 мкс Увеличение gsngs OpenACC, gangs: 410, время: 306337 мкс FFTW, время: 413 мкс Увеличение gsngs OpenACC, gangs: 460, время: 322617 мкс FFTW, время: 405 мкс Увеличение gsngs OpenACC, gangs: 510, время: 301967 мкс FFTW, время: 375 мкс Увеличение gsngs OpenACC, gangs: 560, время: 286345 мкс FFTW, время: 429 мкс Увеличение gsngs OpenACC, gangs: 610, время: 302766 мкс FFTW, время: 415 мкс Увеличение gsngs OpenACC, gangs: 660, время: 307046 мкс FFTW, время: 737 мкс Увеличение gsngs OpenACC, gangs: 710, время: 293869 мкс FFTW, время: 392 мкс Увеличение gsngs OpenACC, gangs: 760, время: 301462 мкс FFTW, время: 359 мкс Увеличение gsngs OpenACC, gangs: 810, время: 297009 мкс FFTW, время: 371 мкс Увеличение gsngs OpenACC, gangs: 860, время: 297531 мкс FFTW, время: 391 мкс Увеличение gsngs OpenACC, gangs: 910, время: 289520 мкс FFTW, время: 406 мкс Увеличение gsngs OpenACC, gangs: 960, время: 288777 мкс FFTW, время: 398 мкс Тест 3: Увеличение числа потоков для OpenMP, фиксированное количество потоков для OpenACC и FFTW Увеличение числа потоков OpenMP, потоки: 10, время: 288925 мкс FFTW, время: 441 мкс Увеличение числа потоков OpenMP, потоки: 60, время: 168474 мкс FFTW, время: 405 мкс Увеличение числа потоков OpenMP, потоки: 110, время: 151735 мкс FFTW, время: 433 мкс Увеличение числа потоков OpenMP, потоки: 160, время: 158562 мкс FFTW, время: 688 мкс Увеличение числа потоков OpenMP, потоки: 210, время: 151278 мкс FFTW, время: 400 мкс Увеличение числа потоков OpenMP, потоки: 260, время: 151664 мкс FFTW, время: 606 мкс Увеличение числа потоков OpenMP, потоки: 310, время: 151464 мкс FFTW, время: 585 мкс Увеличение числа потоков OpenMP, потоки: 360, время: 148681 мкс FFTW, время: 395 мкс Увеличение числа потоков OpenMP, потоки: 410, время: 150290 мкс FFTW, время: 369 мкс Увеличение числа потоков OpenMP, потоки: 460, время: 171640 мкс FFTW, время: 381 мкс Увеличение числа потоков OpenMP, потоки: 510, время: 190035 мкс FFTW, время: 393 мкс Увеличение числа потоков OpenMP, потоки: 560, время: 158247 мкс FFTW, время: 388 мкс Увеличение числа потоков OpenMP, потоки: 610, время: 159049 мкс FFTW, время: 365 мкс Увеличение числа потоков OpenMP, потоки: 660, время: 158907 мкс FFTW, время: 981 мкс Увеличение числа потоков OpenMP, потоки: 710, время: 166592 мкс FFTW, время: 419 мкс Увеличение числа потоков OpenMP, потоки: 760, время: 169678 мкс FFTW, время: 410 мкс Увеличение числа потоков OpenMP, потоки: 810, время: 165086 мкс FFTW, время: 470 мкс Увеличение числа потоков OpenMP, потоки: 860, время: 168457 мкс FFTW, время: 422 мкс Увеличение числа потоков OpenMP, потоки: 910, время: 164858 мкс FFTW, время: 640 мкс Увеличение числа потоков OpenMP, потоки: 960, время: 165735 мкс FFTW, время: 604 мкс Тест 4: Увеличение размера выборки с фиксированным количеством потоков для OpenACC + OpenMP и FFTW Размер выборки: 500, ЦПУ, время: 9944 мкс Размер выборки: 500, OpenACC + OpenMP, время: 2361 мкс Размер выборки: 500, FFTW, время: 67 мкс Размер выборки: 1000, ЦПУ, время: 22769 мкс Размер выборки: 1000, OpenACC + OpenMP, время: 4066 мкс Размер выборки: 1000, FFTW, время: 87 мкс Размер выборки: 5000, ЦПУ, время: 468168 мкс Размер выборки: 5000, OpenACC + OpenMP, время: 87158 мкс Размер выборки: 5000, FFTW, время: 246 мкс Размер выборки: 10000, ЦПУ, время: 1851082 мкс Размер выборки: 10000, OpenACC + OpenMP, время: 285317 мкс Размер выборки: 10000, FFTW, время: 369 мкс Размер выборки: 50000, ЦПУ, время: 48727545 мкс Размер выборки: 50000, OpenACC + OpenMP, время: 6513554 мкс Размер выборки: 50000, FFTW, время: 2287 мкс Размер выборки: 100000, ЦПУ, время: 249052867 мкс Размер выборки: 100000, OpenACC + OpenMP, время: 37928054 мкс Размер выборки: 100000, FFTW, время: 3812 мкс Готово. 10:54:48: /home/golenishevms/CalculationFFT_Golenishchev_KE220/build/MPI_Lab-Debug/CalculationFFT_Golenishchev_KE220 exited with code 0

Визуализация данных (таблицы)

In [4]:
from prettytable **import** PrettyTable

Таблица 1: Увеличение числа потоков для OpenACC + OpenMP и FFTW

```
header_1 = ["Потоки", "OpenACC + OpenMP, время (мкс)", "FFTW, время (мкс)"]
```

```
data_1 = [  
    (10, 303804, 743),  
    (60, 162738, 493),  
    (110, 150007, 400),  
    (160, 149461, 494),  
    (210, 148981, 387),  
    (260, 147864, 379),  
    (310, 148581, 908),  
    (360, 160341, 434),  
    (410, 160634, 386),  
    (460, 146756, 382),  
    (510, 148308, 387),  
    (560, 149473, 366),  
    (610, 149331, 591),  
    (660, 149823, 984),  
    (710, 151405, 420),  
    (760, 152542, 383),  
    (810, 153801, 375),  
    (860, 152010, 480),  
    (910, 150808, 382),  
    (960, 150816, 445),  
]
```

```
# Таблица 2: Увеличение gsngs OpenACC и FFTW
```

```
header_2 = ["Gangs", "OpenACC, время (мкс)", "FFTW, время (мкс)"]
```

```
data_2 = [  
    (10, 291724, 571),  
    (60, 289334, 411),  
    (110, 295173, 390),  
    (160, 286332, 508),  
    (210, 305266, 568),  
    (260, 295021, 359),  
    (310, 293375, 708),  
    (360, 298900, 393),  
    (410, 306337, 413),  
    (460, 322617, 405),  
    (510, 301967, 375),  
    (560, 286345, 429),  
    (610, 302766, 415),  
    (660, 307046, 737),  
    (710, 293869, 392),  
    (760, 301462, 359),  
    (810, 297009, 371),  
    (860, 297531, 391),  
    (910, 289520, 406),  
    (960, 288777, 398),  
]
```

```
# Таблица 3: Увеличение числа потоков для OpenMP и FFTW
```

```
header_3 = ["Потоки", "OpenMP, время (мкс)", "FFTW, время (мкс)"]
```

```
data_3 = [  
    (10, 288925, 441),  
    (60, 168474, 405),  
    (110, 151735, 433),  
    (160, 158562, 688),  
    (210, 151278, 400),  
    (260, 151664, 606),  
    (310, 151464, 585),  
    (360, 148681, 395),  
    (410, 150290, 369),  
    (460, 171640, 381),  
    (510, 190035, 393),  
    (560, 158247, 388),  
    (610, 159049, 365),  
    (660, 158907, 981),  
    (710, 166592, 419),  
    (760, 169678, 410),  
    (810, 165086, 470),  
    (860, 168457, 422),  
    (910, 164858, 640),  
    (960, 165735, 604),  
]
```



```
# Таблица 4: Увеличение размера выборки
header_4 = ["Размер выборки", "ЦПУ, время (мкс)", "OpenACC + OpenMP, время (мкс)", "FFTW, время (мкс)"]
data_4 = [
    (500, 9944, 2361, 67),
    (1000, 22769, 3891, 158),
    (1500, 36962, 5420, 280),
    (2000, 50927, 6983, 377),
]
```

```
def create_pretty_table(header, data):
    table = PrettyTable()
    table.field_names = header
    for row in data:
        table.add_row(row)
    return table
```

```
# Создание и вывод таблиц
tables = [
    create_pretty_table(header_1, data_1),
    create_pretty_table(header_2, data_2),
    create_pretty_table(header_3, data_3),
    create_pretty_table(header_4, data_4),
]
```

```
for i, table in enumerate(tables, start=1):
    print(f"Таблица {i}:\n{table}\n")
```

Таблица 1:

+-----+			
Потоки OpenACC + OpenMP, время (мкс) FFTW, время (мкс)			
+-----+			
10	303804	743	
60	162738	493	
110	150007	400	
160	149461	494	
210	148981	387	
260	147864	379	
310	148581	908	
360	160341	434	
410	160634	386	
460	146756	382	
510	148308	387	
560	149473	366	
610	149331	591	
660	149823	984	
710	151405	420	
760	152542	383	
810	153801	375	
860	152010	480	
910	150808	382	
960	150816	445	
+-----+			

Таблица 2:

+-----+			
Gangs OpenACC, время (мкс) FFTW, время (мкс)			
+-----+			
10	291724	571	
60	289334	411	
110	295173	390	
160	286332	508	
210	305266	568	
260	295021	359	
310	293375	708	
360	298900	393	
410	306337	413	
460	322617	405	
510	301967	375	
560	286345	429	
610	302766	415	
660	307046	737	
710	293869	392	
760	301462	359	
810	297009	371	
860	297531	391	
910	289520	406	
960	288777	398	
+-----+			

Таблица 3:

+-----+			
Потоки OpenMP, время (мкс) FFTW, время (мкс)			

10	288925	441
60	168474	405
110	151735	433
160	158562	688
210	151278	400
260	151664	606
310	151464	585
360	148681	395
410	150290	369
460	171640	381
510	190035	393
560	158247	388
610	159049	365
660	158907	981
710	166592	419
760	169678	410
810	165086	470
860	168457	422
910	164858	640
960	165735	604

Таблица 4:

Размер выборки ЦПУ, время (мкс) OpenACC + OpenMP, время (мкс) FFTW, время (мкс)				
500	9944	2361	67	
1000	22769	3891	158	
1500	36962	5420	280	
2000	50927	6983	377	

Визуализация данных (графики)

In [6]:

```
import matplotlib.pyplot as plt
```

```
# Функция для построения графиков
```

```
def plot_graph(x_data, y_data, labels, x_label, y_label, title):  
    plt.figure(figsize=(10, 6))  
    for i, y in enumerate(y_data):  
        plt.plot(x_data, y, marker='o', label=labels[i]) # добавляем метки для каждой линии  
    plt.xlabel(x_label)  
    plt.ylabel(y_label)  
    plt.title(title)  
    plt.legend()  
    plt.grid(True)  
    plt.show()
```

```
# Таблица 1: Увеличение числа потоков для OpenACC + OpenMP и FFTW
```

```
x_data_1 = [row[0] for row in data_1]  
y_data_1 = [  
    [row[1] for row in data_1], # OpenACC + OpenMP, время (мкс)  
    [row[2] for row in data_1] # FFTW, время (мкс)  
]  
labels_1 = ["OpenACC + OpenMP", "FFTW"]
```

```
plot_graph(x_data_1, y_data_1, labels_1, "Потоки", "Время (мкс)", "Увеличение числа потоков для OpenACC + OpenMP и FFTW")
```

```
# Таблица 2: Увеличение gangs OpenACC и FFTW
```

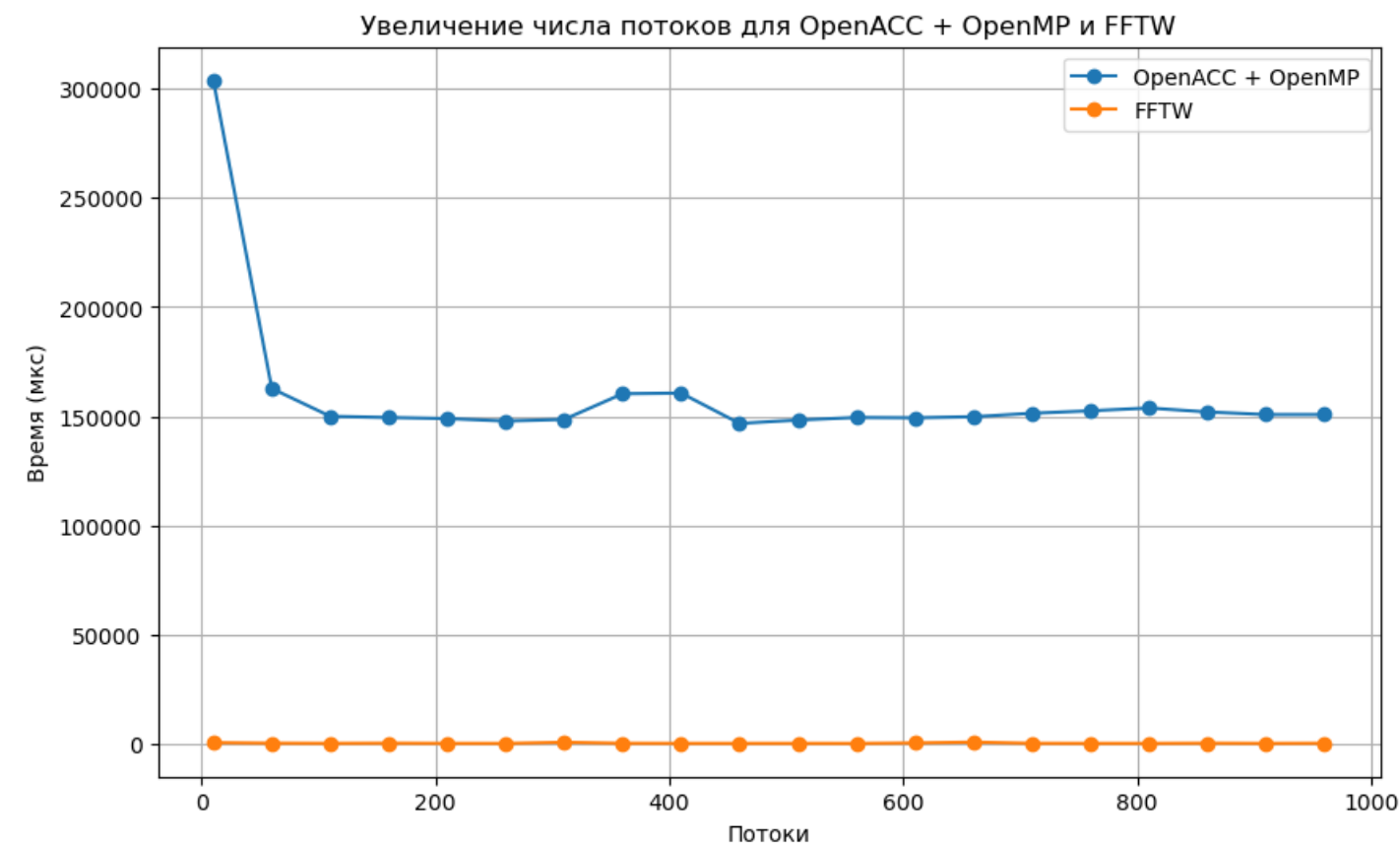
```
x_data_2 = [row[0] for row in data_2]  
y_data_2 = [  
    [row[1] for row in data_2], # OpenACC, время (мкс)  
    [row[2] for row in data_2] # FFTW, время (мкс)  
]  
labels_2 = ["OpenACC", "FFTW"]
```

```
plot_graph(x_data_2, y_data_2, labels_2, "Gangs", "Время (мкс)", "Увеличение gangs OpenACC и FFTW")
```

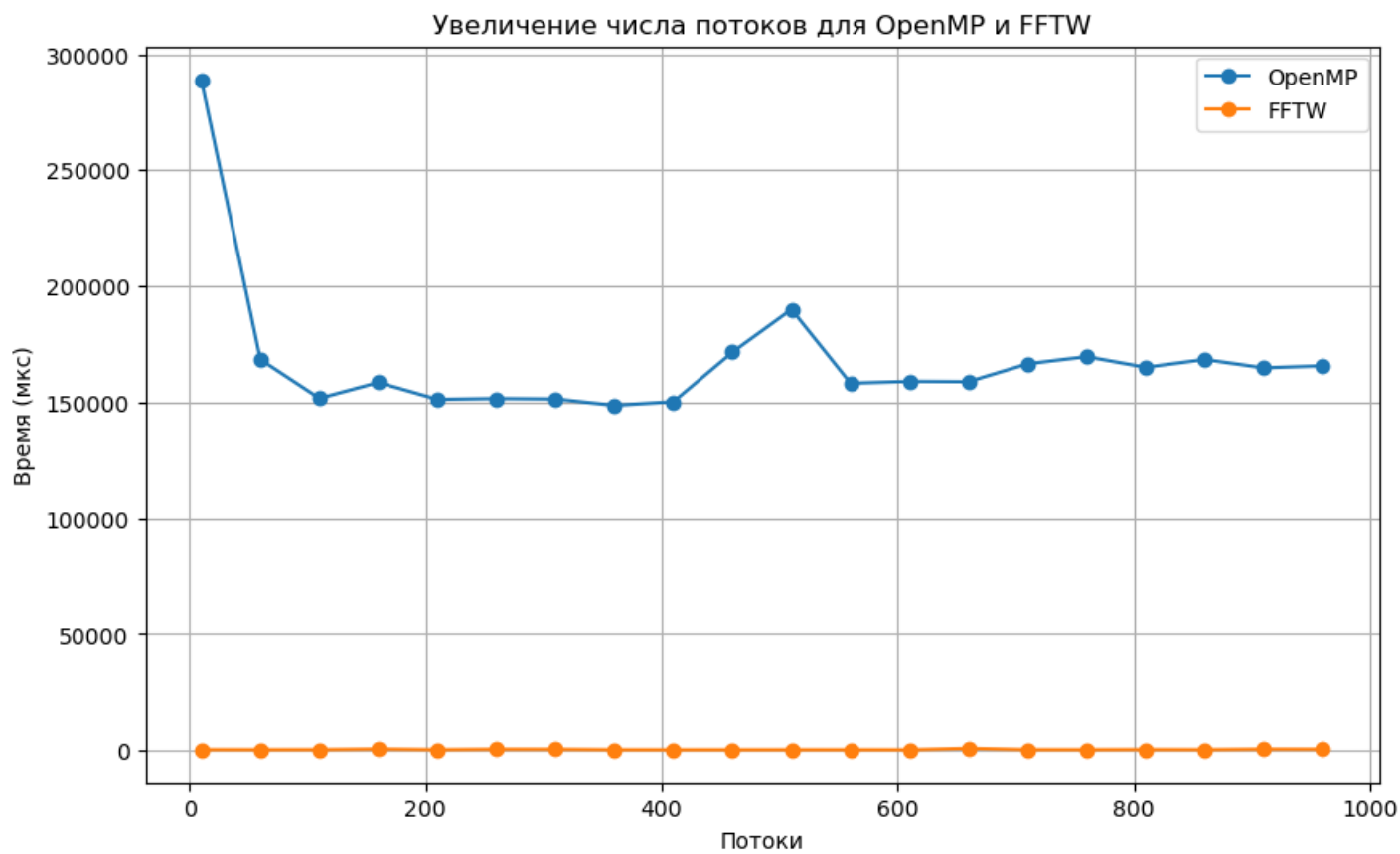
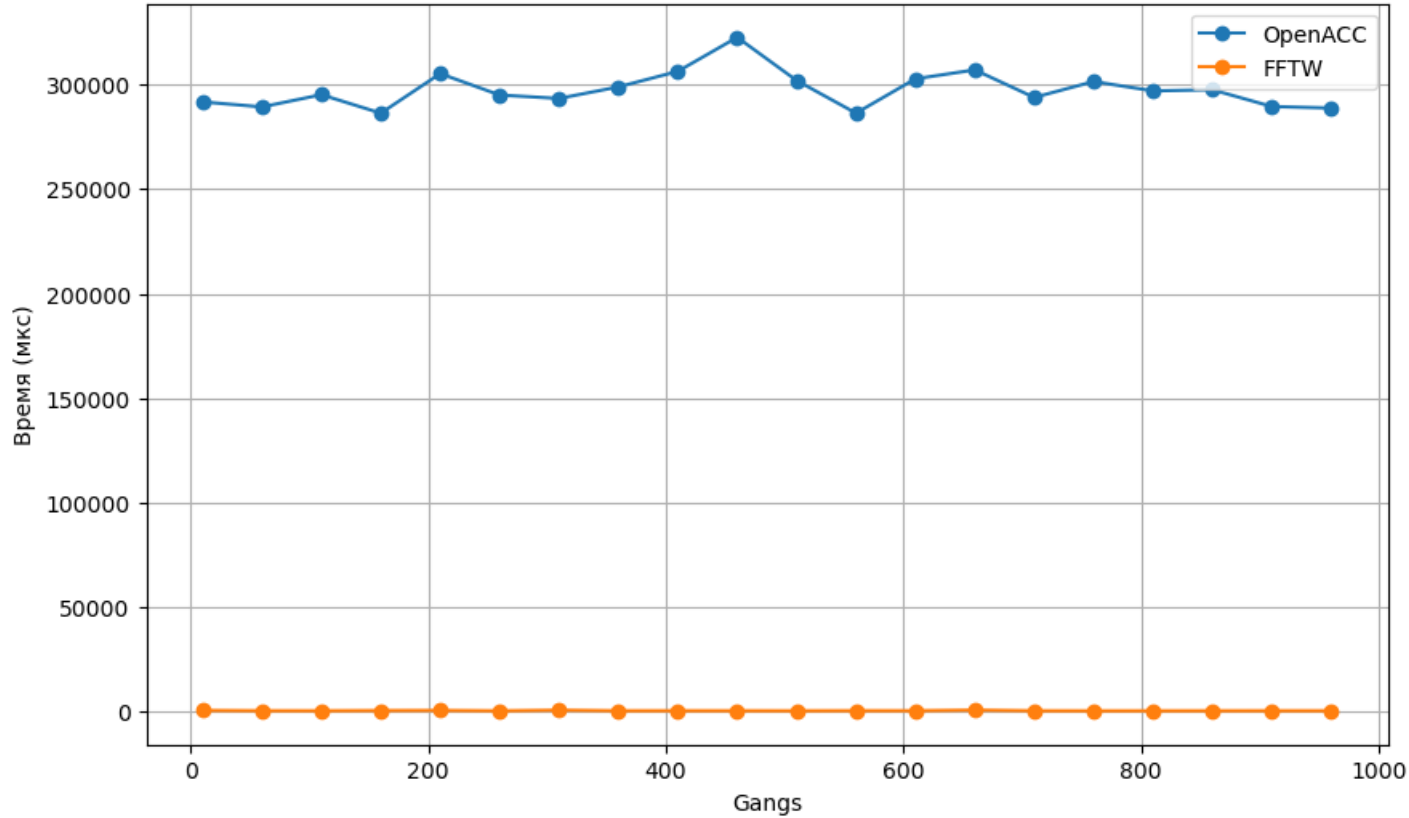
```
# Таблица 3: Увеличение числа потоков для OpenMP и FFTW
```

```
x_data_3 = [row[0] for row in data_3]  
y_data_3 = [  
    [row[1] for row in data_3], # OpenMP, время (мкс)  
    [row[2] for row in data_3] # FFTW, время (мкс)  
]  
labels_3 = ["OpenMP", "FFTW"]
```

```
plot_graph(x_data_3, y_data_3, labels_3, "Потоки", "Время (мкс)", "Увеличение числа потоков для OpenMP и FFTW")
```



Увеличение gangs OpenACC и FFTW



In [7]:

```
import matplotlib.pyplot as plt
```

```
# Данные теста 4
```

```
sample_sizes = [500, 1000, 5000, 10000, 50000, 100000]
```

```
cpu_times = [9944, 22769, 468168, 1851082, 48727545, 249052867]
```

```
openacc_openmp_times = [2361, 4066, 87158, 285317, 6513554, 37928054]
```

```
fftw_times = [67, 87, 246, 369, 2287, 3812]
```

```
plt.figure(figsize=(10, 6))
```

```
plt.plot(sample_sizes, cpu_times, label="CPU", marker="o")
```

```
plt.plot(sample_sizes, openacc_openmp_times, label="OpenACC + OpenMP", marker="s")
```

```
plt.plot(sample_sizes, fftw_times, label="FFTW", marker="^")
```

```
plt.xlabel("Количество точек", fontsize=12)
```

```
plt.ylabel("Время выполнения (мкс)", fontsize=12)
```

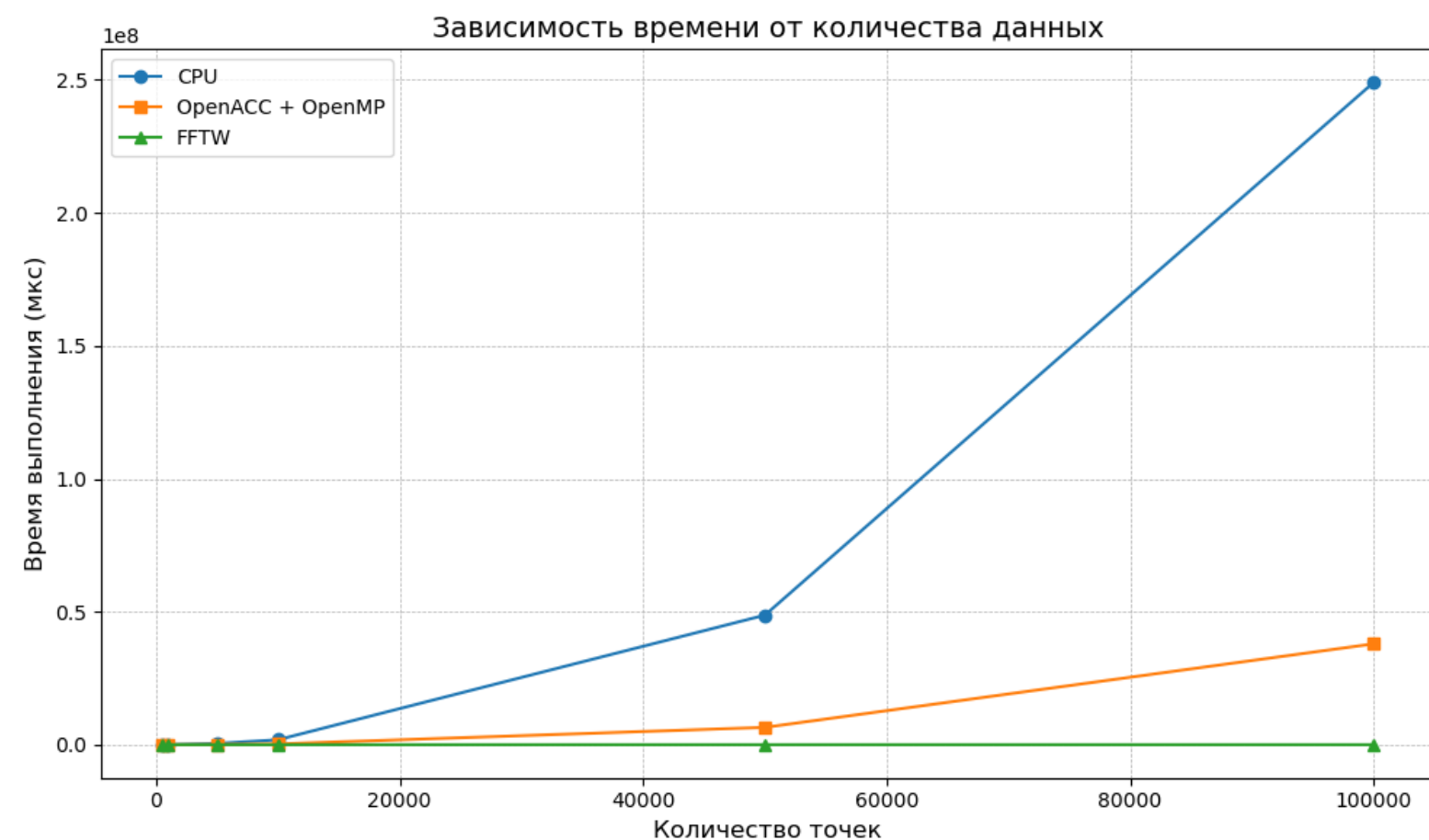
```
plt.title("Зависимость времени от количества данных", fontsize=14)
```

```
plt.legend()
```

```
plt.grid(visible=True, which="both", linestyle="--", linewidth=0.5)
```

```
plt.tight_layout()
```

```
plt.show()
```



Оценка эффективности параллельного алгоритма

```
In [9]:
```

```
import matplotlib.pyplot as plt
```

```
from prettytable import PrettyTable
```

```
# Время последовательного и параллельного алгоритмов для разных потоков
```

```
data = [
```

```
    [10, 303804, 743],
```

```
    [60, 162738, 493],
```

```
    [110, 150007, 400],
```

```
    [160, 149461, 494],
```

```
    [210, 148981, 387],
```

```
    [260, 147864, 379],
```

```
    [310, 148581, 908],
```

```
    [360, 160341, 434],
```

```
    [410, 160634, 386],
```

```
    [460, 146756, 382],
```

```
    [510, 148308, 387],
```

```
    [560, 149473, 366],
```

```

[610, 149331, 591],
[660, 149823, 984],
[710, 151405, 420],
[760, 152542, 383],
[810, 153801, 375],
[860, 152010, 480],
[910, 150808, 382],
[960, 150816, 445]
]

# Число потоков для расчета стоимости
p = 8

# Создаем таблицу
table = PrettyTable()
table.field_names = ["Потоки", "Время OpenACC + OpenMP (мкс)", "Время FFTW (мкс)", "Ускорение S_p(n)", "Стоимость C_p(n)"]

# Списки для данных графиков
threads = [] # Список для потоков
s_p_openacc_values = [] # Ускорение для OpenACC+OpenMP
s_p_fftw_values = [] # Ускорение для FFTW
c_p_openacc_values = [] # Стоимость для OpenACC+OpenMP
c_p_fftw_values = [] # Стоимость для FFTW

# Проходим по данным и рассчитываем ускорение и стоимость
for row in data:
    thread_count, t_openacc, t_fftw = row

    # Рассчитываем ускорение
    s_p_openacc = t_openacc / t_fftw
    s_p_fftw = t_fftw / t_fftw # Ускорение для FFTW всегда равно 1

    # Рассчитываем стоимость
    c_p_openacc = t_openacc * p
    c_p_fftw = t_fftw * p

    # Добавляем данные в таблицу и списки
    table.add_row([thread_count, t_openacc, t_fftw, f"{s_p_openacc:.2f}", f"{c_p_openacc}"])

    threads.append(thread_count) # Добавляем количество потоков
    s_p_openacc_values.append(s_p_openacc) # Ускорение OpenACC+OpenMP
    s_p_fftw_values.append(s_p_fftw) # Ускорение FFTW
    c_p_openacc_values.append(c_p_openacc) # Стоимость OpenACC+OpenMP
    c_p_fftw_values.append(c_p_fftw) # Стоимость FFTW

# Выводим таблицу
print(table)

# Построение графиков
plt.figure(figsize=(12, 6))

# График ускорения
plt.subplot(1, 2, 1)
plt.plot(threads, s_p_openacc_values, marker='o', color='b', label='OpenACC+OpenMP')
plt.plot(threads, s_p_fftw_values, marker='s', color='g', label='FFTW')
plt.xlabel('Потоки')
plt.ylabel('Ускорение S_p(n)')
plt.title('Ускорение S_p(n) от количества потоков')
plt.legend()
plt.grid(True)

# График стоимости
plt.subplot(1, 2, 2)
plt.plot(threads, c_p_openacc_values, marker='o', color='r', label='OpenACC+OpenMP')
plt.plot(threads, c_p_fftw_values, marker='s', color='purple', label='FFTW')
plt.xlabel('Потоки')
plt.ylabel('Стоимость C_p(n)')
plt.title('Стоимость C_p(n) от количества потоков')
plt.legend()
plt.grid(True)

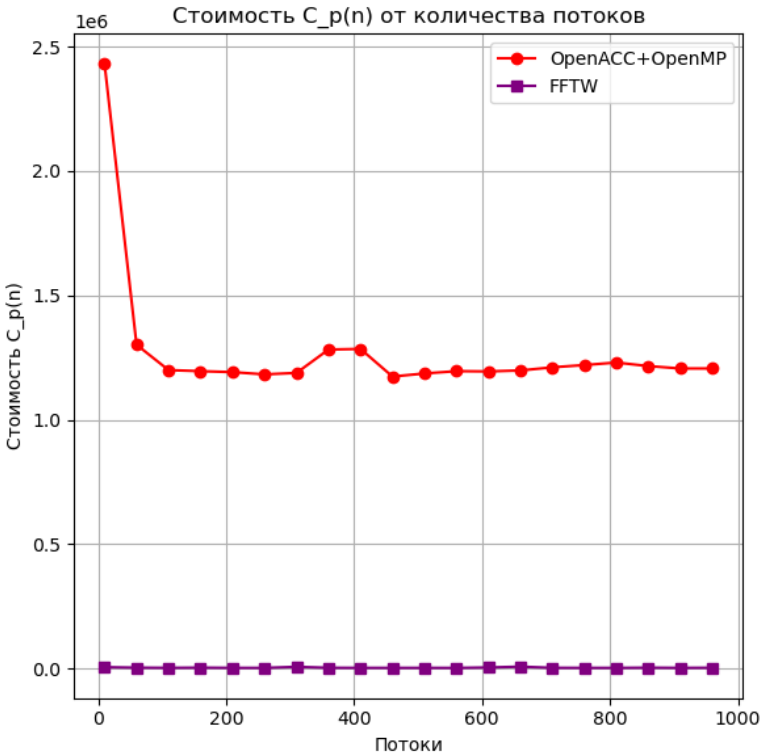
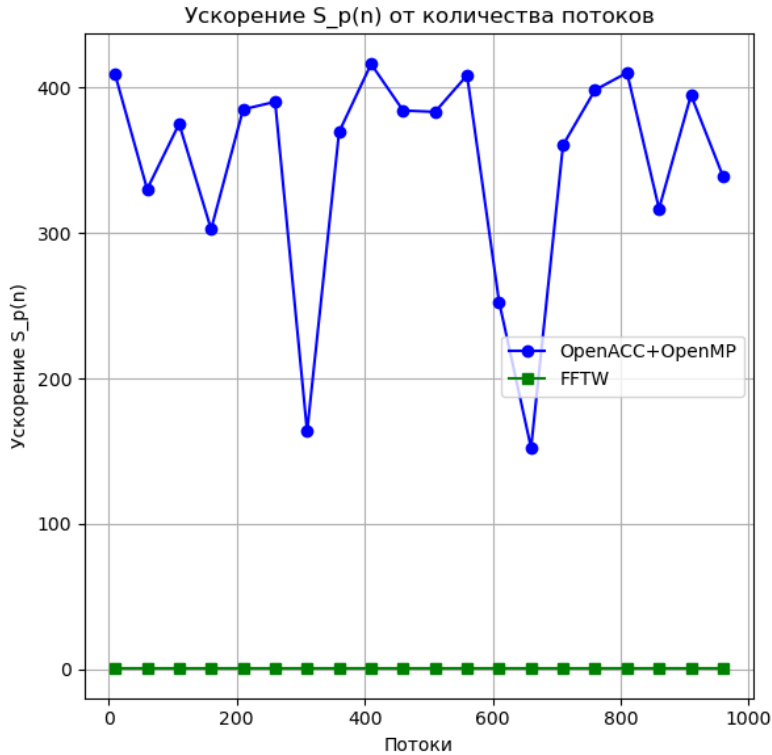
# Показать графики
plt.tight_layout()

```

plt.show()

| Потоки | Время OpenACC + OpenMP (мкс) | Время FFTW (мкс) | Ускорение S_p(n) | Стоимость C_p(n) |

10	303804	743	408.89	2430432
60	162738	493	330.10	1301904
110	150007	400	375.02	1200056
160	149461	494	302.55	1195688
210	148981	387	384.96	1191848
260	147864	379	390.14	1182912
310	148581	908	163.64	1188648
360	160341	434	369.45	1282728
410	160634	386	416.15	1285072
460	146756	382	384.18	1174048
510	148308	387	383.22	1186464
560	149473	366	408.40	1195784
610	149331	591	252.68	1194648
660	149823	984	152.26	1198584
710	151405	420	360.49	1211240
760	152542	383	398.28	1220336
810	153801	375	410.14	1230408
860	152010	480	316.69	1216080
910	150808	382	394.79	1206464
960	150816	445	338.91	1206528



Оценка алогитма:

При 10 потоках максимальное ускорение и относительно высокая стоимость.

При 60-110 потоках наблюдается хороший баланс между временем и стоимостью.

При 460 и более потоках ускорение начинает уменьшаться, что может указывать на избыточное использование потоков с точки зрения производительности.

Этот анализ позволяет выявить оптимальное количество потоков для выполнения задачи, чтобы достичь наибольшего ускорения без излишней нагрузки на систему.

Если хотите, могу рассчитать более детализированное ускорение для каждого количества потоков или предоставить дополнительные метрики

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js