МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Вятский государственный университет»
(ФГБОУ ВО «ВятГУ»)
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Отчёт
Лабораторная работа № 1 по дисциплине
«Параллельное программирование»

Вариант 6

Выполнил студент группы ИВТб-3301-04_____/Бикметов.И.Р
Проверил преподаватель _____ /Исупов К.С

Киров 2025

1 Цель

Получение навыков в исследовании и оценке сложных вычислительных алгоритмов.

2 Задание

Перемножение полиномов с помощью быстрого преобразования Фурье.

3 Словесное описание исследуемого алгоритма

Алгоритм умножения полиномов при помощи быстрого преобразования Фурье состоит из 4-ех частей:

1. Преобразование Фурье для первого полинома;
2. Преобразования Фурье для второго полинома;
3. Умножение полиномов;
4. Обратное преобразование получившегося полинома.

Быстрое преобразование Фурье основывается на том, что степени одних комплексных корней единицы в степени n дают другие. Сначала идет разделение вектора коэффициентов на два вектора, рекурсивно вычисляется значение дискретного преобразования Фурье для них, и после происходит их объединение в одно дискретное преобразование Фурье.

Так как идет постоянное разделение вектора коэффициентов на два, то асимптотическая сложность алгоритма: $(n \cdot log(n))$, где n – размер полинома.

4 Описание набора тестовых примеров и результаты тестирования

В ходе тестирования выполнялось обычное умножение полиномов с асимптотической оценкой $O(n \cdot m)$, где n – это размер первого полинома, а m – второго, а также умножение с помощью быстрого преобразования Фурье.

Размер каждого полинома	Время умножения с помощью быстрого преобразования Фурье, с	Время обычного умножения, с
2^17	1.97862	39.3844
2^18	3.15565	214.978
2^19	6.76427	1033.59
2^20	13.6832	-
2^21	28.6799	-
2^22	79.4351	-

5 Листинг кода

```
    #include <iostream>
#include <vector>
#include <complex>
#include <chrono>
#include <random>
#include <cmath>

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

using namespace std;

// перемножение полиномов
vector<double> MultPoly(vector<double>& poly1, vector<double>& poly2) {
    // получаем размеры полиномов
    int n = poly1.size();
    int m = poly2.size();

    vector<double> result = {};

    // смотрим есть ли смысл их перемножать
    if (n == 0 || m == 0) {
        return result;
    }

    // определяем размер результата
    result.resize(n + m - 1, 0);
    // перемножаем фантанчиком
    for (int i = 0; i < n; i++) {
        if (poly1[i] != 0) {
            for (int j = 0; j < m; j++) {
                result[i + j] = result[i + j] + poly1[i] * poly2[j];
            }
        }
    }

    return result;
}

// прямое преобразование Фурье
vector<complex<double>> FFT(const vector<complex<double>>& vect) {
    int n = vect.size();
    // Выход из рекурсии - преобразование Фурье для одного элемента просто равно этому эле
    if (n == 1) {
        return vector<complex<double>>(1, vect[0]);
    }
```

```
    // вектор для комплексных экспонент
    vector<complex<double>> w(n);
    // вычисление комплексных экспонент
    for (int i = 0; i < n; i++) {
        // =2x/n - вычисление полярного угла
        double alpha = 2 * M_PI * i / n;
        // i=cos+isin - корень из 1
        w[i] = complex<double>(cos(alpha), sin(alpha));
    }

    // Считаем коэффициенты полинома A и B
    vector<complex<double>> A(n / 2); // четные
    vector<complex<double>> B(n / 2); // нечетные
    // заполнение вектороф коэффицентами полинома
    for (int i = 0; i < n / 2; i++) {
        A[i] = vect[i * 2];
        B[i] = vect[i * 2 + 1];
    }

    // получение коэффицентов Фурье
    vector<complex<double>> Av = FFT(A);
    vector<complex<double>> Bv = FFT(B);

    vector<complex<double>> res(n);
    for (int i = 0; i < n; i++){
        // P(i)=A(2i)+iB(2i) - значение преобразования Фурье для частоты i
        res[i] = Av[i % (n / 2)] + w[i] * Bv[i % (n / 2)];
    }

    return res;
}

// обратное преобразование Фурье
void IFFT(vector<complex<double>>& x) {
    int N = x.size();
    // реализация комплексного сопряжения -> a+ib = a-ib
    for (auto& el : x) el = conj(el);
    // Применяем FFT к комплексно-сопряженным элементам
    x = FFT(x);
    // Обратное комплексное сопряжение и нормализация
    for (auto& el : x) el = conj(el) / static_cast<double>(N);
}

// умножение полиномов с применением быстрого преобразования
vector<double> FFTMult(const vector<double>& p1, const vector<double>& p2) {
    int n = p1.size() + p2.size() - 1;
    int size = 1;
    // корректировка длины к степени двойки
    while (size < n) size <<= 1;  // Даем размеры 2, 4, 8, 16, ...
```

```
    vector<complex<double>> f1(size), f2(size);

    // переход к комплексным числам
    for (size_t i = 0; i < p1.size(); ++i) {
        f1[i] = complex<double>(p1[i], 0);
    }
    for (size_t i = 0; i < p2.size(); ++i) {
        f2[i] = complex<double>(p2[i], 0);
    }

    // Выполнение FFT
    vector<complex<double>> resF1 = FFT(f1);
    vector<complex<double>> resF2 = FFT(f2);

    // Перемножение (суть ускорения)
    for (int i = 0; i < size; ++i) {
        resF1[i] *= resF2[i];
    }

    // Выполнение IFFT
    IFFT(resF1);

     //Получение результата
    vector<double> result(n);
    for (int i = 0; i < n; ++i) {
        result[i] = round(resF1[i].real()); // Округляем до ближайшего целого
    }

    return result;
}

// вывод итогового полинома
void PrintResult(vector<double>& data) {
    cout << "Mult result: ";
    for (size_t i = 0; i < data.size(); i++) {
        cout << data[i] << (i < data.size() - 1 ? " + " : " ");
        //if (data[i] >= 0)
        //{
        //    cout << data[i] << (i < data.size() - 1 ? " + " : " ");
        //}
        //else
        //{
        //    cout << data[i];
        //}

    }
    cout << std::endl;
}

// вывод времени в секундах
```

```
void PrintTime(chrono::duration<double> time) {
    cout << "Mult time: " << time.count() << "s" << endl;
}

// генерация случайных чисел типа double
double generateRandomDouble(double lower, double upper) {
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<double> dis(lower, upper);
    return dis(gen);
}


int main()
{
    // Входные данные
    const double LEN = pow(2, 17);
    vector<double> poly1;
    vector<double> poly2;
    poly1.resize(LEN);
    poly2.resize(LEN);

    // заполнение полиномов
    for (long i = 0; i < LEN; i++)
    {
        poly1[i] = generateRandomDouble(-10000000, 10000000);
        poly2[i] = generateRandomDouble(-10000000, 10000000);
    }


    // Умножение полиномов с применением быстрого преобразования
    auto start = chrono::high_resolution_clock::now();
    vector<double> res = FFTMult(poly1, poly2);
    auto end = chrono::high_resolution_clock::now();
    cout << "FFT -> ";
    PrintTime(end - start);
    //PrintResult(res);


    // Классическое умножение полиномов
    start = chrono::high_resolution_clock::now();
    res = MultPoly(poly1, poly2);
    end = chrono::high_resolution_clock::now();
    cout << "Def -> ";
    PrintTime(end - start);
    //PrintResult(res);
}
```

6 Выводы по работе

В ходе лабораторной работы был реализован алгоритм умножения полиномов с помощью быстрого преобразования Фурье. В качестве сравнения выступал обычной алгоритм умножения полиномов. В ходе тестирования было замечено, что алгоритм обычного умножения намного уступает умножению при помощи быстрого преобразования Фурье по времени.