

МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

**ОТЧЕТ ПО ПРАКТИКЕ WORK7**

**ОТЧЕТ**

Студента 3 курса 311 группы  
направления 02.03.02 — Фундаментальная информатика и информационные  
технологии  
факультета КНИИТ  
Забоева Максима Владиславовича

Проверил

Старший преподаватель

\_\_\_\_\_

М. С. Портенко

Саратов 2023

## СОДЕРЖАНИЕ

1	Условие задачи .....	3
2	Практическая часть .....	4
2.1	Последовательная реализация .....	4
2.2	Параллельная реализация .....	6
2.3	Таблица результатов .....	9
3	Тестовые запуски .....	10
3.1	Последовательный алгоритм .....	10
3.2	Параллельный алгоритм .....	10

## 1 Условие задачи

Проведите эксперименты для последовательного и параллельного вычислений БПФ, результаты занесите в таблицу 1. Таблица 1. Результаты вычислительных экспериментов и ускорение вычислений

Номер теста	Размер входного сигнала	Мин. время работы последовательного приложения (сек)	Мин. время работы параллельного приложения (сек)	Ускорение
1	32768			
2	65536			
3	131072			
4	262144			
5	524288			

## 2 Практическая часть

### 2.1 Последовательная реализация

Код программы:

```
#include <iomanip>
#include <iostream>
#include <cmath>
#include <complex>
#include <time.h>
using namespace std;

#define PI (3.14159265358979323846)

//Function for simple initialization of input signal elements
void DummyDataInitialization(complex<double>* mas, int size) {
    for (int i = 0; i < size; i++)
        mas[i] = 0;
    mas[size - size / 4] = 1;
}

// Function for random initialization of objects' elements
void RandomDataInitialization(complex<double>* mas, int size)
{
    srand(unsigned(clock()));
    for (int i = 0; i < size; i++)
        mas[i] = complex<double>(rand() / 1000.0, rand() / 1000.0);
}

//Function for memory allocation and data initialization
void ProcessInitialization(complex<double>*& inputSignal,
    complex<double>*& outputSignal, int& size) {
    // Setting the size of signals
    do
    {
        cout << "Enter the input signal length: ";
        cin >> size;
        if (size < 4)
            cout << "Input signal length should be >= 4" << endl;
        else
        {
            int tmpSize = size;
            while (tmpSize != 1)
            {
                if (tmpSize % 2 != 0)
                {
                    cout << "Input signal length should be powers of two" << endl;
                    size = -1;
                    break;
                }
                tmpSize /= 2;
            }
        }
    } while (size < 4);
    cout << "Input signal length = " << size << endl;
    inputSignal = new complex<double>[size];
    outputSignal = new complex<double>[size];
    //Initialization of input signal elements - tests
    RandomDataInitialization(inputSignal, size);
    //Computational experiments
```

```

        //RandomDataInitialization(inputSignal, size);
    }
    //Function for computational process termination
    void ProcessTermination(complex<double>*& inputSignal,
        complex<double>*& outputSignal) {
        delete[] inputSignal;
        inputSignal = NULL;
        delete[] outputSignal;
        outputSignal = NULL;
    }
    void BitReversing(complex<double>* inputSignal,
        complex<double>* outputSignal, int size) {
        int j = 0, i = 0;
        while (i < size)
        {
            if (j > i)
            {
                outputSignal[i] = inputSignal[j];
                outputSignal[j] = inputSignal[i];
            }
            else
                if (j == i)
                    outputSignal[i] = inputSignal[i];
            int m = size >> 1;
            while ((m >= 1) && (j >= m))
            {
                j -= m;
                m = m >> 1;
            }
            j += m;
            i++;
        }
    }
    __inline void Butterfly(complex<double>* signal,
        complex<double> u, int offset, int butterflySize) {
        complex<double> tem = signal[offset + butterflySize] * u;
        signal[offset + butterflySize] = signal[offset] - tem;
        signal[offset] += tem;
    }
    void SerialFFTCalculation(complex<double>* signal, int size) {
        int m = 0;
        for (int tmp_size = size; tmp_size > 1; tmp_size /= 2, m++);
        for (int p = 0; p < m; p++)
        {
            int butterflyOffset = 1 << (p + 1);
            int butterflySize = butterflyOffset >> 1;
            double coeff = PI / butterflySize;
            for (int i = 0; i < size / butterflyOffset; i++)
                for (int j = 0; j < butterflySize; j++)
                    Butterfly(signal, complex<double>(cos(-j * coeff),
                        sin(-j * coeff)), j + i * butterflyOffset, butterflySize);
        }
    }
    // FFT computation
    void SerialFFT(complex<double>* inputSignal,
        complex<double>* outputSignal, int size) {
        BitReversing(inputSignal, outputSignal, size);
        SerialFFTCalculation(outputSignal, size);
    }
    void PrintSignal(complex<double>* signal, int size) {
        cout << "Result signal" << endl;
    }

```

```

        for (int i = 0; i < size; i++)
            cout << signal[i] << endl;
    }

int main()
{
    complex<double>* inputSignal = NULL;
    complex<double>* outputSignal = NULL;
    int size = 0;
    const int repeatCount = 16;
    double startTime;
    double duration;
    double minDuration = DBL_MAX;
    cout << "Fast Fourier Transform" << endl;
    // Memory allocation and data initialization
    ProcessInitialization(inputSignal, outputSignal, size);
    for (int i = 0; i < repeatCount; i++)
    {
        startTime = clock();
        // FFT computation
        SerialFFT(inputSignal, outputSignal, size);
        duration = (clock() - startTime) / CLOCKS_PER_SEC;
        if (duration < minDuration)
            minDuration = duration;
    }
    cout << setprecision(6);
    cout << "Execution time is " << minDuration << " s. " << endl;
    // Result signal output
    //PrintSignal(outputSignal, size);
    // Computational process termination
    ProcessTermination(inputSignal, outputSignal);
    return 0;
}

```

## 2.2 Параллельная реализация

Код программы:

```

#include <iomanip>
#include <iostream>
#include <cmath>
#include <complex>
#include <time.h>
#include <omp.h>
using namespace std;

#define PI (3.14159265358979323846)

//Function for simple initialization of input signal elements
void DummyDataInitialization(complex<double>* mas, int size) {
    for (int i = 0; i < size; i++)
        mas[i] = 0;
    mas[size - size / 4] = 1;
}

// Function for random initialization of objects' elements
void RandomDataInitialization(complex<double>* mas, int size)
{
    srand(unsigned(clock()));
    for (int i = 0; i < size; i++)

```

```

        mas[i] = complex<double>(rand() / 1000.0, rand() / 1000.0);
    }

    //Function for memory allocation and data initialization
    void ProcessInitialization(complex<double>*& inputSignal,
        complex<double>*& outputSignal, int& size) {
        // Setting the size of signals
        do
        {
            cout << "Enter the input signal length: ";
            cin >> size;
            if (size < 4)
                cout << "Input signal length should be >= 4" << endl;
            else
            {
                int tmpSize = size;
                while (tmpSize != 1)
                {
                    if (tmpSize % 2 != 0)
                    {
                        cout << "Input signal length should be powers of two" << endl;
                        size = -1;
                        break;
                    }
                    tmpSize /= 2;
                }
            }
        } while (size < 4);
        cout << "Input signal length = " << size << endl;
        inputSignal = new complex<double>[size];
        outputSignal = new complex<double>[size];
        //Initialization of input signal elements - tests
        RandomDataInitialization(inputSignal, size);
        //Computational experiments
        //RandomDataInitialization(inputSignal, size);
    }

    //Function for computational process termination
    void ProcessTermination(complex<double>*& inputSignal,
        complex<double>*& outputSignal) {
        delete[] inputSignal;
        inputSignal = NULL;
        delete[] outputSignal;
        outputSignal = NULL;
    }

    void BitReversing(complex<double>* inputSignal,
        complex<double>* outputSignal, int size) {
        int j = 0, i = 0;
        while (i < size)
        {
            if (j > i)
            {
                outputSignal[i] = inputSignal[j];
                outputSignal[j] = inputSignal[i];
            }
            else
            {
                if (j == i)
                    outputSignal[i] = inputSignal[i];
            }
            int m = size >> 1;
            while ((m >= 1) && (j >= m))
            {
                j -= m;
            }
        }
    }

```

```

        m = m >> 1;
    }
    j += m;
    i++;
}
}

__inline void Butterfly(complex<double>* signal,
    complex<double> u, int offset, int butterflySize) {
    complex<double> tem = signal[offset + butterflySize] * u;
    signal[offset + butterflySize] = signal[offset] - tem;
    signal[offset] += tem;
}

void ParallelFFTCalculation(complex<double>* signal, int size) {
    int m = 0;
    for (int tmp_size = size; tmp_size > 1; tmp_size /= 2, m++);
#pragma omp parallel for
    for (int p = 0; p < m; p++)
    {
        int butterflyOffset = 1 << (p + 1);
        int butterflySize = butterflyOffset >> 1;
        double coeff = PI / butterflySize;
#pragma omp parallel for
        for (int i = 0; i < size / butterflyOffset; i++)
            for (int j = 0; j < butterflySize; j++)
                Butterfly(signal, complex<double>(cos(-j * coeff),
                    sin(-j * coeff)), j + i * butterflyOffset, butterflySize);
    }
}

void ParallelFFT(complex<double>* inputSignal,
    complex<double>* outputSignal, int size) {
    BitReversing(inputSignal, outputSignal, size);
    ParallelFFTCalculation(outputSignal, size);
}

void PrintSignal(complex<double>* signal, int size) {
    cout << "Result signal" << endl;
    for (int i = 0; i < size; i++)
        cout << signal[i] << endl;
}

int main()
{
    complex<double>* inputSignal = NULL;
    complex<double>* outputSignal = NULL;
    int size = 0;
    const int repeatCount = 16;
    double startTime;
    double duration;
    double minDuration = DBL_MAX;
    cout << "Fast Fourier Transform" << endl;
    // Memory allocation and data initialization
    ProcessInitialization(inputSignal, outputSignal, size);
    for (int i = 0; i < repeatCount; i++)
    {
        startTime = clock();
        // FFT computation
        ParallelFFT(inputSignal, outputSignal, size);
        duration = (clock() - startTime) / CLOCKS_PER_SEC;
        if (duration < minDuration)

```



```

        minDuration = duration;
    }
    cout << setprecision(6);
    cout << "Execution time is " << minDuration << " s. " << endl;
    // Result signal output

    // Computational process termination
    ProcessTermination(inputSignal, outputSignal);
    return 0;
}

```

## 2.3 Таблица результатов

Номер теста	Размер входного сигнала	Мин. время работы последовательного приложения (сек)	Мин. время работы параллельного приложения (сек)	Ускорение
1	32768	0.018	0.003	6
2	65536	0.037	0.006	6,166666667
3	131072	0.085	0.015	5,666666667
4	262144	0.169	0.034	4,970588235
5	524288	0.367	0.075	4,893333333

## 3 Тестовые запуски

### 3.1 Последовательный алгоритм

```
Консоль отладки Microsoft Visual Studio
Fast Fourier Transform
Enter the input signal length: 65536
Input signal length = 65536
Execution time is 0.038 s.

E:\games\tests\Parallel\x64\Debug\Parallel.exe
Чтобы автоматически закрывать консоль при остановке отладки, нажмите любую клавишу, чтобы закрыть это окно.
```

```
Консоль отладки Microsoft Visual Studio
Fast Fourier Transform
Enter the input signal length: 262144
Input signal length = 262144
Execution time is 0.181 s.

E:\games\tests\Parallel\x64\Debug\Parallel.exe
Чтобы автоматически закрывать консоль при остановке отладки, нажмите любую клавишу, чтобы закрыть это окно.
```

### 3.2 Параллельный алгоритм

```
Консоль отладки Microsoft Visual Studio
Fast Fourier Transform
Enter the input signal length: 32768
Input signal length = 32768
Execution time is 0.003 s.

E:\games\tests\Parallel\x64\Debug\Parallel.exe
Чтобы автоматически закрывать консоль при остановке отладки, нажмите любую клавишу, чтобы закрыть это окно.
```

```
Консоль отладки Microsoft Visual Studio
Fast Fourier Transform
Enter the input signal length: 65536
Input signal length = 65536
Execution time is 0.006 s.

E:\games\tests\Parallel\x64\Debug\Parallel.exe
Чтобы автоматически закрывать консоль при остановке отладки, нажмите любую клавишу, чтобы закрыть это окно.
```