

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

**ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ.
WORK07**

ОТЧЕТ О ПРАКТИКЕ

студентки 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные
технологии
факультета КНиИТ
Мигуновой Анастасии Александровны

Проверил

Старший преподаватель

М. С. Портенко

Саратов 2023

СОДЕРЖАНИЕ

1	Work 7	3
1.1	Условие задачи	3
1.2	Последовательное вычисление БПФ	3
1.2.1	Фрагмент кода	4
1.3	Параллельное вычисление БПФ	7
1.3.1	Фрагмент кода	7
1.4	Результат работы программы	11
2	Характеристики компьютера	13

1 Work 7

1.1 Условие задачи

Проведите эксперименты для последовательного и параллельного вычислений БПФ (Быстрое преобразование Фурье), результаты занесите в таблицу 1.

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			

Рисунок 1 – Таблица 1. Результаты вычислительных экспериментов и ускорение вычислений

Оцените эффективность динамического планирования в OpenMP-версии по сравнению со статическим.

1.2 Последовательное вычисление БПФ

Алгоритм Cooley-Tukey Состоит из двух шагов:

1. Преобразование входного массива данных (бит-реверсирование).
2. Вычисление БПФ.

Бит-реверсирование. Данный этап заключается в изменении порядка следования исходных данных. Двоичный код индекса элемента преобразуется путем изменения порядка следования бит в нем на противоположный. В результате получим данные, расположенные в той последовательности, которая совпадает с окончательным порядком следования прореженных входных отсчетов. Исходными данными для вычисления БПФ является входной сигнал, заданный в виде массива комплексных чисел.

Исходными данными для вычисления БПФ является входной сигнал, заданный в виде массива комплексных чисел. Входной сигнал будет задаваться двумя способами:

1. специальный вид входного сигнала (для оценки корректности реализации алгоритма);

2. случайная генерация данных (для проведения вычислительных экспериментов).

1.2.1 Фрагмент кода

```
#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;
}

```

1.3 Параллельное вычисление БПФ

1.3.1 Фрагмент кода

```

#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;
    }
    while (size < 1);
}

```

```

        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++);
    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);
    }
}

// FFT computation
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
PrintSignal(outputSignal, size);
// Computational process termination
ProcessTermination(inputSignal, outputSignal);
return 0;
}

```

1.4 Результат работы программы

Номер теста	Размер входного сигнала	Мин. время работы последовательного приложения (сек)	Мин. время работы параллельного приложения (сек)	Ускорение
1	32768	0.049	0.008	6,125
2	65536	0.106	0.018	5,888
3	131072	0.229	0.04	5,725
4	262144	0.479	0.087	5,505
5	524288	1.028	0.19	5,41

Рисунок 2 – Таблица 1

```
Fast Fourier Transform
Enter the input signal length: 32768
Input signal length = 32768
Execution time is 0.049 s.
```

Рисунок 3 – Последовательная реализация

```
Fast Fourier Transform
Enter the input signal length: 65536
Input signal length = 65536
Execution time is 0.106 s.
```

Рисунок 4 – Последовательная реализация

```
Fast Fourier Transform
Enter the input signal length: 131072
Input signal length = 131072
Execution time is 0.229 s.
```

Рисунок 5 – Последовательная реализация

```
Fast Fourier Transform
Enter the input signal length: 262144
Input signal length = 262144
Execution time is 0.479 s.
```

Рисунок 6 – Последовательная реализация

```
Fast Fourier Transform
Enter the input signal length: 524288
Input signal length = 524288
Execution time is 1.028 s.
```

Рисунок 7 – Последовательная реализация

```
Fast Fourier Transform
Enter the input signal length: 32768
Input signal length = 32768
Execution time is 0.008 s.
```

Рисунок 8 – Параллельная реализация

```
Fast Fourier Transform
Enter the input signal length: 65536
Input signal length = 65536
Execution time is 0.018 s.
```

Рисунок 9 – Параллельная реализация

```
Fast Fourier Transform
Enter the input signal length: 131072
Input signal length = 131072
Execution time is 0.04 s.
```

Рисунок 10 – Параллельная реализация

```
Fast Fourier Transform
Enter the input signal length: 262144
Input signal length = 262144
Execution time is 0.087 s.
```

Рисунок 11 – Параллельная реализация

```
Fast Fourier Transform
Enter the input signal length: 524288
Input signal length = 524288
Execution time is 0.19 s.
```

Рисунок 12 – Параллельная реализация

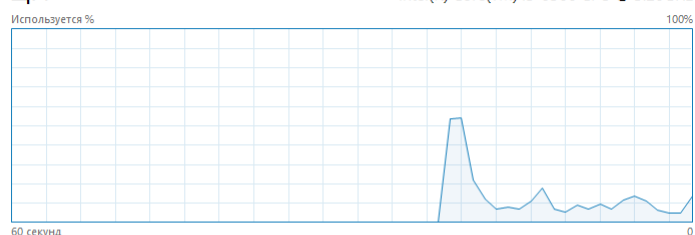
2 Характеристики компьютера

Характеристики устройства

Имя устройства	DESKTOP-MSS8D39
Процессор	Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz 3.20 GHz
Оперативная память	8,00 ГБ
Код устройства	E3B8953D-13B0-42A7-944B-1ED9FD0E C328
Код продукта	00330-80000-00000-AA153
Тип системы	64-разрядная операционная система, процессор x64

ЦП

Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz



Использование	Скорость	Базовая скорость:	3,20 ГГц
14%	3,43 ГГц	Сокетов:	1
Процессы	Потоки	Ядра:	4
220	3285	Логических процессоров:	4
Время работы	Дескрипторы	Виртуализация:	Включено
100:23:51:24	170005	Кэш L1:	256 КБ
		Кэш L2:	1,0 МБ
		Кэш L3:	6,0 МБ