

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 WORK18

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
3	Результаты работы	8
3.1	Характеристики компьютера	8
3.2	Фото результатов	8

1 Условие задачи

Аналогично работе с ОМР выполните следующее задание через МРІ.

Предподготовка Рассчитайте амплитудный спектр тестового сигнала с частотой 10Hz и амплитудой 1 ($\sin(2\pi \cdot 10 \cdot t)$). Длительность сигнала составляет 1 секунду. Частота дискретизации равна числу отсчетов и равна 128. Для значений амплитуды, полученных при помощи БПФ, выполните операцию нормализации.

Задание Периодический сигнал с периодом T , равным 1 секунде, задается функцией слева от знака равенства. Выполните дискретизацию сигнала таким образом, чтобы разрешение по частоте составляло 1 Hz при числе отсчетов 1024. Согласно своему варианту:

1. рассчитайте коэффициенты ряда Фурье, используя параллельную программу БПФ;
2. вычислите значения функции

$$f = \frac{a_0}{2} + \sum_{k=1}^{511} (a_k \cos(k \frac{2\pi}{T} t) + b_k \sin(k \frac{2\pi}{T} t)), 0 < t < T$$

подставив в выражение рассчитанные коэффициенты ряда Фурье;

3. сравните подсчитанные значения функции с полученными аналитическим разложением в ряд Фурье и с точными значениями функции при $0 < t < T$.

Варианты заданий 1. $\frac{\pi}{2} - \frac{\pi t}{T} = \sum_{k=1}^{\infty} \frac{\sin(k \frac{2\pi}{T} t)}{k}, 0 < t < T$

2 Практическая часть

Код программы:

```
#include <iomanip>
#include <iostream>
#include <cmath>
#include <complex>
#include <vector>
#include <time.h>
#include <mpi.h>
using namespace std;
#define PI (3.14159265358979323846)
//Function for simple initialization of input signal elements
void DummyDataInitialization(complex<double>* mas, int size) {
    complex<double>* mas1 = NULL;
    MPI_Bcast(&size, 6, MPI_INT, 0, MPI_COMM_WORLD);
    for (int i = 0; i < size; i++) {
        mas[i] = (PI / 2.0) - ((PI * (i + 1)) / 1024.0);
    }
    MPI_Reduce(&mas, &mas1, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
}
//Function for memory allocation and data initialization
void ProcessInitialization(complex<double>*& inputSignal,
    complex<double>*& outputSignal, int& size) {
    cout << "Input signal length = " << size << endl;
    inputSignal = new complex<double>[size];
    outputSignal = new complex<double>[size];
    DummyDataInitialization(inputSignal, 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;
    complex<double>* inputSignal1 = NULL;
    int ProcRank, ProcNum;
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    MPI_Bcast(&size, 6, MPI_INT, 0, MPI_COMM_WORLD);
    //pragma omp parallel for
    for (i = 0; i < size; ++i) {
        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;
    }
    MPI_Reduce(&outputSignal, &inputSignal1, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
}

__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;
    int ProcRank, ProcNum;
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    MPI_Bcast(&size, 6, MPI_INT, 0, MPI_COMM_WORLD);
    for (int tmp_size = size; tmp_size > 1; tmp_size /= 2, m++);
    MPI_Bcast(&size, 6, MPI_INT, 0, MPI_COMM_WORLD);
    complex<double>* signal1 = NULL;
    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,
                    signal1);
        MPI_Reduce(&signal, &signal1, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    }
}

// 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;
}

void TestResult(complex<double>* inputSignal, complex<double>* outputSignal, int size)
{
    complex<double>* testFunc = new complex<double>[size];
    for (int i = 0; i < size; i++) {
        testFunc[i] = outputSignal[0].real() / 4;
        for (int k = 1; k < size; k++) {
            testFunc[i] += (outputSignal[k].real() / 2 * cos((k * 2 * PI
                * (i + 1)) / 1024.0) - outputSignal[k].imag() / 2 * sin((k * 2 * PI * (i + 1)) / 1024.0));
        }
        testFunc[i] /= size / 2;
    }
}

```

```

    }
    vector<double> testFuncToch(size);
    for (int i = 0; i < size; i++) {
        testFuncToch[i] = 0;
        for (int k = 1; k < size * 16; k++)
            testFuncToch[i] += sin((k * 2 * PI * (i + 1)) / 1024.0) / k;
    }

    for (int i = 0; i < size; i++) {
        cout << testFunc[i].real() << setw(10) << testFuncToch[i] << setw(10) << inputSignal[i].real() << endl;
    }
}

int main()
{
    setlocale(LC_ALL, "Rus");
    complex<double>* inputSignal = NULL;
    MPI_Init(NULL, NULL);
    int ProcRank, ProcNum;
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    complex<double>* outputSignal = NULL;
    int size = 1024;
    double startTime, finishTime;
    double duration;
    double minDuration = DBL_MAX;
    cout << "Fast Fourier Transform" << endl;
    // Memory allocation and data initialization
    ProcessInitialization(inputSignal, outputSignal, size);
    startTime = MPI_Wtime();
    ParallelFFT(inputSignal, outputSignal, size);
    int half_size = size / 2;
    for (int i = 0; i < size; i++) {
        complex<double>* outputSignal1 = NULL;
        if (abs(outputSignal[i].real()) > 0.000001 || abs(outputSignal[i].imag()) > 0.000001) {
            outputSignal[i] = sqrt(pow(outputSignal[i].real(), 2) + pow(outputSignal[i].imag(), 2)) / half_size;
            //cout << outputSignal[i].real() << endl;
        }
        else
            outputSignal[i] = 0;
        MPI_Reduce(&outputSignal, &outputSignal1, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    }

    finishTime = MPI_Wtime();
    duration = (finishTime - startTime) / CLOCKS_PER_SEC;
    ProcessTermination(inputSignal, outputSignal);

    size = 1024;
    ProcessInitialization(inputSignal, outputSignal, size);
    ParallelFFT(inputSignal, outputSignal, size);
    // Result signal output
    //PrintSignal(outputSignal, size);
    // Computational process termination
    TestResult(inputSignal, outputSignal, size);

    ProcessTermination(inputSignal, outputSignal);
    MPI_Finalize();
}

```

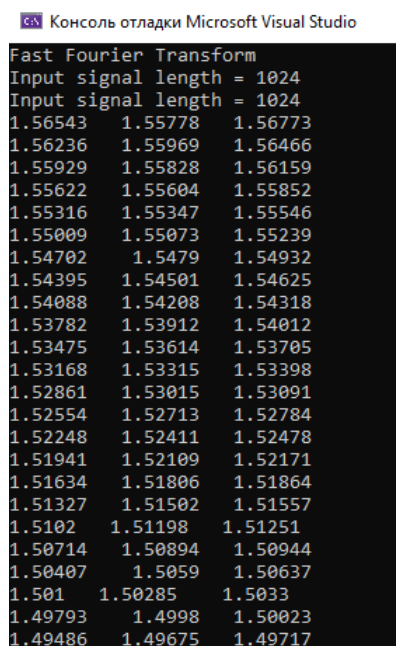
```
    return 0;  
}
```

3 Результаты работы

3.1 Характеристики компьютера

Процессор — 12th Gen Intel Core i5-12600KF, Базовая скорость 3,70ГГц,
Кол-во ядер 10, Кол-во процессоров 16 (включая 4 энергоэффективных ядра).
16гб Оперативной памяти, скорость 3200МГц

3.2 Фото результатов



```
Консоль отладки Microsoft Visual Studio
Fast Fourier Transform
Input signal length = 1024
Input signal length = 1024
1.56543 1.55778 1.56773
1.56236 1.55969 1.56466
1.55929 1.55828 1.56159
1.55622 1.55604 1.55852
1.55316 1.55347 1.55546
1.55009 1.55073 1.55239
1.54702 1.5479 1.54932
1.54395 1.54501 1.54625
1.54088 1.54208 1.54318
1.53782 1.53912 1.54012
1.53475 1.53614 1.53705
1.53168 1.53315 1.53398
1.52861 1.53015 1.53091
1.52554 1.52713 1.52784
1.52248 1.52411 1.52478
1.51941 1.52109 1.52171
1.51634 1.51806 1.51864
1.51327 1.51502 1.51557
1.5102 1.51198 1.51251
1.50714 1.50894 1.50944
1.50407 1.5059 1.50637
1.501 1.50285 1.5033
1.49793 1.4998 1.50023
1.49486 1.49675 1.49717
```