

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

**ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ.
WORK18**

ОТЧЕТ О ПРАКТИКЕ

студента 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные
технологии
факультета КНиИТ
Вильцева Данила Денисовича

Проверил

Старший преподаватель

М. С. Портенко

Саратов 2023

СОДЕРЖАНИЕ

| | | |
|-----|--|----|
| 1 | Work 18..... | 3 |
| 1.1 | Условие задачи..... | 3 |
| 1.2 | Решение. Последовательная версия | 3 |
| 1.3 | Решение. Параллельная версия..... | 7 |
| 2 | Результат работы | 11 |
| 3 | Характеристики компьютера..... | 12 |

1 Work 18

1.1 Условие задачи

Аналогично работе с OMP выполните следующее задание через MPI.

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:

- рассчитайте коэффициенты ряда Фурье, используя параллельную программу БПФ;
- 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;
- сравните подсчитанные значения функции с полученными аналитическим разложением в ряд Фурье и с точными значениями функции при $0 < t < T$.

Вариант 2

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

1.2 Решение. Последовательная версия

```
#include <iostream>
#include <iomanip>
#include <complex>
#include <math.h>
#include <random>
#include <mpi.h>

using namespace std;

const double T = 1;
const int INF = 1000000000;
const long double PI = 3.1415926535897932384626433832795;
```

```

const int logN = 10;
const int N = 1 << logN;

inline int reverse(int a, int n) {
    int b = 0;
    for (long i = 0; i < long(n); i++)
        if ((a & (1 << i)) != 0)
            b |= 1 << (n - 1 - i);
    return b;
}

void ParallelFFTCalculationBit(complex <double> a[N], int n, bool inv) {
    static complex <double> w[logN][N / 2];
    static bool prep = false;

    if (!prep) {
        for (long i = 0; i < long(logN); i++) {
            long double ang = PI / (1 << i);
            for (long j = 0; j < long(1 << i); j++)
                w[i][j] = complex <double>(cos(ang * -j), sin(ang * -j));
        }
        prep = true;
    }

    int ProcRank, ProcNum;

    int logn = 0;
    int logNumprocs = 0;
    while ((1 << logn) < n)
        logn++;
    while ((1 << logNumprocs) < ProcNum)
        logNumprocs++;
    int szproc = (N / ProcNum);
    for (int i = ProcRank * szproc; i < (ProcRank + 1) * szproc; i++) {
        int ni = reverse(i, logn);
        if (i / szproc != ni / szproc)
            a[i] = a[ni];
        else if (i < ni)
            swap(a[i], a[ni]);
    }

    for (long i = 0; i < long(logn - logNumprocs); i++) {
        complex <double> cur, l, r;
        for (int j = ProcRank * szproc; j < (ProcRank + 1) * szproc; j += (1 << (i +
↵ 1))) {
            for (int k = j; k < j + (1 << i); k++) {
                cur = w[i][k - j];
                l = a[k];
            }
        }
    }
}

```

```

        r = a[k + (1 << i)] * cur;
        a[k] = 1 + r;
        a[k + (1 << i)] = 1 - r;
    }
}

for (long i = long(logn - logNumprocs); i < long(logn); i++) {
    for (int j = 0; j < n; j += (1 << (i + 1))) {
        for (int k = j; k < j + (1 << i); k++) {
            complex <double> cur = inv ? conj(w[i][k - j]) : w[i][k -
↪ j];

            complex <double> l = a[k];
            complex <double> r = a[k + (1 << i)] * cur;
            a[k] = 1 + r;
            a[k + (1 << i)] = 1 - r;
        }
    }
}

complex <double> a[N];

void getSignal() {
    for (long i = 1; i < long(N); i++) {
        double t = i * 1.0 / N;
        a[i] = log(2. * sin(PI * t / T));
    }
}

void experiment(double& minT) {
    getSignal();

    double t = MPI_Wtime();
    ParallelFFTCalculationBit(a, N, 0);
    minT = min(minT, MPI_Wtime() - t);
}

int main() {
    setlocale(LC_ALL, "rus");
    srand(time(NULL));

    int ProcRank, ProcNum;

    double minT = INF;
    experiment(minT);
}

```

```

for (long i = 0; i < N; i++) {
    a[i] *= 2.0;
    a[i] /= N;
}

if (ProcRank == 0) {
    cout << "Размер сигнала = " << N << endl;
    cout << left << setw(10) << "Function" << " "
        << setw(10) << "Fourier" << " " << setw(10) << "Exact value" <<
↪ endl;
}

double error = 0;

for (long k = 1; k < N / 2; k++) {
    double t = k * 1.0 / N;

    double fr = 0;
    double f = 0;
    for (int i = ProcRank + 1; i < N / 2; i += ProcNum) {
        fr += a[i].real() * cos(double(i) * 2.0 * PI * t / T) +
            a[i].imag() * sin(double(i) * 2.0 * PI * t / T);
    }

    f += a[0].real() / 2.0;

    double needr = 0;
    double need = 0;
    for (int i = ProcRank + 1; i < 1000000; i += ProcNum) {
        needr += cos(double(i) * 2.0 * PI * t / T) / double(i);
    }

    double exact = log(2.0 * sin(PI * t / T));

    if (ProcRank == 0) {
        cout << setw(10) << f << " " << setw(10) << need << " " << setw(10)
↪ << exact << endl;
        error += fabs(f - exact);
    }
}

if (ProcRank == 0) {
    cout << "Время: " << minT << endl;
}

}

```

1.3 Решение. Параллельная версия

```
#include <iostream>
#include <iomanip>
#include <complex>
#include <math.h>
#include <random>
#include <mpi.h>

using namespace std;

const double T = 1;
const int INF = 1000000000;
const long double PI = 3.1415926535897932384626433832795;

const int logN = 10;
const int N = 1 << logN;

inline int reverse(int a, int n) {
    int b = 0;
    for (long i = 0; i < long(n); i++)
        if ((a & (1 << i)) != 0)
            b |= 1 << (n - 1 - i);
    return b;
}

void ParallelFFTCalculationBit(complex <double> a[N], int n, bool inv) {
    static complex <double> w[logN][N / 2];
    static bool prep = false;

    if (!prep) {
        for (long i = 0; i < long(logN); i++) {
            long double ang = PI / (1 << i);
            for (long j = 0; j < long(1 << i); j++)
                w[i][j] = complex <double>(cos(ang * -j), sin(ang * -j));
        }
        prep = true;
    }

    int ProcRank, ProcNum;
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    int logn = 0;
    int logNumprocs = 0;
    while ((1 << logn) < n)
        logn++;
    while ((1 << logNumprocs) < ProcNum)
        logNumprocs++;
```

```

int szproc = (N / ProcNum);
for (int i = ProcRank * szproc; i < (ProcRank + 1) * szproc; i++) {
    int ni = reverse(i, logn);
    if (i / szproc != ni / szproc)
        a[i] = a[ni];
    else if (i < ni)
        swap(a[i], a[ni]);
}

for (long i = 0; i < long(logn - logNumprocs); i++) {
    complex <double> cur, l, r;
    for (int j = ProcRank * szproc; j < (ProcRank + 1) * szproc; j += (1 << (i +
↵ 1))) {

        for (int k = j; k < j + (1 << i); k++) {
            cur = w[i][k - j];
            l = a[k];
            r = a[k + (1 << i)] * cur;
            a[k] = l + r;
            a[k + (1 << i)] = l - r;
        }
    }
}

for (long i = 0; i < long(ProcNum); i++)
    MPI_Bcast(a + i * (N / ProcNum), N / ProcNum,
        MPI_DOUBLE_COMPLEX, i, MPI_COMM_WORLD);

for (long i = long(logn - logNumprocs); i < long(logn); i++) {
    for (int j = 0; j < n; j += (1 << (i + 1))) {
        for (int k = j; k < j + (1 << i); k++) {
            complex <double> cur = inv ? conj(w[i][k - j]) : w[i][k -
↵ j];

            complex <double> l = a[k];
            complex <double> r = a[k + (1 << i)] * cur;
            a[k] = l + r;
            a[k + (1 << i)] = l - r;
        }
    }
}

complex <double> a[N];

void getSignal() {
    for (long i = 1; i < long(N); i++) {
        double t = i * 1.0 / N;
        a[i] = log(2. * sin(PI * t / T));
    }
}

```



```

}

void experiment(double& minT) {
    getSignal();

    double t = MPI_Wtime();
    ParallelFFTCalculationBit(a, N, 0);
    minT = min(minT, MPI_Wtime() - t);
}

int main() {
    setlocale(LC_ALL, "rus");
    srand(time(NULL));
    MPI_Init(NULL, NULL);
    int ProcRank, ProcNum;
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    double minT = INF;
    experiment(minT);

    for (long i = 0; i < N; i++) {
        a[i] *= 2.0;
        a[i] /= N;
    }

    if (ProcRank == 0) {
        cout << "Размер сигнала = " << N << endl;
        cout << left << setw(10) << "Function" << " "
            << setw(10) << "Fourier" << " " << setw(10) << "Exact value" <<
↵ endl;
    }

    double error = 0;

    for (long k = 1; k < N / 2; k++) {
        double t = k * 1.0 / N;

        double fr = 0;
        double f = 0;
        for (int i = ProcRank + 1; i < N / 2; i += ProcNum) {
            fr += a[i].real() * cos(double(i) * 2.0 * PI * t / T) +
                a[i].imag() * sin(double(i) * 2.0 * PI * t / T);
        }
        MPI_Reduce(&fr, &f, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
        f += a[0].real() / 2.0;

        double needr = 0;
    }
}

```

```

    double need = 0;
    for (int i = ProcRank + 1; i < 1000000; i += ProcNum) {
        needr += cos(double(i) * 2.0 * PI * t / T) / double(i);
    }
    MPI_Reduce(&needr, &need, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

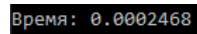
    double exact = log(2.0 * sin(PI * t / T));

    if (ProcRank == 0) {
        cout << setw(10) << f << " " << setw(10) << need << " " << setw(10)
↵ << exact << endl;
        error += fabs(f - exact);
    }
}

if (ProcRank == 0) {
    cout << "Время: " << minT << endl;
}
MPI_Finalize();
}

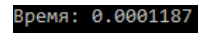
```

2 Результат работы



Время: 0.0002468

Рисунок 1 – Время выполнения последовательной версии



Время: 0.0001187

Рисунок 2 – Время выполнения параллельной версии

Таким образом, MPI даёт ускорение в 2.079 раз

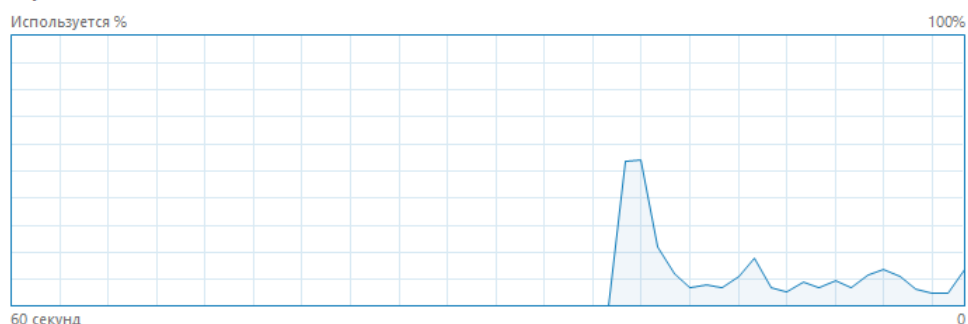
3 Характеристики компьютера

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 | E3BB953D-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 скорости: | 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 |