Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 3

по курсу «Распределение параллельных и распределённых программ»

«Параллельная реализация решения системы линейных алгебраических уравнений с помощью OpenMP»

Студент группы ИУ9-51Б Горбунов А. Д.

Преподаватель Царёв А. С.

Москва 2024

1 Задача

Переделать предыдущую лабораторную работу используя OpenMP вместо MPI.

2 Код решения

Файл main.cpp:

```
#include <omp.h>
#include <iostream>
#include <math.h>
#include <chrono>
#include <thread>

using namespace std;

const double E = 0.00001;
const int N = 50000;

void multMatrixes(double x[N], double y[N], short** A) {
    #pragma omp parallel for
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            y[i] += A[i][j] * x[j];
}

void subMatrixes(double x[N], double y[N], double z[N]) {
    #pragma omp parallel for
    for (int i = 0; i < N; i++)
        z[i] = x[i] - y[i];
}

double scalMatrixes(double u[N], double v[N]) {
```

```
    double result = 0;
    #pragma omp parallel for reduction(+:result)
    for (int i = 0; i < N; i++)
        result += u[i] * v[i];
    return result;
}

double powMatrix(double u[N]) {
    double sum = 0;
    #pragma omp parallel for reduction(+:sum)
    for (int i = 0; i < N; i++)
        sum += u[i] * u[i];
    return sqrt(sum);
}

void printMatrix(double x[N]) {
    printf("\n");
    for (int i = 0; i < N; i++)
        printf("%i\t%f\n", i + 1, x[i]);
    printf("\n");
}

void fillMatrix(double x[N], double a) {
    #pragma omp parallel for
    for (int i = 0; i < N; i++)
        x[i] = a;
}

int kritEnd_bool(double x_n[N], double b[N], short** A) {
    double Ax_n[N];
    fillMatrix(Ax_n, 0);

    #pragma omp parallel for
```

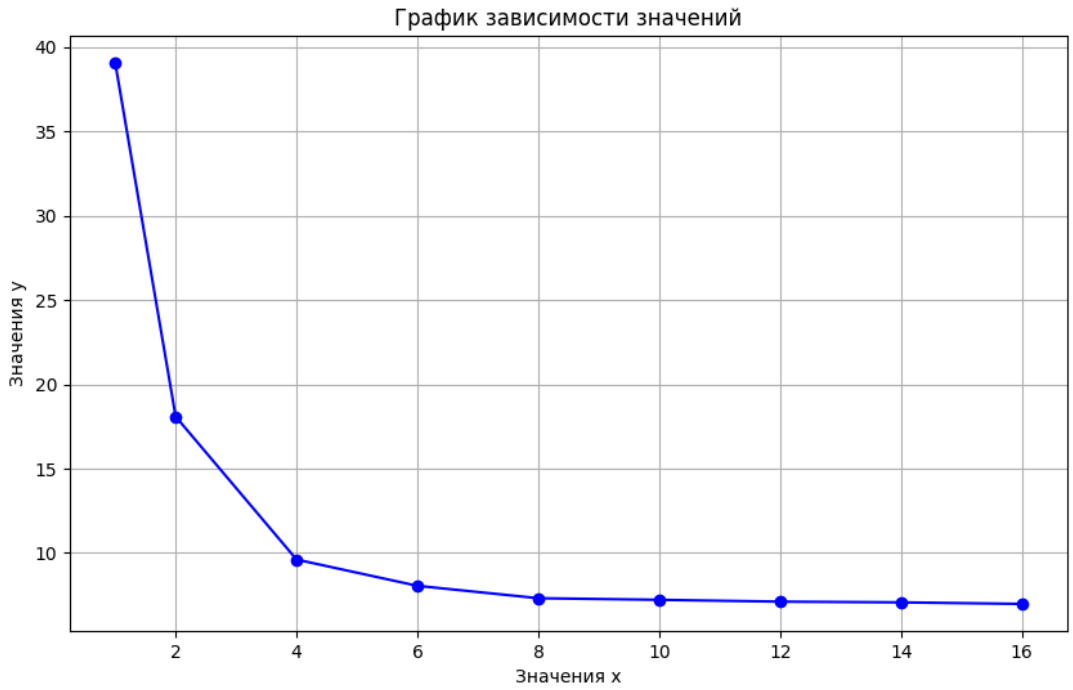
```
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            Ax_n[i] += A[i][j] * x_n[j];

    double z[N];
    fillMatrix(z, 0);

    #pragma omp parallel for
    for (int i = 0; i < N; i++)
        z[i] = Ax_n[i] - b[i];

    double norm_z = powMatrix(z);
    double norm_b = powMatrix(b);

    double k = norm_z / norm_b;

    if (k <= E)
        return 0;
    else
        return 1;
}

void mainProg() {
    double x_n[N];
    double b[N];
    int f = 1;

    short** A = new short*[N];
    for (int i = 0; i < N; i++)
        A[i] = new short[N];

    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
```

```
    for (int j = 0; j < N; j++) {
        if (i == j) {
            A[i][j] = 2;
        } else {
            A[i][j] = 1;
        }
    }
}

fillMatrix(x_n, 0);
fillMatrix(b, N + 1);

do {
    double Ax_n[N];
    fillMatrix(Ax_n, 0);
    multMatrixes(x_n, Ax_n, A);

    double y_n[N];
    fillMatrix(y_n, 0);
    subMatrixes(Ax_n, b, y_n);

    double Ay_n[N];
    fillMatrix(Ay_n, 0);
    multMatrixes(y_n, Ay_n, A);

    double r_n = scalMatrixes(y_n, Ay_n) / scalMatrixes(Ay_n, Ay_n);

    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        y_n[i] = r_n * y_n[i];
    }

    subMatrixes(x_n, y_n, x_n);
```

```
        f = kritEnd_bool(x_n, b, A);

    } while (f != 0);

    for (int i = 0; i < N; i++)
        delete[] A[i];
    delete[] A;
}

void prog(int n) {
    omp_set_num_threads(n);
    auto start_time = std::chrono::steady_clock::now();

    mainProg();

    auto current_time = std::chrono::steady_clock::now();
    auto elapsed_time = std::chrono::duration_cast<std::chrono::milliseconds>
    (current_time - start_time).count();
    printf("Потоки: %d\tВремя: ", n);
    cout << elapsed_time / 1000.0 << endl;
}

int main() {
    omp_set_dynamic(0);

    prog(1);
    prog(2);
    prog(3);
    prog(4);
    prog(5);
    prog(6);
    prog(7);
```

```
    prog(8);
    prog(9);
    prog(10);

    return 0;
}
```

3 График зависимости времени выполнения от числа потоков для N = 50000



4 Заключение

В данной работе я изучил возможности языка C++ в работе с библиотекой OpenMP. Также на основе графика можно сделать вывод, что OpenMP лучше оптимизирует по сравнению с MPI. Кроме того OpenMP легче в использовании.

5 Результат запуска

```
goarty@GoComp:~/Documents/paral_program/lab_3/src$ c++ -fopenmp -o main.o main.cpp
goarty@GoComp:~/Documents/paral_program/lab_3/src$ ./main.o
Потоки: 1       Время: 39.082
Потоки: 2       Время: 18.093
Потоки: 4       Время: 9.611
Потоки: 6       Время: 8.051
Потоки: 8       Время: 7.317
Потоки: 10      Время: 7.222
Потоки: 12      Время: 7.114
Потоки: 14      Время: 7.071
Потоки: 16      Время: 6.979
```