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 № 2

по курсу «Распределение параллельных и распределённых программ»

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

Студент группы ИУ9-51Б Горбунов А. Д.

Преподаватель Царёв А. С.

Москва 2024

1 Задача

1. Написать программу, которая реализует итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b в соответствии с выбранным вариантом. Здесь A – матрица размером N×N, x и b – векторы длины N. Тип элементов – double.

2. Программу распараллелить с помощью MPI с разрезанием матрицы A по строкам на близкие по размеру, возможно не одинаковые, части. Соседние строки матрицы должны располагаться в одном или в соседних MPI-процессах. Реализовать два варианта программы:

1: векторы x и b дублируются в каждом MPI-процессе,

2: векторы x и b разрезаются между MPI-процессами аналогично матрице A. (только для сдающих после срока)

Уделить внимание тому, чтобы при запуске программы на различном числе MPI-процессов решалась одна и та же задача (исходные данные заполнялись одинаковым образом).

3. Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: 1,2, 4, 8, 16. Построить графикизависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и E подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд. Также параметр N разрешено подобрать таким образом, чтобы он нацело делился на на 1,2,4,8 и 16.

2 Код решения

Файл main.cpp:

```
#include <iostream>
#include <mpi.h>
#include <chrono>
#include <cmath>

using namespace std;

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

void multMatrixes(double x[N], double y[N], short** A, int MIN, int MAX){
    for(int i = MIN; i < MAX; 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], int MIN, int MAX){
    for(int i = MIN; i < MAX; i++)
        z[i] = x[i] - y[i];
}

double scalMatrixes(double u[N], double v[N], int MIN, int MAX){
    double x = 0;
    for(int i = MIN; i < MAX; i++)
        x += u[i] * v[i];
    return x;
}

double powMatrix(double u[N]){
    double x = 0;
```

```
    for(int i = 0; i < N; i++)
        x += u[i] * u[i];
    x = sqrt(x);
    return x;
}

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

void fillMatrix(double x[N], double a, int MIN, int MAX){
    for (int i = MIN; i < MAX; i++)
        x[i] = a;
}

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

    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]{0};
    for(int i = 0; i < N; i++)
        z[i] = Ax_n[i] - b[i];

    double k = powMatrix(z)/powMatrix(b);
```

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

void mainProg(double size, double rank, int SREZ, int MIN, int MAX){
    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];

    for (int i = MIN; i < MAX; i++)
        for (int j = 0; j < N; j++)
            if (i == j)
                A[i][j] = 2;
            else
                A[i][j] = 1;

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

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

        double y_n[N];
        fillMatrix(y_n, 0, MIN, MAX);
```

```
subMatrixes(Ax_n, b, y_n, MIN, MAX);

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

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

MPI_Barrier(MPI_COMM_WORLD);
for(int i = MIN; i < MAX; i++)
    y_n[i] = r_n * y_n[i];

subMatrixes(x_n, y_n, x_n, MIN, MAX);

MPI_Barrier(MPI_COMM_WORLD);

double received_message;
double message;

if(rank == 0)
    for (int i = 1; i < size; i++)
        for (int j = i*SREZ; j < (i+1)*SREZ; j++)
        {
            MPI_Recv(&received_message, 1, MPI_DOUBLE, i, i,
            MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            x_n[j] = received_message;
        }
else
    for (int i = MIN; i < MAX; i++)
    {
        message = x_n[i];
        MPI_Send(&message, 1, MPI_DOUBLE, 0,
```

```
        rank, MPI_COMM_WORLD);
    }

MPI_Barrier(MPI_COMM_WORLD);

if(rank == 0)
    f = kritEnd_bool(x_n, b, A);

MPI_Barrier(MPI_COMM_WORLD);

int send[1]{f};

MPI_Bcast(&send, 1, MPI_INT, 0, MPI_COMM_WORLD);
f = send[0];

if(rank == 0)
    for (int i = 1; i < size; i++)
        for (int j = i*SREZ; j < (i+1)*SREZ; j++)
        {
            message = x_n[j];
            MPI_Send(&message, 1, MPI_DOUBLE, i,
            i, MPI_COMM_WORLD);
        }
else
    for (int i = MIN; i < MAX; i++)
    {
        MPI_Recv(&received_message, 1, MPI_DOUBLE, 0,
        rank, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        x_n[i] = received_message;
    }

MPI_Barrier(MPI_COMM_WORLD);
```

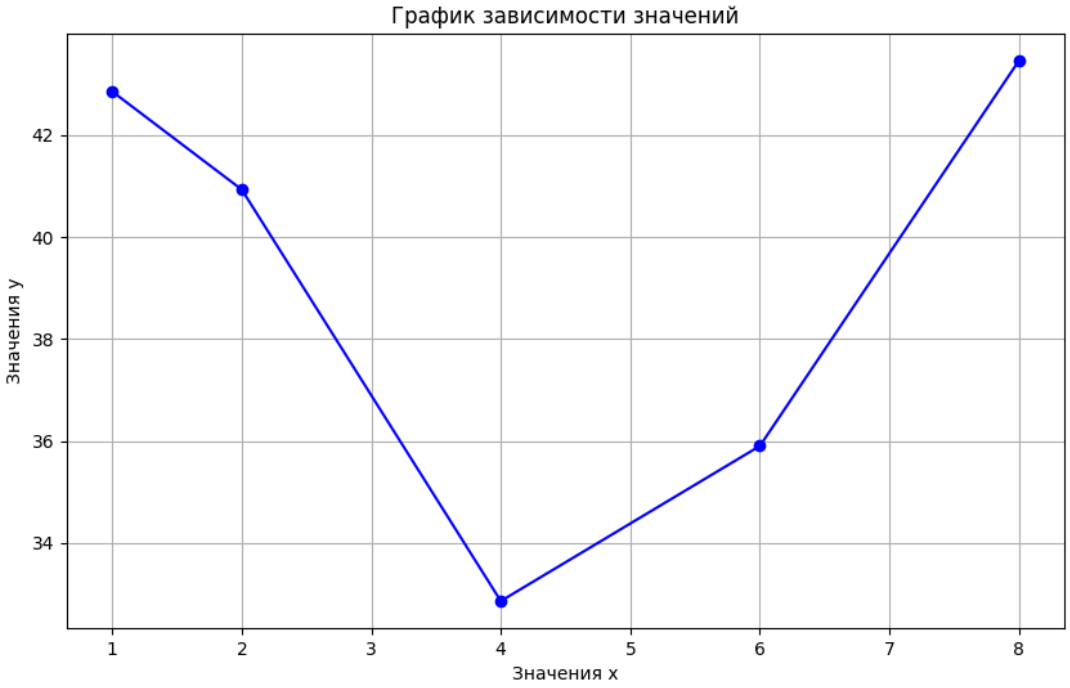
```
    }while(f != 0);

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

int main(int argc, char* argv[]) {
    MPI_Init(&argc, &argv);

    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int SREZ = N/size;
    int MIN = rank*SREZ;
    int MAX = (rank+1)*SREZ;

    auto start_time = std::chrono::steady_clock::now();

    mainProg(size, rank, SREZ, MIN, MAX);

    if(rank == 0)
    {
        auto current_time = std::chrono::steady_clock::now();
        auto elapsed_time = std::chrono::duration_cast<std::chrono::milliseconds>
        (current_time - start_time).count();
        printf("Время: ");
        cout << elapsed_time / 1000.0 << endl;
    }

    MPI_Finalize();
```

```
    return 0;
}
```

Файл start.sh:

```
#!/bin/bash

values=(1 2 4 6 8)

mpic++ -o main.o main.cpp

if [ $? -ne 0 ]; then
    echo "Компиляция не удалась"
    exit 1
fi

for n in "${values[@]}"; do
    echo "Запуск с n = $n"
    mpirun --oversubscribe -np $n main.o
done
```

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 MPI.

5 Результат запуска

```
goarty@GoComp:~/Documents/paral_program/lab_2/src$ ./start.sh
Запуск с n = 1
Время: 42.86
Запуск с n = 2
Время: 40.934
Запуск с n = 4
Время: 32.856
Запуск с n = 6
Время: 35.901
Запуск с n = 8
Время: 43.465
```