Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Пермский национальный исследовательский

политехнический университет»

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5**

по дисциплине **«Технологии блокчейн и распределенные информационные системы»**

**Выполнил** студент гр. АСУ-20-1б

Чувашев Максим Алексеевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись студента)*

**Проверил** доцент кафедры ИТАС

Щапов Владислав Алексеевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(оценка, дата, подпись преподавателя)*

Пермь, 2024

# 1 Анализ предметной области

## 1.1 Постановка задачи

С использованием языка C++ и библиотеки OpenMPI разработать программу для решения СЛАУ методом Гаусса.

## 1.2 Инструменты разработки

Для выполнения лабораторной работы были использованы следующие инструменты:

1. редактор CLion;

# 2 Технология разработки

Матрица, содержащая коэффициенты при неизвестных и свободные коэффициенты, инициализируется в 0 потоке.

// Инициализация матрицы случайными значениями

void init\_matrix(std::vector<double> &ab)

{

    srand(time(0));

    for (int i = 0; i < N \* (N + 1); ++i) {

        ab[i] = static\_cast<double>(rand() / (1000000));

    }

}

Листинг 1 — Инициализация метрицы

Далее матрица равномерно распределяется между потоками. С помощью «MPI\_Scatter». Далее следует прямой ход в методе Гаусса. Каждый процесс проходится по циклу начиная с 0 строки и заканчивая номером конечной строки блока, за который отвечает текущий процесс.

Далее определяем каким рангом выполняется цикл. И если этот процесс с данным рангом, должен обрабатывать именно эти строчки, то в данном процессе производится нормализация строк. Далее процесс отправляет нормализованные строки не в опорные процессы. Следом, происходит приведение к нулю нижележащих строк в данном блоке.

Если же ранг не является опорным, то есть условие не выполняется, то процесс получает нормализованную строку, и приводит к нулю соответсвующие строки своего блока.

    // Рассылка частей матрицы каждому процессу

    MPI\_Scatter(ab.data(), count\_rows\_rank \* (N + 1), MPI\_DOUBLE, ab\_local.data(), count\_rows\_rank \* (N + 1), MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

    // Прямой ход метода Гаусса

    for (int row = 0; row < end\_row; row++)

    {

        int selected\_rank = row / count\_rows\_rank;

        if (rank == selected\_rank)

        {

            int local\_row = row % count\_rows\_rank;

            double scale = ab\_local[local\_row \* (N + 1) + row];

            // Нормализация строки

            for (int col = row; col < N + 1; col++)

            {

                ab\_local[local\_row \* (N + 1) + col] /= scale;

            }

            // Рассылка нормализованной строки другим процессам

            for (int i = selected\_rank + 1; i < size; i++)

            {

                MPI\_Send(ab\_local.data() + (N + 1) \* local\_row, N + 1, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD);

            }

            // Приведение к нулю нижележащих строк

            for (int i = local\_row + 1; i < count\_rows\_rank; i++)

            {

                double op = ab\_local[i \* (N + 1) + row];

                for (int j = row; j < N + 1; j++)

                {

                    ab\_local[i \* (N + 1) + j] -= ab\_local[local\_row \* (N + 1) + j] \* op;

                }

            }

        }

        else

        {

            // Получение нормализованной строки от активного процесса

            MPI\_Recv(selected\_row.data(), N + 1, MPI\_DOUBLE, selected\_rank, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

            // Приведение к нулю соответствующих строк

            for (int i = 0; i < count\_rows\_rank; i++)

            {

                double op = ab\_local[i \* (N + 1) + row];

                for (int j = row; j < N + 1; j++)

                {

                    ab\_local[i \* (N + 1) + j] -= selected\_row[j] \* op;

                }

            }

        }

    }

Листинг 2 — Распределённая реализация «обратного хода»

Следом идет обратных ход. Где выполняется вычисление неизвестных снизу вверх, от последней строки к первой. Находится сумма, которая будет вычтена из свободного коэффициента, и найденный результат будет отправлен в остальные процесы.

    // Обратный ход метода Гаусса

    for (int row = N - 1; row >= 0; row--)

    {

        int selected\_rank = row / count\_rows\_rank;

        int local\_row = row % count\_rows\_rank;

        if (rank == selected\_rank)

        {

            double sum = 0;

            // Вычисление элементов решения

            for (int j = row + 1; j < N; j++)

            {

                sum += ab\_local[local\_row \* (N + 1) + j] \* x[j];

            }

            x[row] = ab\_local[local\_row \* (N + 1) + N] - sum;

        }

        // Рассылка найденных элементов решения

        MPI\_Bcast(x.data(), N, MPI\_DOUBLE, selected\_rank, MPI\_COMM\_WORLD);

    }

# 3 Результаты работы

Результаты выполнения задания лабораторной работы проиллюстрированы на рисунке 1.

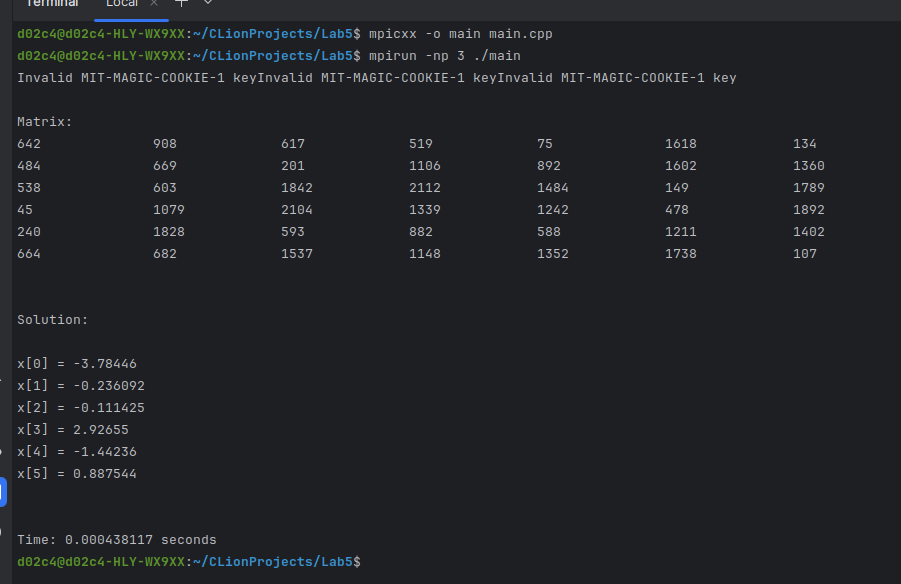


Рисунок 1 – Результат выполнения работы

ПРИЛОЖЕНИЕ А

#include <iostream>

#include <chrono>

#include <vector>

#include <mpi.h>

// Размер матрицы и вектора

const int N = 6;

// Инициализация матрицы случайными значениями

void init\_matrix(std::vector<double> &ab)

{

    srand(time(0));

    for (int i = 0; i < N \* (N + 1); ++i) {

        ab[i] = static\_cast<double>(rand() / (1000000));

    }

}

// Вывод матрицы

void print\_matrix(std::vector<double> ab)

{

    for(int i = 0; i < N; i++){

        for(int j = 0; j < N + 1; j++){

            std::cout << ab[i \* (N + 1) + j] << " \t \t ";

        }

        std::cout << std::endl;

    }

}

// Вывод вектора x

void print\_x(std::vector<double> x, std::vector<double> right\_x)

{

    std::cout << std::endl;

    for(int i = 0; i < N; i++){

        std::cout << "x[" << i << "] = ";

        std::cout << x[i] << std::endl;

    }

}

int main(int argc, char \*argv[])

{

    int size;

    int rank;

    double time;

    std::vector<double> ab(N \* (N + 1));

    std::vector<double> x(N);

    std::vector<double> x\_solution(N);

    // Инициализация MPI

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    if (rank == 0)

    {

        // Инициализация матрицы на корневом процессе

        init\_matrix(ab);

        // Начало отсчёта времени

        time = -MPI\_Wtime();

    }

    const int count\_rows\_rank = N / size;

    const int start\_row = rank \* count\_rows\_rank;

    const int end\_row = start\_row + count\_rows\_rank;

    std::vector<double> ab\_local(count\_rows\_rank \* (N + 1));

    std::vector<double> selected\_row(N + 1);

    // Рассылка частей матрицы каждому процессу

    MPI\_Scatter(ab.data(), count\_rows\_rank \* (N + 1), MPI\_DOUBLE, ab\_local.data(), count\_rows\_rank \* (N + 1), MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

    // Прямой ход метода Гаусса

    for (int row = 0; row < end\_row; row++)

    {

        int selected\_rank = row / count\_rows\_rank;

        if (rank == selected\_rank)

        {

            int local\_row = row % count\_rows\_rank;

            double scale = ab\_local[local\_row \* (N + 1) + row];

            // Нормализация строки

            for (int col = row; col < N + 1; col++)

            {

                ab\_local[local\_row \* (N + 1) + col] /= scale;

            }

            // Рассылка нормализованной строки другим процессам

            for (int i = selected\_rank + 1; i < size; i++)

            {

                MPI\_Send(ab\_local.data() + (N + 1) \* local\_row, N + 1, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD);

            }

            // Приведение к нулю нижележащих строк

            for (int i = local\_row + 1; i < count\_rows\_rank; i++)

            {

                double op = ab\_local[i \* (N + 1) + row];

                for (int j = row; j < N + 1; j++)

                {

                    ab\_local[i \* (N + 1) + j] -= ab\_local[local\_row \* (N + 1) + j] \* op;

                }

            }

        }

        else

        {

            // Получение нормализованной строки от активного процесса

            MPI\_Recv(selected\_row.data(), N + 1, MPI\_DOUBLE, selected\_rank, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

            // Приведение к нулю соответствующих строк

            for (int i = 0; i < count\_rows\_rank; i++)

            {

                double op = ab\_local[i \* (N + 1) + row];

                for (int j = row; j < N + 1; j++)

                {

                    ab\_local[i \* (N + 1) + j] -= selected\_row[j] \* op;

                }

            }

        }

    }

    // Обратный ход метода Гаусса

    for (int row = N - 1; row >= 0; row--)

    {

        int selected\_rank = row / count\_rows\_rank;

        int local\_row = row % count\_rows\_rank;

        if (rank == selected\_rank)

        {

            double sum = 0;

            // Вычисление элементов решения

            for (int j = row + 1; j < N; j++)

            {

                sum += ab\_local[local\_row \* (N + 1) + j] \* x[j];

            }

            x[row] = ab\_local[local\_row \* (N + 1) + N] - sum;

        }

        // Рассылка найденных элементов решения

        MPI\_Bcast(x.data(), N, MPI\_DOUBLE, selected\_rank, MPI\_COMM\_WORLD);

    }

    if (rank == 0)

    {

        // Вывод исходной матрицы

        std::cout << "\n\nMatrix:\n";

        print\_matrix(ab);

        // Вывод решения

        std::cout << "\n\nSolution:\n";

        print\_x(x, x\_solution);

        // Окончание отсчёта времени и вывод времени выполнения

        time += MPI\_Wtime();

        std::cout << "\n\nTime: " << time << " seconds\n";

    }

    // Завершение MPI

    MPI\_Finalize();

    return 0;

}