**Министерство науки и высшего образования Российской Федерации**
Калужский филиал
федерального государственного автономного
образовательного учреждения высшего образования
***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***
***(КФ МГТУ им. Н.Э. Баумана)***

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

# ЛАБОРАТОРНАЯ РАБОТА №5

## «Распараллеливание алгоритмов решения СЛАУ методом Гаусса»

**ДИСЦИПЛИНА: «Параллельные процессы в информационных системах»**

Выполнил: студент гр. ИУК4-31М _____ (Сафронов Н.С.)
(Подпись) (Ф.И.О.)

Проверил: _____ (Корнюшин Ю.П.)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2025

**Цель**: формирование практических навыков распараллеливания алгоритмов решения систем линейных алгебраических уравнений прямыми методами на языке MPI.

**Задачи**

1. Освоить два способа параллельного решения СЛАУ методом Гаусса, связанных с разными способами распределения данных по компьютерам.
2. Разработать параллельный алгоритм, написать и отладить параллельную программу решения СЛАУ методом Гаусса в MPI.
3. Сравнить временные характеристики двух алгоритмов.

**Задание**

**Вариант 6**

Для реализации параллельной программы должен использоваться язык программирования OpenMP.

**Задание 1**

Проработка примеров этой же лабораторной.

Внимательно изучить примеры 1-2. Откомпилировать и запустить на 1-м процессоре.

**Задание 2**

Разработать параллельный алгоритм № 1, написать и отладить параллельную программу решения СЛАУ методом Гаусса в MPI. В данном алгоритме предполагается, что матрица коэффициентов распределена по компьютерам горизонтальными полосами. Варианты СЛАУ приведены в таблице 1.

**Задание 3**

Разработать параллельный алгоритм № 2, написать и отладить параллельную программу решения СЛАУ методом Гаусса в MPI. В данном алгоритме предполагается, что матрица коэффициентов распределена по компьютерам циклически горизонтальными полосами.

| Вариант | Матрицы | Вектор |
|---|---|---|
| 6 | [300 x 300]<br>[530 x 530] | 300<br>530 |

| | | |
|---|---|---|
| | [710 x 710]<br>[1030 x 1030] | 710<br>1030 |

## Результат выполнения работы

### Задание 1

```
PS E:\Dev\bmstu-magistracy\term_3\parallel_computing\labs\lab_5> mpiexec -n 1 .\bin\task_1.exe
Time = 1041
 1.000000 1.000000 1.000000 1.000000
 1.000000 1.000000 1.000000 1.000000
```

**Рисунок 1** – Результат выполнения примера 1

### Задание 2

```
PS E:\Dev\bmstu-magistracy\term_3\parallel_computing\labs\lab_5> mpiexec -n 8 .\bin\task_2.exe 8
Solution x:
x[0] = 2.000000
x[1] = 1.000000
x[2] = 1.000000
x[3] = 1.000000
x[4] = 1.000000
x[5] = 1.000000
x[6] = 1.000000
x[7] = 1.000000
Total execution time: 0.003925 seconds
```

**Рисунок 2** – Результат выполнения задания 2

### Листинг

```
#include <stdio.h>
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char** argv) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (argc < 2) {
        if (rank == 0) {
            fprintf(stderr, "Usage: %s <matrix_size>\n", argv[0]);
        }
        MPI_Finalize();
        return 1;
    }

    int n = atoi(argv[1]);
    if (n <= 0) {
        if (rank == 0) {
            fprintf(stderr, "Error: matrix size must be a positive integer.\n");
        }
```

```
        MPI_Finalize();
        return 1;
    }

    if (n % size != 0) {
        if (rank == 0)
            printf("Error: n (%d) must be divisible by number of processes
(%d)\n", n, size);
        MPI_Finalize();
        return 1;
    }

    int local_n = n / size;
    double *A = (double*)malloc(local_n * n * sizeof(double));
    double *b = (double*)malloc(local_n * sizeof(double));
    double *x = (double*)calloc(n, sizeof(double));

    for (int i = 0; i < local_n; i++) {
        int global_i = rank * local_n + i;
        for (int j = 0; j < n; j++) {
            A[i * n + j] = 0.0;
        }
        if (global_i == 0) {
            A[i * n + 0] = 4.0;
            A[i * n + 1] = 1.0;
            b[i] = 9.0;
        } else if (global_i == n - 1) {
            A[i * n + n - 2] = 1.0;
            A[i * n + n - 1] = 4.0;
            b[i] = 5.0;
        } else {
            A[i * n + global_i - 1] = 1.0;
            A[i * n + global_i    ] = 4.0;
            A[i * n + global_i + 1] = 1.0;
            b[i] = (global_i == 1) ? 7.0 : 6.0;
        }
    }

    double start_time = MPI_Wtime();
    for (int k = 0; k < n; k++) {
        int owner = k / local_n;

        if (rank == owner) {
            int local_k = k % local_n;
            double pivot = A[local_k * n + k];
            for (int j = k; j < n; j++) {
                A[local_k * n + j] /= pivot;
            }
            b[local_k] /= pivot;

            MPI_Bcast(&A[local_k * n + k], n - k, MPI_DOUBLE, owner,
MPI_COMM_WORLD);
            MPI_Bcast(&b[local_k], 1, MPI_DOUBLE, owner, MPI_COMM_WORLD);
        } else {
            double *pivot_row = (double*)malloc((n - k) * sizeof(double));
            double pivot_b;
            MPI_Bcast(pivot_row, n - k, MPI_DOUBLE, owner, MPI_COMM_WORLD);
            MPI_Bcast(&pivot_b, 1, MPI_DOUBLE, owner, MPI_COMM_WORLD);

            for (int i = 0; i < local_n; i++) {
                int global_i = rank * local_n + i;
                if (global_i <= k) continue;
                double factor = A[i * n + k];
                for (int j = k; j < n; j++) {
                    A[i * n + j] -= factor * pivot_row[j - k];
                }
```

```
                b[i] -= factor * pivot_b;
            }
            free(pivot_row);
        }
    }

    for (int k = n - 1; k >= 0; k--) {
        int owner = k / local_n;
        double xk;

        if (rank == owner) {
            int local_k = k % local_n;
            double sum = b[local_k];
            for (int j = k + 1; j < n; j++) {
                sum -= A[local_k * n + j] * x[j];
            }
            xk = sum;
            x[k] = xk;
        }

        MPI_Bcast(&xk, 1, MPI_DOUBLE, owner, MPI_COMM_WORLD);
        if (rank != owner) {
            x[k] = xk;
        }
    }

    double end_time = MPI_Wtime();
    double total_time = end_time - start_time;

    if (rank == 0) {
        printf("Solution x[:8]:\n");
        for (int i = 0; i < 8; i++) {
            printf("x[%d] = %.6f\n", i, x[i]);
        }
        printf("Total execution time: %.6f seconds\n", total_time);
    }

    free(A);
    free(b);
    free(x);
    MPI_Finalize();
    return 0;
}
```

## Задание 3

```
PS E:\Dev\bmstu-magistracy\term_3\parallel_computing\labs\lab_5> mpiexec -n 8 .\bin\task_3.exe 8
Solution x[:8]:
x[0] = 2.000000
x[1] = 1.000000
x[2] = 1.000000
x[3] = 1.000000
x[4] = 1.000000
x[5] = 1.000000
x[6] = 1.000000
x[7] = 1.000000
Total execution time: 0.007306 seconds
```

**Рисунок 3** – Результат выполнения задания 3

## Листинг

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


int main(int argc, char** argv) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (argc < 2) {
        if (rank == 0) {
            fprintf(stderr, "Usage: %s <matrix_size>\n", argv[0]);
        }
        MPI_Finalize();
        return 1;
    }

    int n = atoi(argv[1]);
    if (n <= 0) {
        if (rank == 0) {
            fprintf(stderr, "Error: matrix size must be a positive integer.\n");
        }
        MPI_Finalize();
        return 1;
    }

    int local_n = (n / size) + ((rank < n % size) ? 1 : 0);

    double *A = (double*)malloc(local_n * n * sizeof(double));
    double *b = (double*)malloc(local_n * sizeof(double));
    double *x = (double*)calloc(n, sizeof(double));

    int idx = 0;
    for (int global_i = 0; global_i < n; global_i++) {
        if (global_i % size != rank) continue;

        for (int j = 0; j < n; j++) {
            A[idx * n + j] = 0.0;
        }

        if (global_i == 0) {
            A[idx * n + 0] = 4.0;
            A[idx * n + 1] = 1.0;
            b[idx] = 9.0;
        } else if (global_i == n - 1) {
            A[idx * n + n - 2] = 1.0;
            A[idx * n + n - 1] = 4.0;
            b[idx] = 5.0;
        } else {
            A[idx * n + global_i - 1] = 1.0;
            A[idx * n + global_i    ] = 4.0;
            A[idx * n + global_i + 1] = 1.0;
            b[idx] = (global_i == 1) ? 7.0 : 6.0;
        }
        idx++;
    }

    double start_time = MPI_Wtime();
    for (int k = 0; k < n; k++) {
        int owner = k % size;
        int local_k = k / size;
```

```
        if (rank == owner) {
            if (local_k >= local_n) {
                MPI_Finalize();
                return 1;
            }

            double pivot = A[local_k * n + k];
            for (int j = k; j < n; j++) {
                A[local_k * n + j] /= pivot;
            }
            b[local_k] /= pivot;

            for (int p = 0; p < size; p++) {
                if (p == owner) continue;
                MPI_Send(&A[local_k * n + k], n - k, MPI_DOUBLE, p, 0,
MPI_COMM_WORLD);
                MPI_Send(&b[local_k], 1, MPI_DOUBLE, p, 0, MPI_COMM_WORLD);
            }
        } else {
            double *pivot_row = (double*)malloc((n - k) * sizeof(double));
            double pivot_b;
            MPI_Recv(pivot_row, n - k, MPI_DOUBLE, owner, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            MPI_Recv(&pivot_b, 1, MPI_DOUBLE, owner, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

            for (int i = 0; i < local_n; i++) {
                int global_i = (i * size + rank);
                if (global_i <= k) continue;

                double factor = A[i * n + k];
                for (int j = k; j < n; j++) {
                    A[i * n + j] -= factor * pivot_row[j - k];
                }
                b[i] -= factor * pivot_b;
            }
            free(pivot_row);
        }
    }

    for (int k = n - 1; k >= 0; k--) {
        int owner = k % size;
        int local_k = k / size;

        if (rank == owner) {
            if (local_k >= local_n) {
                MPI_Finalize();
                return 1;
            }

            double sum = b[local_k];
            for (int j = k + 1; j < n; j++) {
                sum -= A[local_k * n + j] * x[j];
            }
            x[k] = sum;

            for (int p = 0; p < size; p++) {
                if (p == owner) continue;
                MPI_Send(&x[k], 1, MPI_DOUBLE, p, 0, MPI_COMM_WORLD);
            }
        } else {
            MPI_Recv(&x[k], 1, MPI_DOUBLE, owner, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        }
    }
```

```
    double end_time = MPI_Wtime();
    double total_time = end_time - start_time;

    if (rank == 0) {
        printf("Solution x[:8]:\n");
        for (int i = 0; i < 8; i++) {
            printf("x[%d] = %.6f\n", i, x[i]);
        }
        printf("Total execution time: %.6f seconds\n", total_time);
    }

    free(A);
    free(b);
    free(x);
    MPI_Finalize();
    return 0;
}
```

## Задание 4

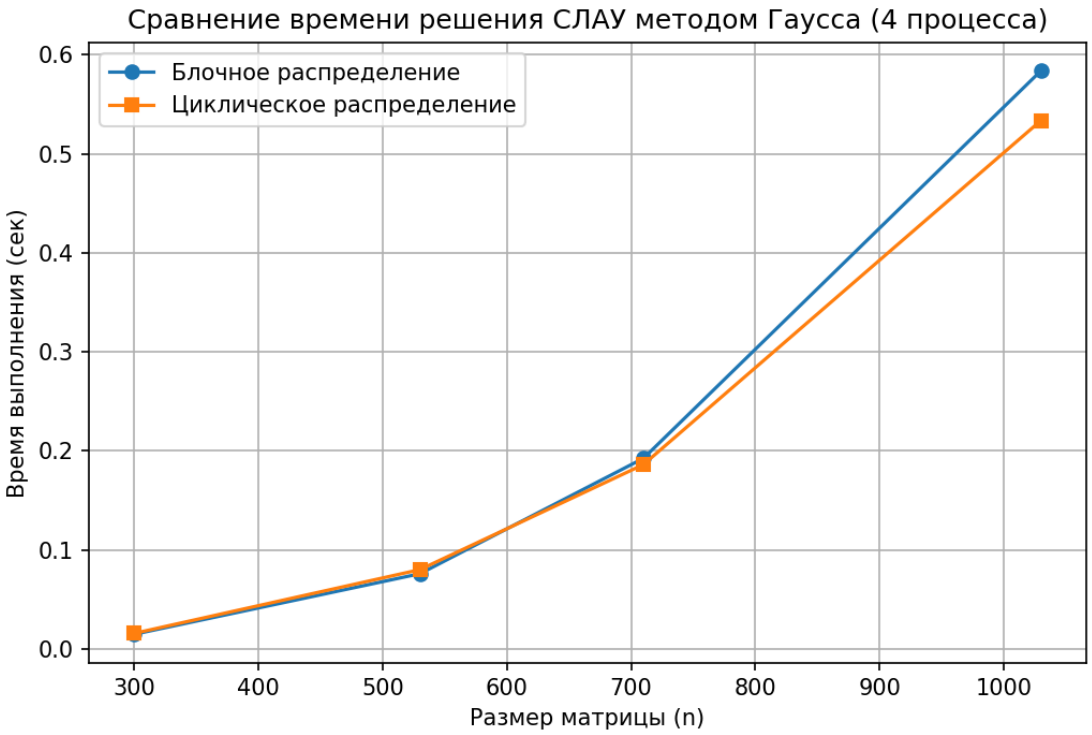


**Рисунок 4** – Результат выполнения задания 4

**Вывод**: в ходе выполнения лабораторной работы были сформированы практические навыки распараллеливания алгоритмов решения систем линейных алгебраических уравнений прямыми методами на языке MPI.