



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 №2**

**«Параллельные программы умножения матрицы на вектор»**

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

Выполнил: студент гр. ИУК4-31М

\_\_\_\_\_  
(Подпись)

(Сафронов Н.С.)  
(Ф.И.О.)

Проверил:

\_\_\_\_\_  
(Подпись)

(Корнюшин Ю.П.)  
(Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:

- Оценка:

Калуга, 2025

**Цель:** формирование практических навыков программирования параллельных процессов, исполняющихся на декартовой топологии связей и топологии "граф".

### **Задачи**

1. Освоить функции задания декартовой топологии и топологии "граф".
2. Освоить методы распараллеливания алгоритмов решения задач, таких как умножение матрицы на вектор.

### **Задание**

#### **Вариант 6**

Программа должна быть реализована на языке программирования C++ с использованием технологии MPI.

#### **Задание 1**

Проработка примеров из пункта "Примеры параллельных программ" этой же лабораторной.

Внимательно изучить примеры 1, 2, 3. Примеры 2 и 3 (Умножение матрицы на вектор на топологиях "кольцо" и "полный граф") откомпилировать и запустить на 4-х компьютерах.

#### **Задание 2**

Параллельное умножение матрицы на вектор на топологии "кольцо".

Разработать алгоритм, написать и отладить параллельную программу умножения матрицы на вектор в топологии "кольцо":  $A \times B = C$ , при условии, что количество строк матрицы и элементов вектора нацело делится на количество компьютеров. Например, матрица A размером [20x20] и вектор B размером [20] на четырех компьютерах.

#### **Задание 3**

Параллельное умножение матрицы на матрицу на топологии "кольцо".

Разработать алгоритм написать и отладить параллельную программу умножения матрицы на матрицу в топологии "кольцо":  $A \times B = C$ , при условии, что количество строк матрицы A и столбцов матрицы B нацело не делится на количество компьютеров. Например, матрица A размером [31x20] и матрица B размером [20x31] на четырех компьютерах.

#### Задание 4

Сравнительные временные характеристики двух алгоритмов умножения матрицы на вектор на топологиях "кольцо" и "полный граф".

В качестве параллельных программ взять примеры 2 и 3 (Умножение матрицы на вектор на топологиях "кольцо" и "полный граф"). Для обеих программ построить небольшие графики зависимости времени решения задачи от размеров матрицы и вектора. Обе программы запустить последовательно для матриц и соответствующих этим матрицам векторов:

Вариант	Матрицы	Вектор
6	[300 x 300]	300
	[530 x 530]	530
	[710 x 710]	710
	[1030 x 1030]	1030

#### Результат выполнения работы

##### Задание 1

```
PS E:\Dev\bmstu-magistracy\3rd-term\parallel-processing\lab2> mpiexec -n 4 .\bin\example2.exe
rank = 0 Time = 0.001187
rank = 0 RM = 120.00
rank = 0 RM = 120.00
rank = 0 RM = 120.00
rank = 0 RM = 120.00
rank = 0 RM = 120.00
rank = 2 Time = 0.000829
rank = 2 RM = 120.00
rank = 2 RM = 120.00
rank = 2 RM = 120.00
rank = 2 RM = 120.00
rank = 2 RM = 120.00
rank = 3 Time = 0.000556
rank = 3 RM = 120.00
rank = 3 RM = 120.00
rank = 3 RM = 120.00
rank = 3 RM = 120.00
rank = 3 RM = 120.00
rank = 1 Time = 0.000862
rank = 1 RM = 120.00
rank = 1 RM = 120.00
rank = 1 RM = 120.00
rank = 1 RM = 120.00
rank = 1 RM = 120.00
```

Рисунок 1 – Результат выполнения примера 2

```

PS E:\Dev\bmstu-magistracy\3rd-term\parallel-processing\lab2> mpiexec -n 4 .\bin\example3.exe
rank = 1 Time = 0.000614
rank = 2 Time = 0.000654
rank = 3 Time = 0.000415
rank = 0 Time = 0.000967
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00
B = 120.00

```

**Рисунок 2** – Результат выполнения примера 3

## Задание 2

```

PS E:\Dev\bmstu-magistracy\3rd-term\parallel-processing\lab2> mpiexec -n 4 .\bin\task2.exe
rank = 1: time = 0.000895 s
rank = 2: time = 0.000873 s
rank = 3: time = 0.000569 s
rank = 0: time = 0.001151 s

C = A * B (first 10 elements):
C[0] = 100.00
C[1] = 100.00
C[2] = 100.00
C[3] = 100.00
C[4] = 100.00
C[5] = 100.00
C[6] = 100.00
C[7] = 100.00
C[8] = 100.00
C[9] = 100.00

```

**Рисунок 3** – Параллельное умножение матрицы на вектор на топологии "кольцо" А размером [20x20] и вектор В размером [20] на четырех компьютерах

## Листинг

```

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

#define ROWS 20
#define COLS 20

int main(int argc, char **argv)

```

```

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

    if (ROWS % size != 0) {
        if (rank == 0)
            fprintf(stderr, "Error: ROWS (%d) can't be divided by size (%d)
returning whole number\n", ROWS, size);
        MPI_Finalize();
        return 1;
    }

    const int local_rows = ROWS / size;
    const int block = COLS / size;

    double *A_local = (double *)malloc(local_rows * COLS * sizeof(double));
    double *B_block = (double *)malloc(block * sizeof(double));
    double *C_local = (double *)calloc(local_rows, sizeof(double));

    for (int i = 0; i < local_rows; i++) {
        for (int j = 0; j < COLS; j++) {
            A_local[i * COLS + j] = 1.0;
        }
    }

    for (int i = 0; i < local_rows; i++) {
        for (int j = 0; j < COLS; j++) {
            C_local[i] += 3.0;
        }
    }
    for (int j = 0; j < block; j++) {
        B_block[j] = 2.0;
    }

    // Создание топологии "кольцо"
    int dims[1] = {0};
    int periods[1] = {1}; // периодическое кольцо
    MPI_Dims_create(size, 1, dims);
    MPI_Comm ring_comm;
    MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, 0, &ring_comm);

    int src, dst;
    MPI_Cart_shift(ring_comm, 0, 1, &src, &dst);

    double start_time = MPI_Wtime();

    for (int step = 0; step < size; step++) {
        int col_offset = ((rank - step + size) % size) * block;

        // Накопление: C_local += A_local[:, col_offset : col_offset+block] *
B_block
        for (int i = 0; i < local_rows; i++) {
            for (int j = 0; j < block; j++) {
                C_local[i] += A_local[i * COLS + col_offset + j] * B_block[j];
            }
        }

        // Циклический сдвиг блока вектора по кольцу
        if (size > 1) {
            MPI_Sendrecv_replace(B_block, block, MPI_DOUBLE,
                                dst, 0, src, 0, ring_comm, MPI_STATUS_IGNORE);
        }
    }
}

```

```

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

// Сбор полного результата на процессе 0
double *C_full = NULL;
if (rank == 0) {
    C_full = (double *)malloc(ROWS * sizeof(double));
}

MPI_Gather(C_local, local_rows, MPI_DOUBLE,
          C_full, local_rows, MPI_DOUBLE,
          0, MPI_COMM_WORLD);

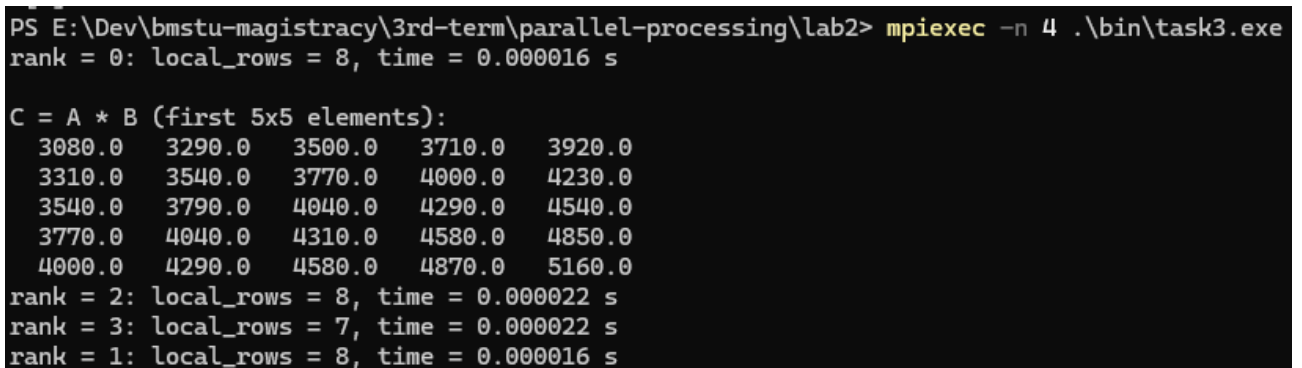
printf("rank = %d: time = %.6f s\n", rank, elapsed);

if (rank == 0) {
    printf("\nC = A * B (first 10 elements):\n");
    for (int i = 0; i < (ROWS < 10 ? ROWS : 10); i++) {
        printf("C[%d] = %.2f\n", i, C_full[i]);
    }
    free(C_full);
}

free(A_local);
free(B_block);
free(C_local);
MPI_Comm_free(&ring_comm);
MPI_Finalize();
return 0;
}

```

### Задание 3



```

PS E:\Dev\bmstu-magistracy\3rd-term\parallel-processing\lab2> mpiexec -n 4 .\bin\task3.exe
rank = 0: local_rows = 8, time = 0.000016 s

C = A * B (first 5x5 elements):
3080.0  3290.0  3500.0  3710.0  3920.0
3310.0  3540.0  3770.0  4000.0  4230.0
3540.0  3790.0  4040.0  4290.0  4540.0
3770.0  4040.0  4310.0  4580.0  4850.0
4000.0  4290.0  4580.0  4870.0  5160.0
rank = 2: local_rows = 8, time = 0.000022 s
rank = 3: local_rows = 7, time = 0.000022 s
rank = 1: local_rows = 8, time = 0.000016 s

```

**Рисунок 4** – Параллельное умножение матрицы на матрицу на топологии "кольцо" А размером [31x20] и матрица В размером [20x31] на четырех компьютерах

### Листинг

```

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

#define ROWS_A 31
#define COLS_A 20
#define COLS_B 31

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

```

```

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

// Создание топологии "кольцо" (обязательно по условию)
int dims[1] = {0};
int periods[1] = {1};
MPI_Dims_create(size, 1, dims);
MPI_Comm ring_comm;
MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, 0, &ring_comm);

// Неравномерное распределение строк A
int local_rows = ROWS_A / size;
int remainder = ROWS_A % size;
if (rank < remainder) {
    local_rows++; // первые 'остаток' процессов получают +1 строку
}

// Смещение начала строк для текущего процесса
int offset = rank * (ROWS_A / size) + (rank < remainder ? rank : remainder);

double *A_local = (double *)malloc(local_rows * COLS_A * sizeof(double));
double *B = (double *)malloc(COLS_A * COLS_B * sizeof(double));
double *C_local = (double *)calloc(local_rows * COLS_B, sizeof(double));

// Инициализация данных
for (int i = 0; i < local_rows; i++) {
    for (int j = 0; j < COLS_A; j++) {
        A_local[i * COLS_A + j] = (double)(offset + i + j + 1);
    }
}

for (int i = 0; i < COLS_A; i++) {
    for (int j = 0; j < COLS_B; j++) {
        B[i * COLS_B + j] = (double)(i + j + 2);
    }
}

// Локальное умножение
double start_time = MPI_Wtime();

for (int i = 0; i < local_rows; i++) {
    for (int k = 0; k < COLS_A; k++) {
        double a_ik = A_local[i * COLS_A + k];
        for (int j = 0; j < COLS_B; j++) {
            C_local[i * COLS_B + j] += a_ik * B[k * COLS_B + j];
        }
    }
}

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

int *recvcounts = NULL;
int *displs = NULL;
double *C_full = NULL;

if (rank == 0) {
    recvcounts = (int *)malloc(size * sizeof(int));
    displs = (int *)malloc(size * sizeof(int));
    C_full = (double *)malloc(ROWS_A * COLS_B * sizeof(double));

    int total = 0;
    for (int r = 0; r < size; r++) {
        int lr = ROWS_A / size + (r < (ROWS_A % size) ? 1 : 0);
        recvcounts[r] = lr * COLS_B;
        displs[r] = total;
    }
}

```

```

        total += recvcnts[r];
    }
}

int my_count = local_rows * COLS_B;
MPI_Gatherv(C_local, my_count, MPI_DOUBLE,
            C_full, recvcnts, displs, MPI_DOUBLE,
            0, MPI_COMM_WORLD);

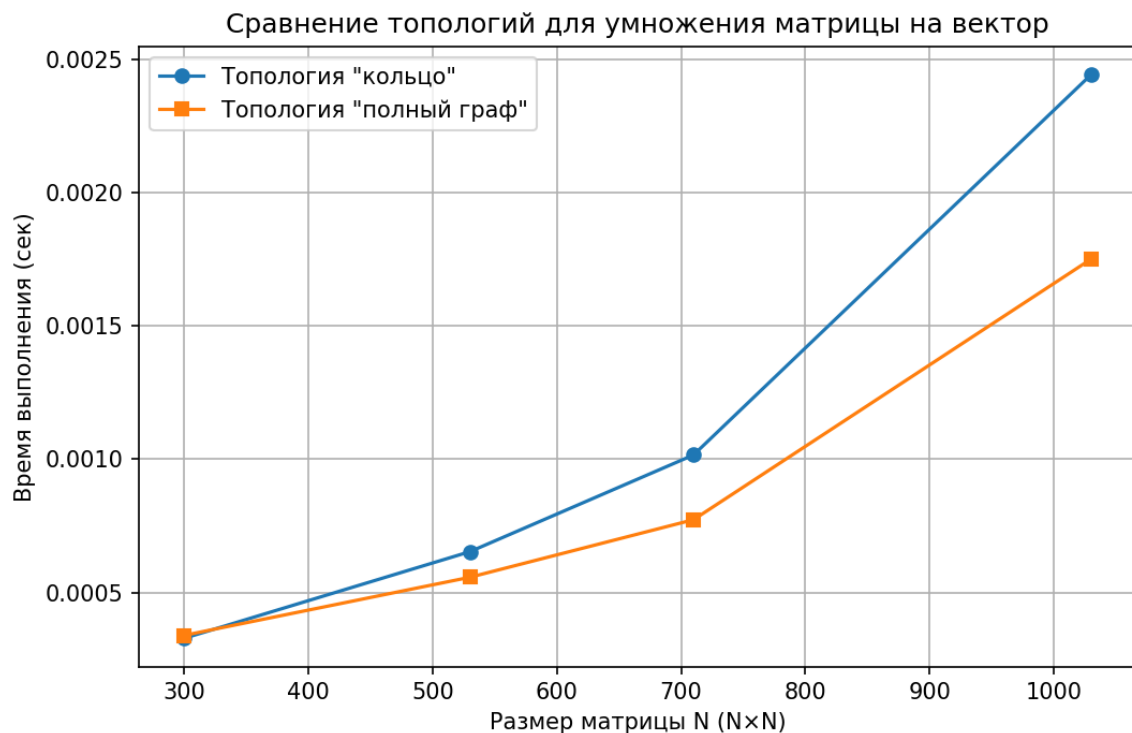
printf("rank = %d: local_rows = %d, time = %.6f s\n", rank, local_rows,
elapsed);

if (rank == 0) {
    printf("\nC = A * B (first 5x5 elements):\n");
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            printf("%8.1f ", C_full[i * COLS_B + j]);
        }
        printf("\n");
    }
    free(recvcnts);
    free(displs);
    free(C_full);
}

free(A_local);
free(B);
free(C_local);
MPI_Comm_free(&ring_comm);
MPI_Finalize();
return 0;
}

```

## Задание 4



**Рисунок 5** – Сравнение топологий для умножения матрицы на вектор



## Листинг программы топологии «кольцо»

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

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

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

    int N = atoi(argv[1]);

    if (N % size != 0) {
        if (rank == 0) fprintf(stderr, "N must be divisible by number of
processes!\n");
        MPI_Finalize();
        return 1;
    }

    const int local_rows = N / size;
    const int block = N / size;

    double *A_local = (double*)malloc(local_rows * N * sizeof(double));
    double *B_block = (double*)malloc(block * sizeof(double));
    double *C_local = (double*)calloc(local_rows, sizeof(double));

    for (int i = 0; i < local_rows; i++)
        for (int j = 0; j < N; j++)
            A_local[i * N + j] = 3.0;

    for (int j = 0; j < block; j++)
        B_block[j] = 2.0;

    int dims[1] = {0};
    int periods[1] = {1};
    MPI_Dims_create(size, 1, dims);
    MPI_Comm ring;
    MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, 0, &ring);

    int src, dst;
    MPI_Cart_shift(ring, 0, 1, &src, &dst);

    MPI_Barrier(MPI_COMM_WORLD);
    double t1 = MPI_Wtime();

    for (int step = 0; step < size; step++) {
        int col_offset = ((rank - step + size) % size) * block;
        for (int i = 0; i < local_rows; i++) {
            for (int j = 0; j < block; j++) {
                C_local[i] += A_local[i * N + col_offset + j] * B_block[j];
            }
        }
        if (size > 1) {
            MPI_Sendrecv_replace(B_block, block, MPI_DOUBLE, dst, 0, src, 0,
```

```

ring, MPI_STATUS_IGNORE);
    }
}

double t2 = MPI_Wtime();
double local_time = t2 - t1;

double max_time;
MPI_Reduce(&local_time, &max_time, 1, MPI_DOUBLE, MPI_MAX, 0,
MPI_COMM_WORLD);

if (rank == 0) {
    printf("ring %d %.6f\n", N, max_time);
}

free(A_local);
free(B_block);
free(C_local);
MPI_Comm_free(&ring);
MPI_Finalize();
return 0;
}

```

### Листинг программы топологии «полный граф»

```

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

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

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

    int N = atoi(argv[1]);

    if (N % size != 0) {
        if (rank == 0) fprintf(stderr, "N must be divisible by number of
processes!\n");
        MPI_Finalize();
        return 1;
    }

    const int local_rows = N / size;

    double *A_local = (double*)malloc(local_rows * N * sizeof(double));
    double *B = (double*)malloc(N * sizeof(double));
    double *C_local = (double*)calloc(local_rows, sizeof(double));

    for (int i = 0; i < local_rows; i++)
        for (int j = 0; j < N; j++)
            A_local[i * N + j] = 1.0;

    for (int j = 0; j < N; j++)
        B[j] = 2.0;

    int *index = (int*)malloc(size * sizeof(int));
    int *edges = (int*)malloc(size * (size - 1) * sizeof(int));

```

```

int pos = 0;
for (int i = 0; i < size; i++) {
    index[i] = size - 1;
    for (int j = 0; j < size; j++) {
        if (i != j) edges[pos++] = j;
    }
}

MPI_Comm graph_comm;
MPI_Graph_create(MPI_COMM_WORLD, size, index, edges, 0, &graph_comm);
MPI_Comm_rank(graph_comm, &rank);

MPI_Barrier(MPI_COMM_WORLD);
double t1 = MPI_Wtime();

for (int i = 0; i < local_rows; i++) {
    C_local[i] = 0.0;
    for (int j = 0; j < N; j++) {
        C_local[i] += A_local[i * N + j] * B[j];
    }
}

MPI_Allgather(C_local, local_rows, MPI_DOUBLE, B, local_rows, MPI_DOUBLE,
graph_comm);

double t2 = MPI_Wtime();
double local_time = t2 - t1;

double max_time;
MPI_Reduce(&local_time, &max_time, 1, MPI_DOUBLE, MPI_MAX, 0,
MPI_COMM_WORLD);

if (rank == 0) {
    printf("graph %d %.6f\n", N, max_time);
}

free(index);
free(edges);
free(A_local);
free(B);
free(C_local);
MPI_Comm_free(&graph_comm);
MPI_Finalize();
return 0;
}

```

**Вывод:** в ходе выполнения лабораторной работы были сформированы практические навыки программирования параллельных процессов, исполняющихся на декартовой топологии связей и топологии "граф".