МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра вычислительной техники

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Параллельные вычисления»
Тема: «Коллективные функции»

Студент гр. 1307 _____ Голубев М.А.

Преподаватель _____ Манжиков Л.П.

Санкт-Петербург
2025

Цель работы.

Освоить функции коллективной обработки данных.

Задание 1 (по вариантам).

Решить задание 1 из лаб. работы 2 с применением коллективных функций.

Задание 2 (по вариантам).

В полученной матрице (по результатам выполнения задания 1) найти:

Решить задание 1 или 2 из лаб. работы 3 с применением коллективных функций.

Задание 2 из лаб. работы №2 - создать n процессов и найти минимальный положительный элемент массива.

Текст программы lab4_1.cpp.

```
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <time.h>
#include "mpi.h"

#define DEFAULT_ARR_SIZE 10

int main(int argc, char **argv)
{
   int rank, size;
   int local_min = INT_MAX;
   int global_min = INT_MAX;
   int array_size = 20;

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

   if (rank == 0)
   {
       if (argc > 1)
       {
           array_size = atoi(argv[1]);
```

```
            if (array_size <= 0)
            {
                printf("Размер массива не задан, используется размер по умолчанию
(20).\n");
                array_size = DEFAULT_ARR_SIZE;
            }
        }
        printf("Используемый размер массива: %d\n", array_size);
    }

    MPI_Bcast(&array_size, 1, MPI_INT, 0, MPI_COMM_WORLD);

    int *array = NULL;
    if (rank == 0)
    {
        array = (int *)malloc(array_size * sizeof(int));
        srand(time(NULL));
        printf("Массив: ");
        for (int i = 0; i < array_size; ++i)
        {
            array[i] = rand() % 100 - 50;
            printf("%d ", array[i]);
        }
        printf("\n");
    }

    int local_size = array_size / size;
    int *local_array = (int *)malloc(local_size * sizeof(int));

    MPI_Scatter(array, local_size, MPI_INT, local_array, local_size, MPI_INT, 0,
MPI_COMM_WORLD);

    for (int i = 0; i < local_size; ++i)
    {
        if (local_array[i] > 0 && local_array[i] < local_min)
        {
            local_min = local_array[i];
        }
    }

    MPI_Reduce(&local_min, &global_min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
```

```
    if (rank == 0)
    {
        if (global_min != INT_MAX)
        {
            printf("Минимальный положительный элемент: %d\n", global_min);
        }
        else
        {
            printf("В массиве нет положительных элементов.\n");
        }
        free(array);
    }

    free(local_array);
    MPI_Finalize();
    return 0;
}
```

```
mishhgun@MacBook-Air-Mihail lab2 % docker run mpich-2-2
Используемый размер массива: 20
Массив: -7 21 -36 -21 -20 -22 28 -49 -8 36 -39 47 46 41 0 -42 -49 17 36 -2
Минимальный положительный элемент: 21
```

Рисунок 1. Запуск программы на 8-ми процессах.

Задание 1 из лаб. работы №3 - в прямоугольной матрице, состоящей из 0 и 1 инвертировать четные столбцы.

Текст программы lab4_2.cpp.

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

#define ROWS 6
#define COLS 4

void print_matrix(int matrix[ROWS][COLS], int rows, int cols)
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            printf("%d ", matrix[i][j]);
```

```
        }
        printf("\n");
    }
}

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);
    int matrix[ROWS][COLS];
    int local_rows = ROWS / (size - 1);
    int local_matrix[local_rows][COLS];

    if (rank == 0)
    {
        printf("Исходная матрица:\n");
        for (int i = 0; i < ROWS; i++)
        {
            for (int j = 0; j < COLS; j++)
            {
                matrix[i][j] = rand() % 2;
            }
        }
        print_matrix(matrix, ROWS, COLS);
    }

    MPI_Scatter(matrix, local_rows * COLS, MPI_INT, local_matrix, local_rows *
COLS, MPI_INT, 0, MPI_COMM_WORLD);

    for (int i = 0; i < local_rows; i++)
    {
        for (int j = 1; j < COLS; j += 2)
        {
            local_matrix[i][j] = 1 - local_matrix[i][j];
        }
    }

    MPI_Gather(local_matrix, local_rows * COLS, MPI_INT, matrix, local_rows * COLS,
MPI_INT, 0, MPI_COMM_WORLD);
```

```
    if (rank == 0)
    {
        printf("Матрица после инверсии четных столбцов:\n");
        print_matrix(matrix, ROWS, COLS);
    }

    MPI_Finalize();
    return 0;
}
```

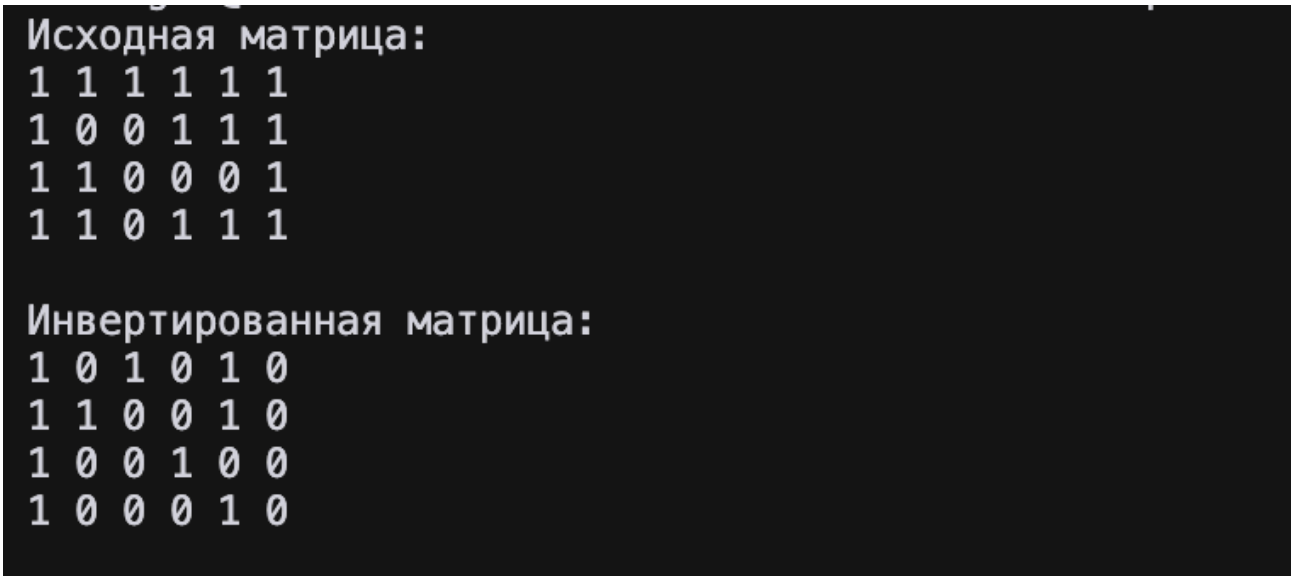

Рисунок 1. Запуск программы на 7-ми процессах.

Выводы.

В процессе выполнения лабораторной работы мы успешно освоили функции коллективной обработки данных в MPI. Мы разобрались в их принципах и приобрели практические навыки их применения для реализации параллельных алгоритмов. Мы научились использовать коллективные операции для обмена данными между процессами и эффективного распределения вычислительной нагрузки. Также мы узнали, что для корректного выполнения параллельной программы необходимо синхронизировать процессы при работе с коллективными функциями.