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»

Студентка гр. 1307 _____ Грунская Н.Д.

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.

Текст программы task1.cpp.

```
#include <cstdlib>
#include <iostream>
#include <ctime>
#include "mpi.h"
using namespace std;

int main(int argc, char *argv[])
{
    int rank, size;
   MPI_Status status;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   int ai=0, max=0;
   srand(rank* time(NULL));
   ai = (rand() % 100) + 1;
   printf("rank = %d, a%d = %d\n", rank,rank,ai);
   MPI_Reduce(&ai,&max,1,MPI_INT,MPI_MAX,0, MPI_COMM_WORLD);
   if(rank==0){
     printf("max = %d\n",max);
   }
    MPI_Finalize();
    return 0;
```

```
}
```

```
natalagrunskaa@MacBook-Pro-Natala lab4 % mpiexec -n 3 ./task1
rank = 0, a0 = 31
rank = 1, a1 = 9
rank = 2, a2 = 17
max = 31
natalagrunskaa@MacBook-Pro-Natala lab4 % mpiexec -n 8 ./task1
rank = 1, a1 = 94
rank = 2, a2 = 87
rank = 3, a3 = 19
rank = 4, a4 = 12
rank = 5, a5 = 44
rank = 0, a0 = 31
rank = 6, a6 = 37
rank = 7, a7 = 83
max = 94
```

Рисунок 1. Запуск программы на 3-х и 8-и процессах.

Текст программы task2.cpp.

```
#include <stdio.h>
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include "mpi.h"

#define ROWS 5
#define COLUMNS 6

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][COLUMNS];
    int modified_matrix[ROWS][COLUMNS];

    if (rank == 0)
    {
```

```
    srand(time(NULL));
    std::cout << "Old matrix:\n";
    for (int i = 0; i < ROWS; i++)
    {
        for (int j = 0; j < COLUMNS; j++)
        {
            matrix[i][j] = (rand() % 21) - 10; // Генерация чисел от -10 до 10
            std::cout << matrix[i][j] << " ";
        }
        std::cout << "\n";
    }
}

// Определяем количество строк для каждого процесса
int *send_counts = new int[size]; // Количество строк для каждого процесса
int *displs = new int[size];      // Смещения для каждого процесса

int rows_per_process = ROWS / size;
int remainder = ROWS % size;

for (int i = 0; i < size; i++)
{
    send_counts[i] = (i < remainder) ? (rows_per_process + 1) * COLUMNS : rows_per_process * COLUMNS;
    displs[i] = (i == 0) ? 0 : displs[i - 1] + send_counts[i - 1];
}

// Количество строк, которые обрабатывает текущий процесс
int local_rows = (rank < remainder) ? rows_per_process + 1 : rows_per_process;
int local_elements = local_rows * COLUMNS;

// Локальный буфер для хранения строк, обрабатываемых текущим процессом
int *local_matrix = new int[local_elements];

// Распределение строк матрицы с использованием MPI_Scatterv
MPI_Scatterv(matrix, send_counts, displs, MPI_INT, local_matrix, local_elements, MPI_INT, 0, MPI_COMM_WORLD);

// Обработка локальной части матрицы
for (int i = 0; i < local_rows; i++)
{
    for (int j = 0; j < COLUMNS; j++)
    {
```

```
            int index = i * COLUMNS + j;
            if (local_matrix[index] > 0)
            {
                local_matrix[index] = 1;
            }
            else if (local_matrix[index] < 0)
            {
                local_matrix[index] = -1;
            }
        }
    }

    // Сбор обработанных данных обратно в процесс с рангом 0
    MPI_Gatherv(local_matrix, local_elements, MPI_INT, modified_matrix, send_counts, displs, MPI_INT, 0,
MPI_COMM_WORLD);

    if (rank == 0)
    {
        std::cout << "\nNew matrix:\n";
        for (int i = 0; i < ROWS; i++)
        {
            for (int j = 0; j < COLUMNS; j++)
            {
                std::cout << modified_matrix[i][j] << " ";
            }
            std::cout << "\n";
        }
    }

    // Освобождение памяти
    delete[] send_counts;
    delete[] displs;
    delete[] local_matrix;

    MPI_Finalize();
    return 0;
}
```

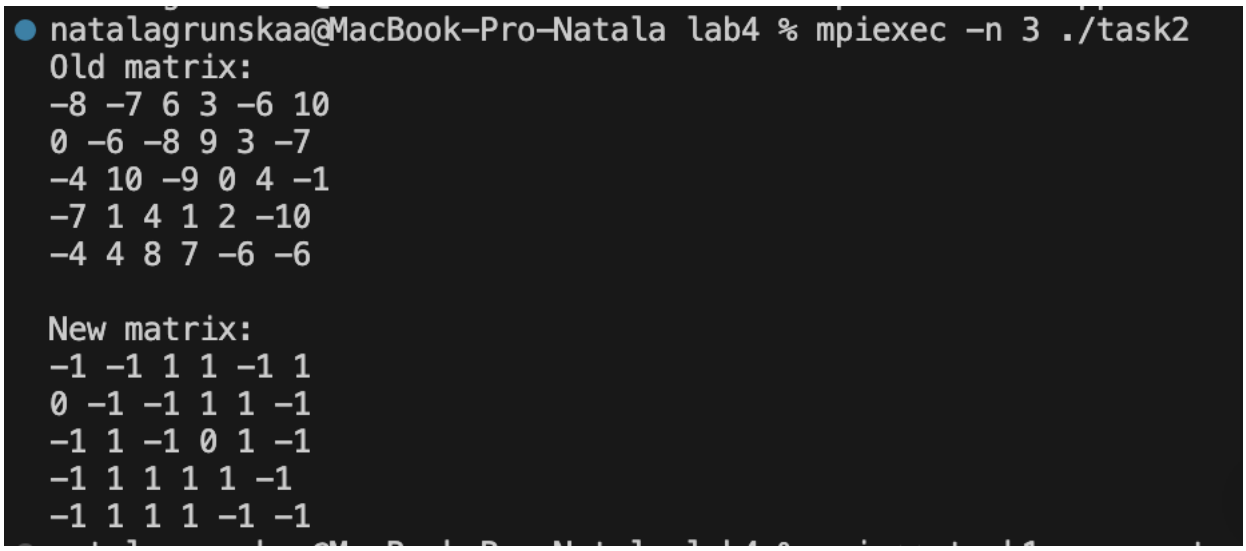

Рисунок 2. Запуск программы на 3 процессах

Выводы.

В ходе выполнения лабораторной работы были успешно освоены функции коллективной обработки данных в MPI. Получено понимание принципов их работы и практические навыки применения для реализации параллельных алгоритмов. Приобретен опыт использования коллективных операций для обмена данными между процессами и эффективного распределения вычислительной нагрузки. Также закреплены знания о необходимости синхронизации процессов при работе с коллективными функциями для обеспечения корректного выполнения параллельной программы.