**МИНОБРНАУКИРОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА(ЛЕНИНА)**
**Кафедра Вычислительной техники**

# ОТЧЁТ

**по лабораторной работе №4**

**по дисциплине «Параллельные алгоритмы и системы»**

**Тема: Коллективные функции**

Студенты гр. 1307 Виноградов А.С.

Преподаватель

Санкт-Петербург

2025

# Лабораторная работа №4

## Цель работы

Освоить коллективные функции

**Задание на лабораторную работу**

Задание 1 (по вариантам).

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

(Запустить n процессов и найти по вариантам:

1. Сумму нечетных элементов вектора;)

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. (Повернуть матрицу и посчитать положительные элементы

## Ход работы

В ходе выполнения лабораторной программы, были видоизменены лабораторные с применением коллективных функций

Что теперь делает программа:

- Вместо использования MPI_Send и MPI_Recv для сбора результатов в процессе 0, используется коллективная функция MPI_Reduce.
- Каждый процесс вычисляет локальную сумму нечетных чисел в своем диапазоне.
- MPI_Reduce собирает все локальные суммы и суммирует их в процессе 0.
- Процесс 0 выводит общую сумму.

2 задание

Алгоритм:

- Используются коллективные функции MPI_Bcast, MPI_Scatterv, MPI_Gatherv и MPI_Reduce.
- Матрица разбивается на части, которые распределяются между процессами.
- Каждый процесс меняет местами столбцы в своей части матрицы.
- После обмена столбцов подсчитывается количество положительных чисел в локальной части.
- Результаты собираются в процессе 0, где выводится итоговая матрица и количество положительных чисел.

## Код программы (lab1.part1.java)

```
package vinandy.Lab4;

import mpi.MPI;

public class Lab4_1 {
    public static void main(String[] args) {
        MPI.Init(args);

        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();

        int start = Integer.parseInt(System.getProperty("start", "1"));
        int end = Integer.parseInt(System.getProperty("end", "100"));
        int range = end - start + 1;

        if (rank >= range) {
            System.out.println("Process " + rank + " is idle.");
        } else {
            int localRange = range / size;
            int remainder = range % size;

            int localStart, localEnd;

            if (rank < remainder) {
                localStart = start + rank * (localRange + 1);
                localEnd = localStart + localRange;
            } else {
                localStart = start + (rank * localRange) + remainder;
                localEnd = localStart + localRange - 1;
            }

            if (localStart > localEnd) {
                int temp = localStart;
                localStart = localEnd;
                localEnd = temp;
            }

            int localSum = 0;
            for (int i = localStart; i <= localEnd; i++) {
                if (i % 2 != 0) { // Проверяем, является ли число нечетным
                    localSum += i;
                }
            }

            System.out.println("Proc " + rank + " (" + localStart + " to " + localEnd + ")
local sum: " + localSum);

            // Используем MPI_Reduce для сбора суммы
            int[] globalSum = new int[1];
            MPI.COMM_WORLD.Reduce(new int[]{localSum}, 0, globalSum, 0, 1, MPI.INT, MPI.SUM,
0);

            if (rank == 0) {
                System.out.println("Total odd sum: " + globalSum[0]);
            }
        }

        MPI.Finalize();
    }
}
```

Задание 2

```
package vinandy.Lab4;

import mpi.MPI;

public class Lab4_2 {
    public static void main(String[] args) {
        MPI.Init(args);

        int np = MPI.COMM_WORLD.Size();
        int rank = MPI.COMM_WORLD.Rank();

        int[][] matrix = null;
        int rows = 0, cols = 0;

        if (rank == 0) {
            String matrixString = System.getProperty("matrix", "-1 -2 3 4;-5 6 -7 0;9 -10 -11
1");
            String[] rowsArray = matrixString.split(";");
            rows = rowsArray.length;
            cols = rowsArray[0].split("\\s+").length;
            matrix = new int[rows][cols];
            for (int i = 0; i < rows; i++) {
                String[] elements = rowsArray[i].split("\\s+");
                for (int j = 0; j < cols; j++) {
                    matrix[i][j] = Integer.parseInt(elements[j]);
                }
            }
            System.out.println("Initial Matrix:");
            printMatrix(matrix);
        }

        // Рассылка размеров матрицы
        int[] dimensions = new int[2];
        if (rank == 0) {
            dimensions[0] = rows;
            dimensions[1] = cols;
        }
        MPI.COMM_WORLD.Bcast(dimensions, 0, 2, MPI.INT, 0);
        rows = dimensions[0];
        cols = dimensions[1];

        // Определение размера локальных данных
        int blockSize = cols / np;
        int remainder = cols % np;
        int[] counts = new int[np];
        int[] displacements = new int[np];

        for (int i = 0; i < np; i++) {
            counts[i] = (blockSize + (i < remainder ? 1 : 0)) * rows;
            displacements[i] = (i == 0) ? 0 : displacements[i - 1] + counts[i - 1];
        }

        // Локальная матрица
        int localCols = counts[rank] / rows;
        int[] localMatrix = new int[rows * localCols];
        int[] flattenedMatrix = null;

        if (rank == 0) {
            flattenedMatrix = new int[rows * cols];
            for (int r = 0; r < rows; r++) {
                System.arraycopy(matrix[r], 0, flattenedMatrix, r * cols, cols);
            }
        }

        // Распределение данных
```

```
        MPI.COMM_WORLD.Scatterv(flattenedMatrix, 0, counts, displacements, MPI.INT,
localMatrix, 0, localMatrix.length, MPI.INT, 0);

        // Обмен столбцами попарно
        for (int i = 0; i < localCols; i += 2) {
            if (i + 1 < localCols) {
                for (int r = 0; r < rows; r++) {
                    int temp = localMatrix[r * localCols + i];
                    localMatrix[r * localCols + i] = localMatrix[r * localCols + i + 1];
                    localMatrix[r * localCols + i + 1] = temp;
                }
            }
        }

        // Сбор данных обратно в процесс 0
        int[] resultMatrix = null;
        if (rank == 0) {
            resultMatrix = new int[rows * cols];
        }
        MPI.COMM_WORLD.Gatherv(localMatrix, 0, localMatrix.length, MPI.INT, resultMatrix, 0,
counts, displacements, MPI.INT, 0);

        // Подсчет положительных чисел
        int localPositiveCount = 0;
        for (int value : localMatrix) {
            if (value > 0) {
                localPositiveCount++;
            }
        }

        // Сбор количества положительных чисел
        int[] totalPositiveCount = new int[1];
        MPI.COMM_WORLD.Reduce(new int[]{localPositiveCount}, 0, totalPositiveCount, 0, 1,
MPI.INT, MPI.SUM, 0);

        // Вывод результата
        if (rank == 0) {
            System.out.println("\nMatrix after swapping columns:");
            for (int r = 0; r < rows; r++) {
                for (int c = 0; c < cols; c++) {
                    System.out.print(resultMatrix[r * cols + c] + " ");
                }
                System.out.println();
            }
            System.out.println("\nNumber of positive elements: " + totalPositiveCount[0]);
        }

        MPI.Finalize();
    }

    private static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int elem : row) {
                System.out.print(elem + " ");
            }
            System.out.println();
        }
    }
}
```

## Вывод

В ходе выполнения лабораторной работы были применены знания, ранее полученные при выполнении прошлых работ. С помощью коллективных функций оба кода адаптированы для использования коллективных функций MPI.