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

# ЛАБОРАТОРНАЯ РАБОТА №4

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

**Тема: Коллективные функции**

Студент гр. 1307 _____ Угрюмов М.М.

Преподаватель _____

Санкт-Петербург
2025

## Введение

**Тема работы**: коллективные функции

**Цель работы**: освоить функции коллективной обработки данных.

## Ход работы

**Задание №1** – запустить n процессов и найти количество четных положительных элементов с использованием коллективных функций MPI. В качестве примера исходное кол-во процессов – -np = 4

Разница между программами в лабораторных работах №2 и №4 заключается в использовании коллективных функций MPI во второй программе, что делает ее более эффективной в распределенной обработке данных. В первой программе каждый процесс независимо генерирует свой собственный массив случайных чисел, выполняет локальный подсчет четных положительных чисел и затем передает результаты в процесс 0 через MPI_Send и MPI_Recv. Процесс 0 вручную собирает данные от всех остальных процессов, складывая их значения. Этот подход менее оптимален, так как требует явного обмена сообщениями между процессами, что может приводить к задержкам и усложнению кода.

Во второй программе используется MPI_Scatter, MPI_Reduce и другие коллективные функции, которые автоматически выполняют передачу данных между процессами, избавляя от необходимости явного использования MPI_Send и MPI_Recv. Вместо того чтобы каждый процесс генерировал полный массив случайных чисел, теперь это делает только процесс 0, а затем MPI_Scatter раздает части массива разным процессам. Это позволяет каждому процессу обрабатывать только свой участок данных, что снижает избыточность вычислений и исключает дублирование данных. После локального подсчета четных положительных чисел каждым процессом вместо ручной передачи результатов через MPI_Send и MPI_Recv используется MPI_Reduce, которая автоматически суммирует результаты со всех процессов и передает итоговое значение в процесс 0.

Результат представлен на Рис.1.

```
MPJ Express (0.44) is started in the multicore configuration
Process 0 (Thread ID: 22) numbers: [-10, -38, 5, -91, -10, 64, -32, -14, -94, 14]
Process 0 found 2 even positive numbers.
Process 1 (Thread ID: 25) numbers: [96, 62, 81, -61, 12, -25, -83, -7, 73, 52]
Process 1 found 4 even positive numbers.
Process 3 (Thread ID: 23) numbers: [-16, 56, -68, -73, -64, -77, -73, -75, -70, -15]
Process 3 found 1 even positive numbers.
Process 2 (Thread ID: 24) numbers: [1, -17, 40, 91, -13, -46, 76, 10, -67, 83]
Process 2 found 3 even positive numbers.
Total even positive numbers: 10
```

Рис.1 – Результат подсчета положительных четных чисел

**Задание №2** – Произведение суммы положительных элементов верхней половины матрицы на сумму отрицательных элементов нижней половины матрицы с использованием коллективных функций MPI.

В предыдущей лабораторной работе использовались MPI_Send и MPI_Recv для передачи данных между процессами. В новой версии программы для этой задачи применяются коллективные функции MPI, что упрощает код и делает его более эффективным. MPI_Scatter позволяет процессу 0 передать части массива всем процессам одновременно, ускоряя распределение данных. MPI_Reduce автоматически собирает локальные результаты и выполняет операцию суммирования, избавляя от необходимости вручную собирать и обрабатывать данные. MPI_Bcast используется для рассылки всей матрицы от процесса 0 ко всем остальным, заменяя серию отдельных отправок на одну коллективную операцию. Эти функции оптимизированы для многопроцессорных систем и ускоряют выполнение программы за счет более эффективного распределения и сбора данных. Результат представлен на Рис.2.

```
MPJ Express (0.44) is started in the multicore configuration
Matrix:
[-36,91, -33,14, 39,72, -22,40, 46,87]
[-25,96, -47,60, 64,04, 44,58, -16,82]
[-52,31, 41,04, 16,11, 26,87, 42,72]
[-22,57, -73,93, -68,87, 25,22, 59,95]
Result (positive sum upper half * negative sum lower half): -42494.26927597148
```

Рис.2 – Результат произведения

## Листинг программ

### Lab4Task1.java

```
import mpi.MPI;
import java.util.Random;

public class Lab4Task1 {

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

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

        int numElements = 40;
        int elementsPerProc = numElements / size;

        int[] numbers = new int[numElements];
        int[] localNumbers = new int[elementsPerProc];

        if (rank == 0) {
            Random random = new Random();
            for (int i = 0; i < numElements; i++) {
                numbers[i] = random.nextInt(201) - 100;
            }
        }

        MPI.COMM_WORLD.Scatter(numbers, 0, elementsPerProc, MPI.INT,
                localNumbers, 0, elementsPerProc, MPI.INT, 0);

        int localEvenCount = countPositive(localNumbers);

        System.out.printf("Process %d (Thread ID: %d) numbers: %s\n", rank,
                Thread.currentThread().getId(), arrayToString(localNumbers));
        System.out.printf("Process %d found %d even positive numbers.\n", rank,
                localEvenCount);

        int[] totalEvenCount = new int[1];
```

```
            MPI.COMM_WORLD.Reduce(new int[]{localEvenCount}, 0, totalEvenCount, 0, 1, MPI.INT,
MPI.SUM, 0);

            if (rank == 0) {
                System.out.printf("Total even positive numbers: %d\n", totalEvenCount[0]);
            }

            MPI.Finalize();
        }

        private static int countPositive(int[] numbers) {
            int count = 0;
            for (int num : numbers) {
                if (num > 0 && num % 2 == 0) {
                    count++;
                }
            }
            return count;
        }

        private static String arrayToString(int[] array) {
            StringBuilder sb = new StringBuilder("[");
            for (int i = 0; i < array.length; i++) {
                sb.append(array[i]);
                if (i < array.length - 1) {
                    sb.append(", ");
                }
            }
            sb.append("]");
            return sb.toString();
        }
    }
```

## Lab4Task2.java

```
import mpi.MPI;
import java.util.Arrays;
import java.util.stream.Collectors;

public class Lab4Task2 {
    public static void main(String[] args) throws Exception {
        MPI.Init(args);

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

        int rows = 4, cols = 5;
        double[][] matrix = new double[rows][cols];
```

```
double[] matrixFlat = new double[rows * cols];

if (rank == 0) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            matrix[i][j] = Math.random() * 200 - 100;
        }
    }
    System.out.println("Matrix:");
    printMatrix(matrix);
    for (int i = 0; i < rows; i++) {
        System.arraycopy(matrix[i], 0, matrixFlat, i * cols, cols);
    }
}

MPI.COMM_WORLD.Bcast(matrixFlat, 0, rows * cols, MPI.DOUBLE, 0);

for (int i = 0; i < rows; i++) {
    System.arraycopy(matrixFlat, i * cols, matrix[i], 0, cols);
}

int rowsPerProcess = rows / size;
int extraRows = rows % size;
int myRows = rank < extraRows ? rowsPerProcess + 1 : rowsPerProcess;
int offset = rank * rowsPerProcess + Math.min(rank, extraRows);

double[] localMatrixFlat = new double[myRows * cols];

MPI.COMM_WORLD.Scatter(matrixFlat, 0, myRows * cols, MPI.DOUBLE,
        localMatrixFlat, 0, myRows * cols, MPI.DOUBLE, 0);

double localPositiveSum = 0;
double localNegativeSum = 0;

for (int i = 0; i < myRows; i++) {
    int globalRow = offset + i;
    for (int j = 0; j < cols; j++) {
        double value = localMatrixFlat[i * cols + j];
        if (globalRow < rows / 2 && value > 0) {
            localPositiveSum += value;
        } else if (globalRow >= rows / 2 && value < 0) {
            localNegativeSum += value;
        }
    }
}

double[] totalPositiveSum = new double[1];
double[] totalNegativeSum = new double[1];

MPI.COMM_WORLD.Reduce(new double[]{localPositiveSum}, 0,
```

```
                totalPositiveSum, 0, 1, MPI.DOUBLE, MPI.SUM, 0);
        MPI.COMM_WORLD.Reduce(new double[]{localNegativeSum}, 0,
                totalNegativeSum, 0, 1, MPI.DOUBLE, MPI.SUM, 0);

        if (rank == 0) {
            double result = totalPositiveSum[0] * totalNegativeSum[0];
            System.out.println("Result (positive sum upper half * negative sum lower half): "
+ result);
        }

        MPI.Finalize();
    }

    private static void printMatrix(double[][] matrix) {
        for (double[] row : matrix) {
            System.out.println(Arrays.stream(row)
                    .mapToObj(v -> String.format("%.2f", v))
                    .collect(Collectors.joining(", ", "[", "]")));
        }
    }
}
```