МИНОБРНАУКИ РОССИИ
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 _____ Таланков В.Р.

Преподаватель _____

Санкт-Петербург
2025

Лабораторная работа №4

Цель работы

Освоить функции коллективной обработки данных.

Задание на лабораторную работу

Задание 1.

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

Задание 2.

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

Ход работы

В ходе выполнения работы были изменены код лабораторной 2 и 3.

При изменении лабораторной работы №2 были добавлены использования функций Scatterv и Reduce. Код был сильно упрощен благодаря этому, например, такой блок кода:

```
if (rank == 0) {
                    // P0 send parts of vector
                    int offset = 0;
                    for (int i = 0; i < np; i++) {
                        int count = blockSize + (i == np - 1 ? remainder : 0);
                        if (i == 0) {
                            // P0 get local vector
                            System.arraycopy(vector, offset, localVector, 0, count);
                        } else {
                            // Send vectors to other P's
                            MPI.COMM_WORLD.Send(vector, offset, count, MPI.INT, i, 0);
                        }
                        offset += count;
                    }
                } else {
                    // Other P's get local vector
                    MPI.COMM_WORLD.Recv(localVector, 0, localSize, MPI.INT, 0, 0);
                }
```

лишь одной строкой:

```
MPI.COMM_WORLD.Scatterv(vector, 0, counts, displacements, MPI.INT, localVector, 0, localSize, MPI.INT, 0);
```

Код программы (lab4.part1.java)

```
package neko.lab4;

import mpi.MPI;

public class part1 {
    public static void main(String[] args) {
        MPI.Init(args);

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

        // Get vector
        String vectorString = System.getProperty("vector", "1 2 3 4 5");
        String[] elements = vectorString.split("\\s+");
        int[] vector = new int[elements.length];
        for (int i = 0; i < elements.length; i++) {
            vector[i] = Integer.parseInt(elements[i]);
        }

        // Size of vector
        int size = vector.length;
        int blockSize = size / np;
        int remainder = size % np;
        int[] counts = new int[np];
        int[] displacements = new int[np];

        // Определяем размеры блоков для каждого процесса
        for (int i = 0; i < np; i++) {
            counts[i] = blockSize + (i < remainder ? 1 : 0);
            displacements[i] = (i == 0) ? 0 : displacements[i - 1] + counts[i - 1];
        }

        // Local vector to P's
        int localSize = counts[rank];
        int[] localVector = new int[localSize];

        // Allocation data to P's
        MPI.COMM_WORLD.Scatterv(vector, 0, counts, displacements, MPI.INT, localVector,
0, localSize, MPI.INT, 0);

        // Sum inside P
        int localSum = 0;
        for (int value : localVector) {
            if (value % 2 != 0) {
                localSum += value;
            }
        }

        // Get sum by reduce
        int[] totalSum = new int[1];
        MPI.COMM_WORLD.Reduce(new int[]{localSum}, 0, totalSum, 0, 1, MPI.INT, MPI.SUM,
0);
        // Output from P0
        if (rank == 0) {
            System.out.println("Sum: " + totalSum[0]);
        }
        MPI.Finalize();
    }
}
```

При изменении лабораторной 3 были применены функции **Bcast** для передачи размеров матрицы, **Scatterv** для распределения данных между процессами, **Gatherv** для сбора обработанных данных и **Reduce** для подсчета неизменных элементов.

Код программы (lab4.part2.java)

```
package neko.lab4;

import mpi.MPI;

public class part2 {
    public static void main(String[] args) {
        MPI.Init(args);

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

        // Input or set matrix
        int[][] matrix = null;
        int rows = 0, cols = 0;

        if (rank == 0) {
            String matrixString = System.getProperty("matrix", "-1 2 -3 4;5 -6 7 -8;-9
10 -11 12");
            String[] matrixRows = matrixString.split(";");
            rows = matrixRows.length;
            cols = matrixRows[0].split("\\s+").length;
            matrix = new int[rows][cols];
            for (int i = 0; i < rows; i++) {
                String[] elements = matrixRows[i].split("\\s+");
                for (int j = 0; j < cols; j++) {
                    matrix[i][j] = Integer.parseInt(elements[j]);
                }
            }
        }

        // Size of matrix
        int[] sizeInfo = new int[2];
        if (rank == 0) {
            sizeInfo[0] = rows;
            sizeInfo[1] = cols;
        }

        // Share size of metrix
        MPI.COMM_WORLD.Bcast(sizeInfo, 0, 2, MPI.INT, 0);
        rows = sizeInfo[0];
        cols = sizeInfo[1];

        // Resolve size of local matrix
        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];
```

```
        }

        // Size of local columns
        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);
            }
        }

        // Allocation data to P's
        MPI.COMM_WORLD.Scatterv(flattenedMatrix,  0,  counts,  displacements,  MPI.INT,
localMatrix, 0, localMatrix.length, MPI.INT, 0);

        // Counting unchanged numbers
        int unchangedCount = 0;

        // Change numbers to 1, -1 and 0
        for (int i = 0; i < localMatrix.length; i++) {
            if (localMatrix[i] > 0) {
                if (localMatrix[i] == 1) {
                    unchangedCount++;
                }
                else {
                    localMatrix[i] = 1;
                }
            } else if (localMatrix[i] < 0) {
                if (localMatrix[i] == -1) {
                    unchangedCount++;
                }
                else {
                    localMatrix[i] = -1;
                }
            } else {
                {
                    unchangedCount++;
                }
            }
        }

        // Result matrix
        int[] resultMatrix = null;
        if (rank == 0) {
            resultMatrix = new int[rows * cols];
        }

        // Get local data from P's
        MPI.COMM_WORLD.Gatherv(localMatrix,     0,     localMatrix.length,     MPI.INT,
resultMatrix, 0, counts, displacements, MPI.INT, 0);

        // Count unchanged elements
        int[] totalUnchanged = new int[1];
        MPI.COMM_WORLD.Reduce(new  int[]{unchangedCount},  0,  totalUnchanged,  0,  1,
MPI.INT, MPI.SUM, 0);

        // Results
        if (rank == 0) {
```

```
            System.out.println("Result Matrix:");
            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("Total unchanged elements: " + totalUnchanged[0]);
        }

        MPI.Finalize();
    }
}
```

Вывод

В ходе выполнения лабораторной работы мы смогли на практических примерах ознакомиться с устройством коллективных функций и тем, насколько они упрощают и улучшают код, совершая те же самые действия. Это заметно упрощает читаемость кода и уменьшает его размер (иногда на треть).

Коллективные функции можно использовать и в будущем, что ускорит разработку и позволит избежать детального написания функций и кода.