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

ОТЧЁТ

по лабораторной работе №3
по дисциплине «Параллельные алгоритмы и системы»
Тема: Передача данных между процессами

Студенты гр. 1307 _____ Таланков В.Р.

Преподаватель _____

Санкт-Петербург
2025

Лабораторная работа №3

Цель работы

Освоить функции передачи данных между процессами.

Задание на лабораторную работу

Задание 1.

Заменить в прямоугольной матрице все положительные элементы на 1, отрицательные на -1.

Задание 2.

Количество элементов, оставшихся неизменными в новой матрице.

Ход работы

В ходе выполнения работы был реализован итоговый файл, покрывающий задания. Запуск данного файла производится через командную строку внутри Intellij IDEA.

Процесс 0 распределяет данные матрицы между процессами (в том числе оставляя для себя параметры) и отправляет их на обработку. В конце эти обработанные данные так же собираются в этом процессе и уже результируются.

Рисунок 1 – Результат работы программы (4 процесса)

Код программы (lab1.part1.java)

```
package neko.lab3;

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();

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

        // Size of matrix
        int rows = 0, cols = 0;
        if (rank == 0) {
            rows = matrix.length;
            cols = matrix[0].length;
        }

        // Sending and getting local matrix
        if (rank == 0) {
            // P0 send sizes of matrix
            for (int i = 1; i < np; i++) {
                MPI.COMM_WORLD.Send(new int[]{rows}, 0, 1, MPI.INT, i, 0);
                MPI.COMM_WORLD.Send(new int[]{cols}, 0, 1, MPI.INT, i, 1);
            }
        } else {
            // Other P's gets matrix
            int[] receivedRows = new int[1];
            int[] receivedCols = new int[1];
            MPI.COMM_WORLD.Recv(receivedRows, 0, 1, MPI.INT, 0, 0);
            MPI.COMM_WORLD.Recv(receivedCols, 0, 1, MPI.INT, 0, 1);
            rows = receivedRows[0];
            cols = receivedCols[0];
        }

        // Resolve size of local matrix
        int blockSize = cols / np;
        int remainder = cols % np;

        // Size of local columns
```

```
        int localCols = blockSize + (rank < remainder ? 1 : 0);
        int[] localMatrix = new int[rows * localCols];

        if (rank == 0) {
            // P0 sends parts of matrix
            int offset = 0;
            for (int i = 0; i < np; i++) {
                int count = blockSize + (i < remainder ? 1 : 0);
                if (i == 0) {
                    // Local part for P0
                    for (int r = 0; r < rows; r++) {
                        if (count > 0) System.arraycopy(matrix[r], offset, localMatrix,
r * count, count);
                    }
                } else {
                    // Parts for other P's
                    int[] data = new int[rows * count];
                    for (int r = 0; r < rows; r++) {
                        if (count > 0) System.arraycopy(matrix[r], offset, data, r *
count, count);
                    }
                    MPI.COMM_WORLD.Send(data, 0, data.length, MPI.INT, i, 2);
                }
                offset += count;
            }
        } else {
            // Getting local matrix from P0
            MPI.COMM_WORLD.Recv(localMatrix, 0, localMatrix.length, MPI.INT, 0, 2);
        }

        // Counting unchanged numbers
        int unchangedCount = 0;

        // Change numbers to 1, -1 and 0
        for (int c = 0; c < localCols; c++) {
            for (int r = 0; r < rows; r++) {
                int value = localMatrix[r * localCols + c];
                if (value > 0) {
                    if (value == 1)
                    {
                        unchangedCount++;
                    }
                    else {
                        localMatrix[r * localCols + c] = 1;
                    }
                } else if (value < 0) {
                    if (value == -1)
                    {
                        unchangedCount++;
                    }
                    else {
                        localMatrix[r * localCols + c] = -1;
                    }
                } else {
                    unchangedCount++;
                }
            }
        }

        // Sum results inside P0
        if (rank == 0) {
```

```
            // Create result matrix
            int[][] resultMatrix = new int[rows][cols];
            int offset = 0;
            int totalUnchanged = 0;

            for (int i = 0; i < np; i++) {
                int count = blockSize + (i < remainder ? 1 : 0);
                if (i == 0) {
                    // Local part for P0
                    for (int r = 0; r < rows; r++) {
                        if (count > 0) System.arraycopy(localMatrix, r * count,
resultMatrix[r], offset, count);
                    }
                    totalUnchanged += unchangedCount; // Result count of unchanged
numbers
                } else {
                    // Parts from other P's
                    int[] data = new int[rows * count];
                    MPI.COMM_WORLD.Recv(data, 0, data.length, MPI.INT, i, 3);
                    for (int r = 0; r < rows; r++) {
                        if (count > 0) System.arraycopy(data, r * count,
resultMatrix[r], offset, count);
                    }

                    // Unchanged numbers from other P's
                    int[] recvUnchanged = new int[1];
                    MPI.COMM_WORLD.Recv(recvUnchanged, 0, 1, MPI.INT, i, 4);
                    totalUnchanged += recvUnchanged[0];
                }
                offset += count;
            }

            // Results
            System.out.println("Result Matrix:");
            for (int r = 0; r < rows; r++) {
                for (int c = 0; c < cols; c++) {
                    System.out.print(resultMatrix[r][c] + " ");
                }
                System.out.println();
            }

            System.out.println("Total unchanged elements: " + totalUnchanged);
        } else {
            // Sending results to P0
            MPI.COMM_WORLD.Send(localMatrix, 0, localMatrix.length, MPI.INT, 0, 3);
            MPI.COMM_WORLD.Send(new int[]{unchangedCount}, 0, 1, MPI.INT, 0, 4);
        }

        MPI.Finalize();
    }
}
```

Вывод

В ходе выполнения лабораторной работы были применены знания, ранее полученные при выполнении прошлых работ. С помощью стандартных команд MPI Send и Recv производился обмен данными, распределяя нагрузку между процессами.

Данная работа позволила закрепить навыки написания кода с использованием библиотеки MPI, решая задачу по работе с матрицей и ее обработкой.