

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

Виноградов А.С.

Преподаватель

Санкт-Петербург

2025

Лабораторная работа №3

Цель работы

Освоить функции передачи данных между процессами.

Задание на лабораторную работу

Задание 1.

Поменять местами попарно столбцы прямоугольной матрицы

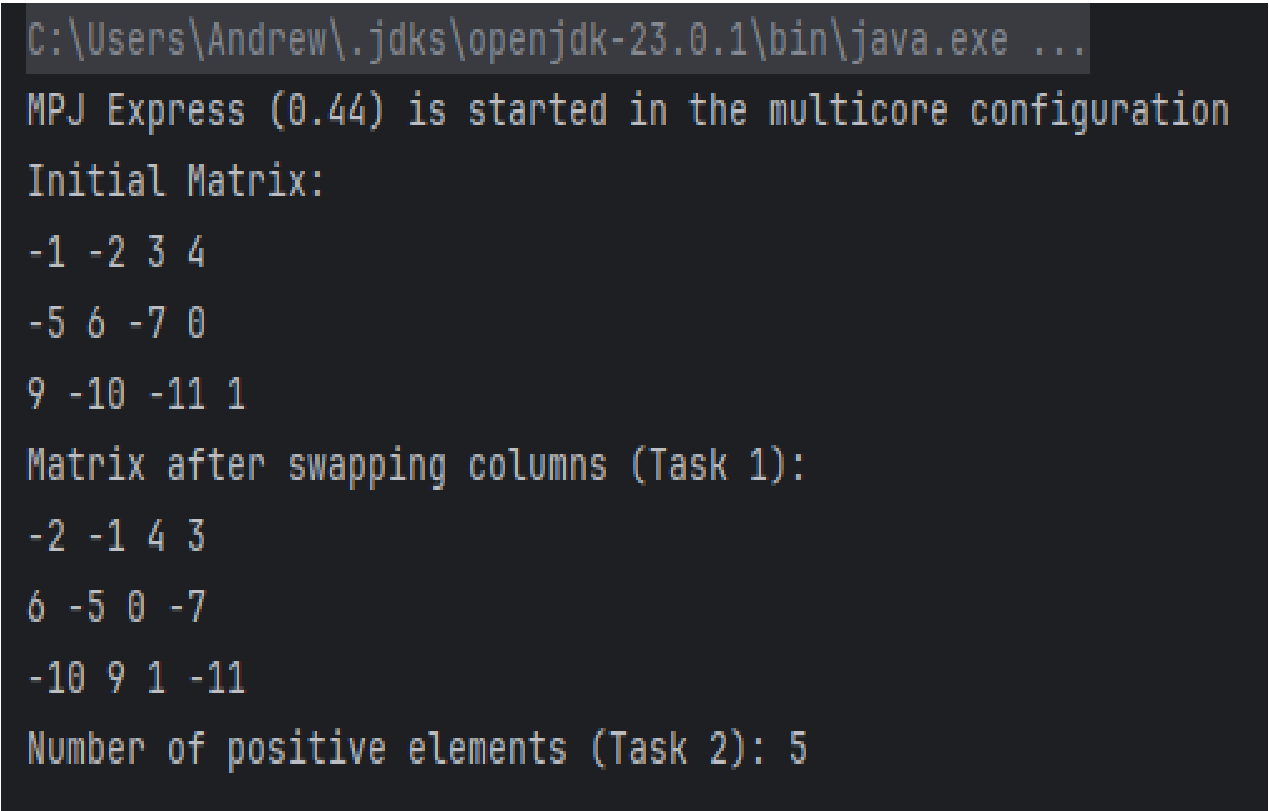
Задание 2.

Количество положительных чисел в новой матрице.

Ход работы

В ходе выполнения работы был реализован итоговый файл, покрывающий задания.

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.



```
C:\Users\Andrew\.jdk\openjdk-23.0.1\bin\java.exe ...
MPJ Express (0.44) is started in the multicore configuration
Initial Matrix:
-1 -2 3 4
-5 6 -7 0
9 -10 -11 1
Matrix after swapping columns (Task 1):
-2 -1 4 3
6 -5 0 -7
-10 9 1 -11
Number of positive elements (Task 2): 5
```

Рисунок 1 – Результат работы программы (10 процессов)

Код программы

```
package vinandy.Lab3;

import mpi.MPI;

public class Lab3 {
    public static void main(String[] args) {
        MPI.Init(args);

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

        int[][] matrix;
        if (rank == 0) {
            String matrixString = System.getProperty("matrix", "-1 -2 3 4;-5 6 -7 0;9 -10 -11
1");

            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]);
                }
            }
            System.out.println("Initial Matrix:");
            printMatrix(matrix);
        } else {
            matrix = null;
        }

        int[] dimensions = new int[2];
        if (rank == 0) {
            dimensions[0] = matrix.length;
            dimensions[1] = matrix[0].length;
        }

        MPI.COMM_WORLD.Bcast(dimensions, 0, 2, MPI.INT, 0);
        int rows = dimensions[0];
        int cols = dimensions[1];

        int blockSize = cols / 2 / np;
        int remainder = (cols / 2) % np;
        int localPairs = blockSize + (rank < remainder ? 1 : 0);
        int localCols = localPairs * 2;

        int[] localMatrix = new int[rows * localCols];

        if (rank == 0) {
            int offset = 0;
            for (int i = 0; i < np; i++) {
                int currPairs = blockSize + (i < remainder ? 1 : 0);
                int currCols = currPairs * 2;

                if (i == 0) {
                    for (int r = 0; r < rows; r++) {
                        System.arraycopy(matrix[r], 0, localMatrix, r * localCols, localCols);
                    }
                } else {
                    int[] sendBuffer = new int[rows * currCols];
                    for (int r = 0; r < rows; r++) {
                        System.arraycopy(matrix[r], offset, sendBuffer, r * currCols,
currCols);
                    }
                    MPI.COMM_WORLD.Send(sendBuffer, 0, sendBuffer.length, MPI.INT, i, 0);
                }
            }
        }
    }
}
```

```

        }
        offset += currCols;
    }
} else {
    MPI.COMM_WORLD.Recv(localMatrix, 0, localMatrix.length, MPI.INT, 0, 0);
}

for (int pair = 0; pair < localPairs; pair++) {
    int leftCol = pair * 2;
    int rightCol = pair * 2 + 1;
    for (int r = 0; r < rows; r++) {
        int temp = localMatrix[r * localCols + leftCol];
        localMatrix[r * localCols + leftCol] = localMatrix[r * localCols + rightCol];
        localMatrix[r * localCols + rightCol] = temp;
    }
}

int[][] resultMatrix = null;
if (rank == 0) {
    resultMatrix = new int[rows][cols];
    int offset = 0;
    for (int i = 0; i < np; i++) {
        int currPairs = blockSize + (i < remainder ? 1 : 0);
        int currCols = currPairs * 2;

        if (i == 0) {
            for (int r = 0; r < rows; r++) {
                System.arraycopy(localMatrix, r * localCols, resultMatrix[r], 0,
localCols);
            }
        } else {
            int[] recvBuffer = new int[rows * currCols];
            MPI.COMM_WORLD.Recv(recvBuffer, 0, recvBuffer.length, MPI.INT, i, 1);
            for (int r = 0; r < rows; r++) {
                System.arraycopy(recvBuffer, r * currCols, resultMatrix[r], offset,
currCols);
            }
            offset += currCols;
        }
    }
} else {
    MPI.COMM_WORLD.Send(localMatrix, 0, localMatrix.length, MPI.INT, 0, 1);
}

if (rank == 0) {
    System.out.println("\nMatrix after swapping columns (Task 1):");
    printMatrix(resultMatrix);

    int positiveCount = 0;
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            if (resultMatrix[r][c] > 0) {
                positiveCount++;
            }
        }
    }
    System.out.println("\nNumber of positive elements (Task 2): " + positiveCount);
}

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 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.