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

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине «Параллельные алгоритмы и системы»

Тема: ПЕРЕДАЧА ДАННЫХ МЕЖДУ ПРОЦЕССАМИ

Студент гр. 1307 _____ Николаев К.Д.

Преподаватель _____

Санкт-Петербург

2025

Введение

Тема работы: Передача данных между процессами.

Цель работы: Освоить функции передачи данных между процессами.

Задания (Вариант 8):

1) В прямоугольной матрице минимальным воздействием на элементы, сделать все значения кратными 2 и 3. Примеры: из 5 делаем 6, из 7 делаем тоже 6, из 10 делаем 12, из 15 делаем тоже 12.
2) Среднее арифметическое значение положительных отклонений чисел от последнего элемента матрицы.

```
Initial matrix:
 27   8   4  29  21  18
 16   1  28  28   1   4
  1  29  28  12  19   1
 20  25  29  23  12  29
 29  17   8  15  12  15
Process  0 (ID:  22) is modifying row -  0
Process  1 (ID:  24) is modifying row -  1
Process  2 (ID:  23) is modifying row -  2
Process  0 (ID:  22) is modifying row -  3
Process  1 (ID:  24) is modifying row -  4
Modified matrix:
 24   6   6  30  18  18
 18   0  30  30   0   6
  0  30  30  12  18   0
 18  24  30  24  12  30
 30  18   6  12  12  12
```

Рисунок 1 - результат работы 1-ой части программы (количество процессов = 3).

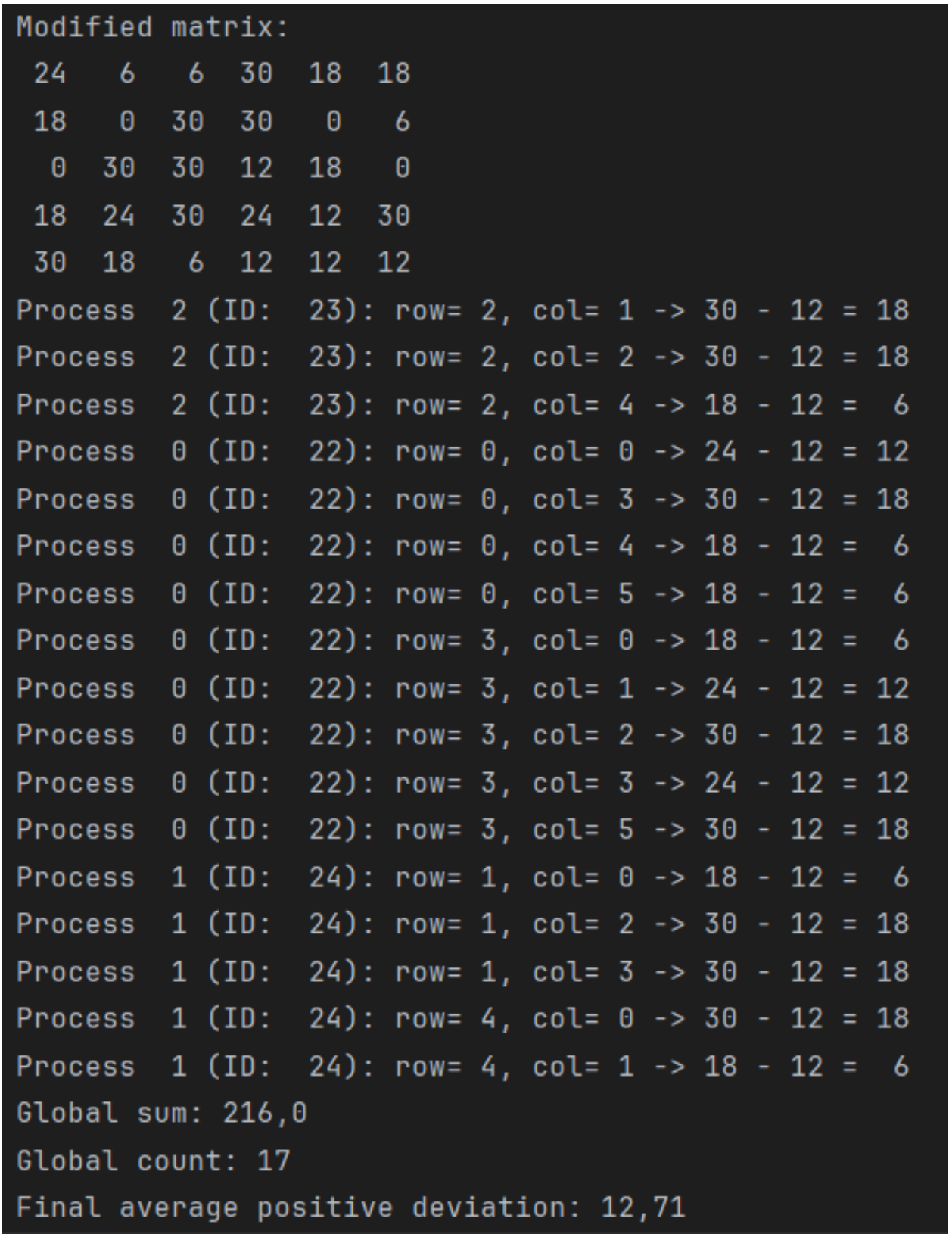

Рисунок 2 - результат работы 2-ой части программы (количество процессов = 3).

Логика программы:

1. Инициализация и генерация матрицы:

- Процесс 0 создает матрицу rows × cols со случайными числами от 1 до 30.
- Выводит исходную матрицу.

2. Распределение строк между процессами:
   - Каждая строка закрепляется за определенным процессом (i % size).
   - Процесс 0 отправляет строки другим процессам, если они отвечают за них.
   - Ответственный процесс модифицирует строку, заменяя числа на ближайшие, кратные 2 и 3.
   - Затем измененная строка отправляется обратно процессу 0.
3. Рассылка измененной матрицы:
   - Процесс 0 передает обновленную матрицу всем процессам с помощью Bcast.
4. Определение последнего элемента матрицы и его рассылка:
   - Последний элемент (matrix[rows - 1][cols - 1]) передается всем процессам через Bcast.
5. Локальное вычисление отклонений:
   - Каждый процесс обрабатывает только часть строк (по индексу i = rank + size), чтобы избежать дублирования.
   - Вычисляет разницу между каждым элементом и последним элементом матрицы.
   - Если разница положительная, добавляет ее в локальную сумму и увеличивает счетчик.
6. Сбор и расчет глобального среднего отклонения:
   - Все локальные суммы и счетчики передаются процессу 0 через Reduce.
   - Процесс 0 вычисляет среднее отклонение и выводит результат.

Вывод

В ходе выполнения лабораторной работы была изучена работа с библиотекой MPI (Message Passing Interface) в Java для организации параллельных вычислений. Программа реализует обработку матрицы с распределением строк между процессами и вычислением средних отклонений значений от последнего элемента.

1. Использованные методы MPI

MPI.Init(args):

Инициализирует MPI-окружение. Должен быть вызван перед использованием любых других MPI-функций.

MPI.COMM_WORLD.Rank():

Возвращает ранг (идентификатор) текущего процесса. Ранг используется для идентификации процесса и управления его поведением.

MPI.COMM_WORLD.Size():

Возвращает общее количество процессов, участвующих в выполнении программы.

MPI.COMM_WORLD.Send(buffer, offset, count, datatype, destination, tag):

Отправляет данные другому процессу. Параметры:

buffer: Массив данных для отправки.

offset: Смещение в массиве.

count: Количество элементов для отправки.

datatype: Тип данных (например, MPI.INT).

destination: Ранг процесса-получателя.

tag: Метка сообщения (используется для идентификации).

MPI.COMM_WORLD.Recv(buffer, offset, count, datatype, source, tag):

Получает данные от другого процесса. Параметры:

buffer: Массив для сохранения полученных данных.

offset: Смещение в массиве.

count: Количество элементов для получения.

datatype: Тип данных (например, MPI.INT).

source: Ранг процесса-отправителя.

tag: Метка сообщения (должна совпадать с тегом отправки).

MPI.COMM_WORLD.Bcast(buffer, offset, count, datatype, root):

Осуществляет широковещательную (broadcast) передачу данных от одного процесса (root) ко всем остальным. Параметры:

buffer: Массив данных, который отправляет процесс-источник (root) и получает все остальные процессы.

offset: Смещение в массиве.

count: Количество элементов для передачи.

datatype: Тип данных (например, MPI.INT).

root: Ранг процесса, который выполняет отправку данных.

MPI.COMM_WORLD.Reduce(sendbuf, sendoffset, recvbuf, recvoffset, count, datatype, op, root):

Выполняет редукцию (сбор данных от всех процессов) с применением указанной операции (op) и сохраняет результат на указанном процессе (root). Параметры:

sendbuf: Локальный массив с данными, которые передает каждый процесс.

sendoffset: Смещение в sendbuf.

recvbuf: Массив, в котором процесс root получает итоговый результат.

recvoffset: Смещение в recvbuf.

count: Количество элементов для редукции.

datatype: Тип данных (например, MPI.DOUBLE).

op: Операция редукции (например, MPI.SUM для суммирования, MPI.MAX для поиска максимального значения).

root: Процесс, который собирает и сохраняет итоговый результат.

MPI.Finalize():

Завершает работу MPI-окружения. Освобождает ресурсы, выделенные для MPI. Должен быть вызван в конце программы.

2. Основные выводы

Распределение задач:

MPI позволяет разделять работу между процессами, избегая дублирования вычислений

Обмен данными:

Использование Send, Recv, Bcast позволяет организовать эффективное взаимодействие между процессами.

Гибкость:

Возможность распределять строки матрицы динамически между процессами делает программу масштабируемой.

Эффективность:

Применение Reduce позволяет минимизировать затраты на сбор данных и их обработку в одном процессе.

3. Заключение

В ходе выполнения лабораторной работы были изучены и реализованы основные методы MPI, позволяющие организовать параллельную обработку данных. Программа демонстрирует эффективное использование MPI для распределенной обработки матрицы и вычисления статистических данных.

Приложение А

```
package lebibop.lab3;

import mpi.MPI;
import mpi.MPIException;

import java.util.Random;

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

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

        int rows = Integer.parseInt(System.getProperty("row", "10"));
        int cols = Integer.parseInt(System.getProperty("col", "11"));

        if (rows == cols) {
            cols++;
        }

        int[][] matrix = new int[rows][cols];

        if (rank == 0) {
            Random rand = new Random();
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    matrix[i][j] = rand.nextInt(30) + 1;
                }
            }
            System.out.println("Initial matrix:");
            printMatrix(matrix);
        }

        for (int i = 0; i < rows; i++) {
            int responsibleProcess = i % size;
            int[] rowBuffer = new int[cols];

            if (rank == 0) {
```

```
            System.arraycopy(matrix[i], 0, rowBuffer, 0, cols);
            if (responsibleProcess != 0) {
                MPI.COMM_WORLD.Send(rowBuffer, 0, cols, MPI.INT, responsibleProcess, i);
            }
        }

        if (rank == responsibleProcess) {
            if (rank != 0) {
                MPI.COMM_WORLD.Recv(rowBuffer, 0, cols, MPI.INT, 0, i);
            }
            System.out.printf("Process %2d (ID: %3d) is modifying row - %2d%n",
                    rank, Thread.currentThread().getId(), i);
            modifyRow(rowBuffer);

            if (rank != 0) {
                MPI.COMM_WORLD.Send(rowBuffer, 0, cols, MPI.INT, 0, i);
            }
        }

        if (rank == 0 && responsibleProcess != 0) {
            MPI.COMM_WORLD.Recv(rowBuffer, 0, cols, MPI.INT, responsibleProcess, i);
        }

        if (rank == 0) {
            System.arraycopy(rowBuffer, 0, matrix[i], 0, cols);
        }
    }

    for (int i = 0; i < rows; i++) {
        MPI.COMM_WORLD.Bcast(matrix[i], 0, cols, MPI.INT, 0);
    }

    if (rank == 0) {
        System.out.println("Modified matrix:");
        printMatrix(matrix);
    }

    int[] lastElementBuffer = new int[1];

    if (rank == 0) {
        lastElementBuffer[0] = matrix[rows - 1][cols - 1];
    }
```

```
        MPI.COMM_WORLD.Bcast(lastElementBuffer, 0, 1, MPI.INT, 0);

        int lastElement = lastElementBuffer[0];

        double localSum = 0;
        int localCount = 0;

        for (int i = rank; i < rows; i += size) {
            for (int j = 0; j < cols; j++) {
                int deviation = matrix[i][j] - lastElement;
                if (deviation > 0) {
                    localSum += deviation;
                    localCount++;
                    System.out.printf("Process %2d (ID: %3d): row=%2d, col=%2d -> %2d - %2d = %2d%n",
                            rank, Thread.currentThread().getId(), i, j, matrix[i][j], lastElement, deviation);
                }
            }
        }

        double[] globalSum = new double[1];
        int[] globalCount = new int[1];

        MPI.COMM_WORLD.Reduce(new double[]{localSum}, 0, globalSum, 0, 1, MPI.DOUBLE, MPI.SUM, 0);
        MPI.COMM_WORLD.Reduce(new int[]{localCount}, 0, globalCount, 0, 1, MPI.INT, MPI.SUM, 0);

        if (rank == 0) {
            System.out.printf("Global sum: %.1f%n", globalSum[0]);
            System.out.printf("Global count: %2d%n", globalCount[0]);
            double finalResult = (globalCount[0] > 0) ? (globalSum[0] / globalCount[0]) : 0;
            System.out.printf("Final average positive deviation: %.2f%n", finalResult);
        }

        MPI.Finalize();
    }

    private static void modifyRow(int[] row) {
        for (int i = 0; i < row.length; i++) {
            row[i] = makeDivisibleBy2And3(row[i]);
        }
    }
```

```
    private static int makeDivisibleBy2And3(int num) {
        int lower = (num / 6) * 6;
        int upper = lower + 6;
        return (num - lower <= upper - num) ? lower : upper;
    }

    private static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int num : row) {
                System.out.printf("%3d ", num);
            }
            System.out.println();
        }
    }
}
```