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:
 30  23  16  27  18  28
 22  20  28  13   5  10
 22  20  26  14  18  27
 20   4  15  18   4  25
  7   7  25  10   7  24
Process 1 received rows 2 to 3
Process 0 received rows 0 to 1
Process 2 received rows 4 to 4
```

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

```
Modified matrix:
 30  24  18  24  18  30
 24  18  30  12   6  12
 24  18  24  12  18  24
 18   6  12  18   6  24
  6   6  24  12   6  24
Process  0 (ID:  22): row= 0, col= 0 -> 30 - 24 =  6
Process  0 (ID:  22): row= 0, col= 5 -> 30 - 24 =  6
Process  1 (ID:  24): row= 1, col= 2 -> 30 - 24 =  6
Global sum: 18,0
Global count:  3
Final average positive deviation: 6,00
```

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

Логика программы:

Инициализация и генерация матрицы:

- Процесс 0 создаёт матрицу rows × cols, заполняя её случайными числами от 1 до 30.
- Исходная матрица выводится на экран.
- Матрица преобразуется в одномерный массив (flatMatrix) для удобства передачи.

Распределение строк между процессами (Scatterv):

- Вычисляется, сколько строк получает каждый процесс (чтобы нагрузка делилась равномерно).
- Используется MPI.Scatterv, чтобы каждому процессу отправить только его строки.
- Каждый процесс сообщает, какие строки он получил.

Модификация строк каждым процессом:

- Каждый процесс обрабатывает только свои строки.
- Каждое число заменяется на ближайшее число, кратное 2 и 3.

Сбор модифицированных строк обратно к процессу 0 (Gatherv):

- Используется MPI.Gatherv, чтобы собрать все изменённые строки обратно на процесс 0.
- Процесс 0 восстанавливает двумерную матрицу и выводит модифицированную версию.

Определение последнего элемента матрицы и его рассылка:

- Процесс 0 берёт последний элемент матрицы (matrix[rows - 1][cols - 1]) и рассылает его всем процессам через MPI.Bcast.

Локальное вычисление положительных отклонений:

- Каждый процесс обрабатывает разные строки по принципу i = rank + size * k (чтобы избежать повторной обработки).
- Для каждого элемента вычисляется разность с последним элементом.
- Если разность положительная — она добавляется к локальной сумме и увеличивается локальный счётчик.

Сбор и расчёт глобального среднего отклонения:

- Локальные суммы и счётчики собираются на процесс 0 с помощью MPI.Reduce.
- Процесс 0 вычисляет глобальное среднее положительное 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.Gatherv(sendbuf, sendoffset, sendcount, sendtype, recvbuf, recvoffset, recvcounts, displs, recvtype, root):

Собирает переменное количество данных от всех процессов на root.

MPI.COMM_WORLD.Scatterv(sendbuf, sendoffset, sendcounts, displs, sendtype, recvbuf, recvoffset, recvcount, recvtype, root):

Рассылает переменное количество данных разным процессам.

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

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];
        int[] flatMatrix = 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++) {
                System.arraycopy(matrix[i], 0, flatMatrix, i * cols, cols);
            }
        }

        int baseRows = rows / size;
        int extra = rows % size;
        int[] sendCounts = new int[size];
        int[] displs = new int[size];
        int offset = 0;
        for (int i = 0; i < size; i++) {
            int myRows = baseRows + (i < extra ? 1 : 0);
```

```
    sendCounts[i] = myRows * cols;
    displs[i] = offset;
    offset += sendCounts[i];
}

int recvCount = sendCounts[rank];
int recvRows = recvCount / cols;
int[] recvBuffer = new int[recvCount];

MPI.COMM_WORLD.Scatterv(flatMatrix, 0, sendCounts, displs, MPI.INT,
        recvBuffer, 0, recvCount, MPI.INT, 0);

if (recvRows > 0) {
    int firstRow = displs[rank] / cols;
    int lastRow = firstRow + recvRows - 1;
    System.out.printf("Process %d received rows %d to %d%n", rank, firstRow, lastRow);
} else {
    System.out.printf("Process %d received no rows%n", rank);
}

for (int i = 0; i < recvBuffer.length; i++) {
    recvBuffer[i] = makeDivisibleBy2And3(recvBuffer[i]);
}

MPI.COMM_WORLD.Gatherv(recvBuffer, 0, recvCount, MPI.INT,
        flatMatrix, 0, sendCounts, displs, MPI.INT, 0);

if (rank == 0) {
    for (int i = 0; i < rows; i++) {
        System.arraycopy(flatMatrix, i * cols, matrix[i], 0, cols);
    }
    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;
    int globalRowStart = displs[rank] / cols;

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