

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 по процессам

Студент гр. 1307

Угрюмов М.М.

Преподаватель

Санкт-Петербург

2025

Введение

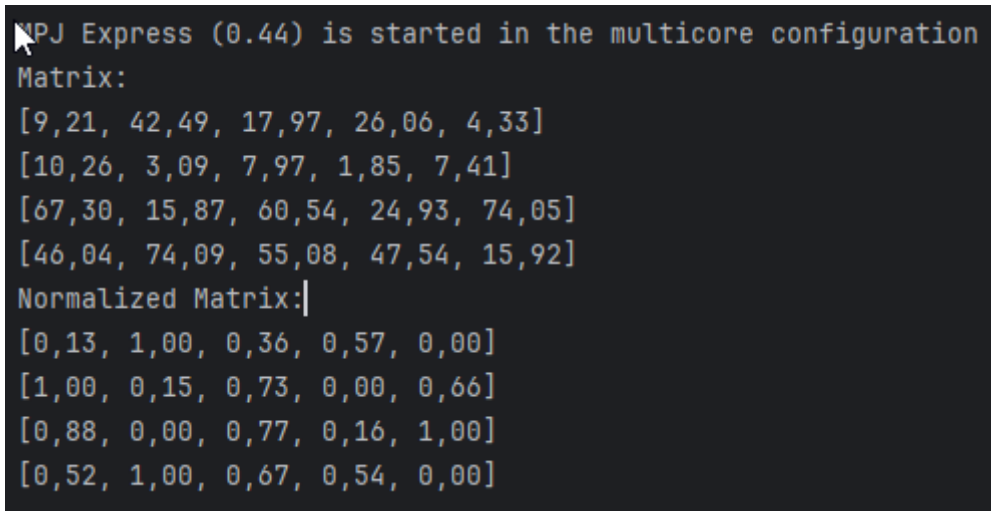
Тема работы: запуск параллельной программы и передача данных по процессам.

Цель работы: освоить функции передачи данных между процессами.

Ход работы

Задание №1 – нормировать элементы каждой строки прямоугольной матрицы (свести значения в диапазон от 0 до 1), вариант 15;

Нормализация матрицы – это процесс преобразования значений так, чтобы они находились в диапазоне от 0 до 1. В данном случае нормализация выполняется **по строкам**, то есть каждая строка обрабатывается отдельно. Для этого сначала находится **минимальное и максимальное значение** в строке. Затем каждое число в строке пересчитывается по формуле: вычитается минимальное значение, а результат делится на разницу между максимальным и минимальным значениями. Это позволяет преобразовать минимальный элемент строки в 0, а максимальный — в 1, при этом все остальные элементы получают значения между 0 и 1 в зависимости от их положения относительно этих границ. Результат представлен на Рис.1.



```
MPJ Express (0.44) is started in the multicore configuration
Matrix:
[9,21, 42,49, 17,97, 26,06, 4,33]
[10,26, 3,09, 7,97, 1,85, 7,41]
[67,30, 15,87, 60,54, 24,93, 74,05]
[46,04, 74,09, 55,08, 47,54, 15,92]
Normalized Matrix:|
[0,13, 1,00, 0,36, 0,57, 0,00]
[1,00, 0,15, 0,73, 0,00, 0,66]
[0,88, 0,00, 0,77, 0,16, 1,00]
[0,52, 1,00, 0,67, 0,54, 0,00]
```

Рис.1 – Результат нормирования матрицы

Задание №2 (Вариант 15): Произведение суммы положительных элементов верхней половины матрицы на сумму отрицательных элементов нижней половины матрицы. Каждый процесс, идентифицированный своим уникальным рангом, работает с частью матрицы и вычисляет локальные суммы: одну для положительных элементов верхней половины матрицы и другую для отрицательных элементов нижней половины. Эти локальные суммы рассчитываются только для данных, которые процесс получил в свою часть матрицы, разделенную между всеми процессами. Распределение матрицы по процессам осуществляется с учетом количества строк, которое каждый процесс обрабатывает, что обеспечивает равномерную нагрузку. После того как каждый процесс вычисляет свои локальные суммы, они собираются на процессе с рангом 0 с помощью функции Reduce, которая объединяет данные с использованием операции суммы (MPI.SUM). Корневой процесс (ранг 0) получает итоговые суммы для положительных и отрицательных элементов и затем вычисляет произведение этих двух сумм, которое и является конечным результатом. Таким образом, программа эффективно распределяет задачи между процессами, используя параллелизм для вычислений, и агрегирует результаты в корневом процессе для получения итогового значения. Результат представлен на Рис.2.

```
MPI Express (0.44) is started in the multicore configuration
Matrix:
[-30,14, -78,28, 58,35, 32,37, -0,39]
[-17,82, 71,75, 54,51, 70,64, 89,99]
[78,38, -18,47, 87,50, -51,40, -96,90]
[68,34, -4,32, -31,94, -66,93, 34,99]
Rank 3 positive sum: 0.0
Rank 2 positive sum: 0.0
Rank 1 positive sum: 286.89239826313883
Rank 3 negative sum: -103.19333526844369
Rank 0 positive sum: 90.71259623991841
Rank 2 negative sum: -166.76169259972852
Rank 1 negative sum: 0.0
Rank 0 negative sum: 0.0
Result (positive sum upper half * negative sum lower half): -101936.36681423384
```

Рис.2 – Результат произведения

Листинг программ

Lab3Task1.java

```
import mpi.MPI;

import java.util.Arrays;
import java.util.stream.Collectors;

public class Lab3Task1 {
    public static void main(String[] args) throws Exception {
        MPI.Init(args);

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

        int rows = 4, cols = 5;
        double[][] matrix = null;
        double[][] normalizedMatrix = new double[rows][cols];

        if (rank == 0) {
            matrix = new double[rows][cols];
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    matrix[i][j] = Math.random() * 100;
                }
            }
            System.out.println("Matrix:");
            printMatrix(matrix);
        }

        int rowsPerProcess = rows / size;
        int extraRows = rows % size;
        int myRows = rank < extraRows ? rowsPerProcess + 1 : rowsPerProcess;

        double[][] localMatrix = new double[myRows][cols];

        if (rank == 0) {
            int offset = 0;
            for (int i = 0; i < size; i++) {
                int sendRows = (i < extraRows) ? rowsPerProcess + 1 : rowsPerProcess;
                for (int j = 0; j < sendRows; j++) {
                    if (i == 0) {
                        System.arraycopy(matrix[offset + j], 0, localMatrix[j], 0, cols);
                    } else {
                        MPI.COMM_WORLD.Send(matrix[offset + j], 0, cols, MPI.DOUBLE, i, 0);
                    }
                }
                offset += sendRows;
            }
        }
    }
}
```

```

    } else {
        for (int i = 0; i < myRows; i++) {
            MPI.COMM_WORLD.Recv(localMatrix[i], 0, cols, MPI.DOUBLE, 0, 0);
        }
    }

    for (int i = 0; i < myRows; i++) {
        double min = Arrays.stream(localMatrix[i]).min().orElse(0);
        double max = Arrays.stream(localMatrix[i]).max().orElse(1);
        for (int j = 0; j < cols; j++) {
            localMatrix[i][j] = (localMatrix[i][j] - min) / (max - min);
        }
    }

    if (rank == 0) {
        int offset = 0;
        for (int i = 0; i < size; i++) {
            int recvRows = (i < extraRows) ? rowsPerProcess + 1 : rowsPerProcess;
            for (int j = 0; j < recvRows; j++) {
                if (i == 0) {
                    System.arraycopy(localMatrix[j], 0, normalizedMatrix[offset + j], 0,
cols);

                    } else {
                        MPI.COMM_WORLD.Recv(normalizedMatrix[offset + j], 0, cols, MPI.DOUBLE,
i, 0);

                    }
                }
                offset += recvRows;
            }
            System.out.println("Normalized Matrix:");
            printMatrix(normalizedMatrix);
        } else {
            for (int i = 0; i < myRows; i++) {
                MPI.COMM_WORLD.Send(localMatrix[i], 0, cols, MPI.DOUBLE, 0, 0);
            }
        }

        MPI.Finalize();
    }

    private static void printMatrix(double[][] matrix) {
        for (double[] row : matrix) {
            System.out.println(Arrays.stream(row)
                .mapToObj(v -> String.format("%.2f", v)) // Округляем до 2 знаков
                .collect(Collectors.joining(", ", "[", "]"))); // Форматируем как массив
        }
    }
}

```

Lab3Task2.java

```
import mpi.MPI;

import java.util.Arrays;
import java.util.stream.Collectors;

public class Lab3Task2 {
    public static void main(String[] args) throws Exception {
        MPI.Init(args);

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

        int rows = 4, cols = 5;
        double[][] matrix = null;
        double[] positiveSumUpperHalf = new double[1];
        double[] negativeSumLowerHalf = new double[1];

        if (rank == 0) {
            matrix = new double[rows][cols];
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    matrix[i][j] = Math.random() * 200 - 100;
                }
            }
            System.out.println("Matrix:");
            printMatrix(matrix);
        }

        int rowsPerProcess = rows / size;
        int extraRows = rows % size;
        int myRows = rank < extraRows ? rowsPerProcess + 1 : rowsPerProcess;
        int offset = rank * rowsPerProcess + Math.min(rank, extraRows);

        double[][] localMatrix = new double[myRows][cols];

        if (rank == 0) {
            int rowOffset = 0;
            for (int i = 0; i < size; i++) {
                int sendRows = (i < extraRows) ? rowsPerProcess + 1 : rowsPerProcess;
                for (int j = 0; j < sendRows; j++) {
                    if (i == 0) {
                        System.arraycopy(matrix[rowOffset + j], 0, localMatrix[j], 0, cols);
                    } else {
                        MPI.COMM_WORLD.Send(matrix[rowOffset + j], 0, cols, MPI.DOUBLE, i, 0);
                    }
                }
                rowOffset += sendRows;
            }
        }
    }
}
```

```

    } else {
        for (int i = 0; i < myRows; i++) {
            MPI.COMM_WORLD.Recv(localMatrix[i], 0, cols, MPI.DOUBLE, 0, 0);
        }
    }

    double localPositiveSum = 0;
    for (int i = 0; i < myRows; i++) {
        int globalRow = offset + i;
        if (globalRow < rows / 2) {
            for (int j = 0; j < cols; j++) {
                if (localMatrix[i][j] > 0) {
                    localPositiveSum += localMatrix[i][j];
                }
            }
        }
    }

    double localNegativeSum = 0;
    for (int i = 0; i < myRows; i++) {
        int globalRow = offset + i;
        if (globalRow >= rows / 2) {
            for (int j = 0; j < cols; j++) {
                if (localMatrix[i][j] < 0) {
                    localNegativeSum += localMatrix[i][j];
                }
            }
        }
    }

    System.out.println("Rank " + rank + " positive sum: " + localPositiveSum);
    System.out.println("Rank " + rank + " negative sum: " + localNegativeSum);

    MPI.COMM_WORLD.Reduce(
        new double[]{localPositiveSum}, 0,
        positiveSumUpperHalf, 0,
        1,
        MPI.DOUBLE,
        MPI.SUM,
        0
    );

    MPI.COMM_WORLD.Reduce(
        new double[]{localNegativeSum}, 0,
        negativeSumLowerHalf, 0,
        1,
        MPI.DOUBLE,
        MPI.SUM,
        0
    );

```

```

        if (rank == 0) {
            double result = positiveSumUpperHalf[0] * negativeSumLowerHalf[0];
            System.out.println("Result (positive sum upper half * negative sum lower half): "
+ result);
        }

        MPI.Finalize();
    }

    private static void printMatrix(double[][] matrix) {
        for (double[] row : matrix) {
            System.out.println(Arrays.stream(row)
                .mapToObj(v -> String.format("%.2f", v))
                .collect(Collectors.joining(", ", "[", "]")));
        }
    }
}

```