

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

**ЛАБОРАТОРНАЯ РАБОТА №4
по дисциплине «Параллельные алгоритмы и системы»
Тема: КОЛЛЕКТИВНЫЕ ФУНКЦИИ**

Студент гр. 1307

Николаев К.Д.

Преподаватель

Санкт-Петербург

2025

Введение

Тема работы: Коллективные функции.

Цель работы: Освоить функции коллективной обработки данных.

Задания:

- 1) Решить задание 2 из лаб. работы 2 с применением коллективных функций (Запустить n процессов и найти по вариантам – Сумму элементов из заданного пользователем диапазона).
- 2) Решить задание 2 из лаб. работы 3 с применением коллективных функций (Среднее арифметическое значение положительных отклонений чисел от последнего элемента матрицы).

```
Process 0 (ID: 22): [ 100; 200] -> localSum = 15150
Process 3 (ID: 23): [ 401; 500] -> localSum = 45050
Process 1 (ID: 25): [ 201; 300] -> localSum = 25050
Process 2 (ID: 24): [ 301; 400] -> localSum = 35050
Total sum: 120300
```

Рисунок 1 - результат работы 1-го задания (количество процессов = 4, интервал = [100; 500]).

```

Matrix:
11  30   8  14  14   3
23  25  16   2   9  18
 3   3  29   2  17  29
 7   8  17   7  25  28
 4   8  21  28  23   9
13   2   1  27  24  25
25   2   6  22  10  16
Process 2 (Thread ID: 23): row= 5, col= 3 -> 27 - 16 = 11
Process 2 (Thread ID: 23): row= 5, col= 4 -> 24 - 16 =  8
Process 2 (Thread ID: 23): row= 5, col= 5 -> 25 - 16 =  9
Process 2 (Thread ID: 23): row= 6, col= 0 -> 25 - 16 =  9
Process 2 (Thread ID: 23): row= 6, col= 3 -> 22 - 16 =  6
Process 0 (Thread ID: 22): row= 0, col= 1 -> 30 - 16 = 14
Process 0 (Thread ID: 22): row= 1, col= 0 -> 23 - 16 =  7
Process 0 (Thread ID: 22): row= 1, col= 1 -> 25 - 16 =  9
Process 0 (Thread ID: 22): row= 1, col= 5 -> 18 - 16 =  2
Process 0 (Thread ID: 22): row= 2, col= 2 -> 29 - 16 = 13
Process 0 (Thread ID: 22): row= 2, col= 4 -> 17 - 16 =  1
Process 1 (Thread ID: 24): row= 3, col= 2 -> 17 - 16 =  1
Process 1 (Thread ID: 24): row= 3, col= 4 -> 25 - 16 =  9
Process 1 (Thread ID: 24): row= 3, col= 5 -> 28 - 16 = 12
Process 1 (Thread ID: 24): row= 4, col= 2 -> 21 - 16 =  5
Process 1 (Thread ID: 24): row= 4, col= 3 -> 28 - 16 = 12
Process 1 (Thread ID: 24): row= 4, col= 4 -> 23 - 16 =  7
Process 0 (Thread ID: 22): row= 2, col= 5 -> 29 - 16 = 13
Global sum: 148,0
Global count: 18
Final average positive deviation: 8,22

```

Рисунок 2 - результат работы 2-го задания (количество процессов = 3).

Ключевые моменты использования коллективных функций в задании 1:

- **Bcast:**

Используется для рассылки диапазона чисел (start и end) от процесса 0 ко всем остальным процессам. Это гарантирует, что все процессы работают с одинаковыми исходными данными.

- **Reduce:**

Используется для сбора локальных сумм (localSum) всех процессов в одну глобальную сумму (globalSum) на процессе 0. Операция выполняет параллельное сложение, что делает ее эффективной для больших объемов данных.

Ключевые моменты использования коллективных функций в задании 2:

- **Bcast:**

Для рассылки последнего элемента матрицы (lastElement) от процесса 0 ко всем процессам.

- **Reduce:**

Сбор локальных сумм положительных отклонений (localSum) в глобальную сумму (globalSum) на процессе 0.

Сбор локальных счетчиков положительных отклонений (localCount) в глобальный счетчик (globalCount) на процессе 0.

- **Scatter:**

Эффективная динамическая рассылка только нужных данных чтобы не пересылать целиком.

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 использованием коллективных функций MPI, таких как Bcast и Reduce, которые позволяют эффективно распределять данные между процессами и собирать результаты для дальнейшей обработки.

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.

Преимущества:

- Обеспечивает согласованность данных: все процессы получают одинаковые значения, что критично для корректной работы программы.
- Минимизирует объем кода и сложность синхронизации, так как данные автоматически доставляются всем участникам.

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.

Преимущества:

- Эффективное агрегирование данных: операция выполняется параллельно, что значительно ускоряет процесс сбора результатов.
- Автоматическое применение заданной операции (например, сложение через MPI.SUM), что упрощает код и снижает вероятность ошибок.

2. Основные выводы

Повышение производительности:

Коллективные функции оптимизированы для параллельных систем и обеспечивают эффективную коммуникацию между процессами, минимизируя время ожидания и синхронизации.

Упрощение кода:

Использование стандартных функций MPI позволяет избежать ручной реализации алгоритмов рассылки и сбора данных, что делает код более чистым и понятным.

Масштабируемость:

Программы, использующие коллективные функции, легко масштабируются на большое количество процессов без значительных изменений в логике работы.

Надежность:

Благодаря встроенной проверке корректности и автоматической синхронизации, использование коллективных функций минимизирует риск ошибок при работе с распределенными системами.

3. Заключение

В ходе выполнения лабораторной работы были изучены и применены коллективные функции MPI.

Приложение А

Task1:

```
package lebibop.lab4;
```

```
import mpi.MPI;
```

```
public class Task1 {
```

```
    public static void main(String[] args) {
```

```
        MPI.Init(args);
```

```
        int rank = MPI.COMM_WORLD.Rank();
```

```
        int size = MPI.COMM_WORLD.Size();
```

```
        int[] range = new int[2];
```

```
        range[0] = Integer.parseInt(System.getProperty("start", "1"));
```

```
        range[1] = Integer.parseInt(System.getProperty("end", "100"));
```

```
        MPI.COMM_WORLD.Bcast(range, 0, 2, MPI.INT, 0);
```

```
        int start = range[0];
```

```
        int end = range[1];
```

```
        int totalNumbers = end - start + 1;
```

```
        int numbersPerProcess = totalNumbers / size;
```

```
        int remainder = totalNumbers % size;
```

```
        int localStart = start + rank * numbersPerProcess + Math.min(rank, remainder);
```

```
        int localEnd = localStart + numbersPerProcess - 1;
```

```
        if (rank < remainder) {
```

```
            localEnd++;
```

```
        }
```

```
        int localSum = 0;
```

```
        if (localStart <= end) {
```

```
            for (int i = localStart; i <= localEnd; i++) {
```



```

        localSum += i;
    }
    System.out.printf("Process %2d (ID: %3d): [%5d; %5d] -> localSum = %d%n",
        rank, Thread.currentThread().getId(), localStart, localEnd, localSum);
} else {
    System.out.printf("Process %2d (ID: %3d): does not participate (no range assigned).%n",
        rank, Thread.currentThread().getId());
}

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

if (rank == 0) {
    System.out.printf("Total sum: %d", globalSum[0]);
}

MPI.Finalize();
}
}

```

Task2:

```
package lebibop.lab4;
```

```
import mpi.MPI;
```

```
import mpi.MPIException;
```

```
import java.util.Random;
```

```

public class Task2 {
    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("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);

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

```

}