Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Дисциплина: Распределенные задачи и алгоритмы**

Работу выполнила: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Ромашкина

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

**Задание:** Реализовать задачу фильтрации, используя неблокирующие обмены и Waitall(). Использовать 4 потока первого уровня.

**Ход работы**

Алгоритм сортировки парного сравнения:

Алгоритм состоит из чередования двух шагов. На первом шаге алгоритма сравниваются пары значений, как показано на рисунке 1.

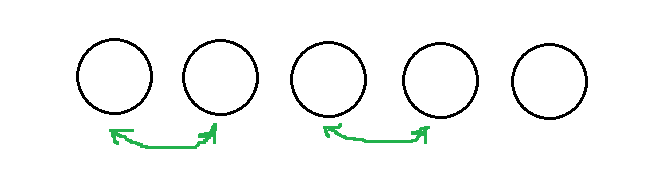


Рисунок 1 – Сортировка без смещения

На втором шаге рассматриваются непересекающиеся пары со смещением равным 1. Схема показана на рисунке 2.

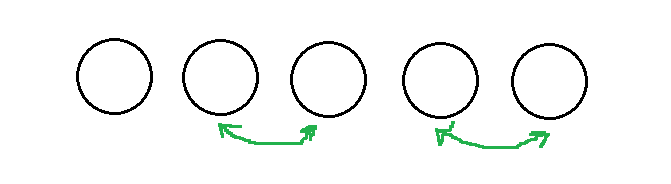


Рисунок 2 – Сортировка со смещением

Если в паре нарушается условие возрастания, элементы меняются местами. Алгоритм заканчивает работу, когда все элементы массива будут отсортированы по возрастанию.

На рисунке 3 показан результат работы программы с использованием 8 потоков. Исходный массив чисел для сортировки равен [14, 5, 9, 0, 80, 3, 1, 15].

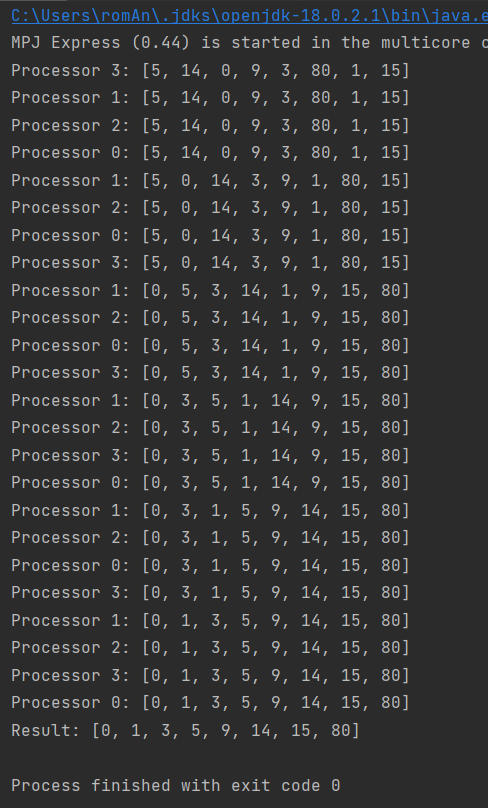


Рисунок 3 - Результат

На рисунке 4 показан результат работы программы с использованием 3 потоков.

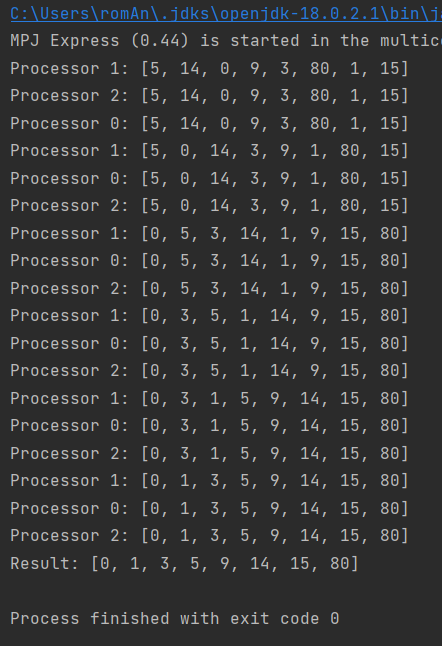


Рисунок 5 - Результат

Ниже представлен листинг программы на языке Java.

public static boolean check\_array(int[] array) {

for (int i = 0; i < array.length - 2; i++)

if (array[i] > array[i + 1]) return true;

return false;

}

public static void sort\_(int[] array, int size, int rank) {

boolean flag = true;

int n = array.length;

if (n > 1) {

int beginning\_selection\_pairs = 1;

int array\_div\_2 = n / 2;

int number\_pairs;

int number\_active\_processors;

int max\_number\_active\_processors;

int[] number\_iterations\_per\_processor;

Request[] req, req1;

Request r;

while (flag) {

beginning\_selection\_pairs = 1 - beginning\_selection\_pairs;

number\_pairs = n % 2 == 0 ? array\_div\_2 - beginning\_selection\_pairs : array\_div\_2;

number\_active\_processors = Math.min(size, number\_pairs);

max\_number\_active\_processors = Math.min(size, array\_div\_2);;

number\_iterations\_per\_processor = new int[number\_active\_processors];

for (int i = 0; i < number\_active\_processors; i++)

number\_iterations\_per\_processor[i] = (i < number\_pairs % number\_active\_processors) ? (number\_pairs / number\_active\_processors + 1) : (number\_pairs / number\_active\_processors);

req = new Request[number\_pairs + 1];

req1 = new Request[max\_number\_active\_processors];

// System.out.println("-----------------------------------------------------");

// System.out.println("flag = " + flag);

// System.out.println("beginning\_selection\_pairs = " + beginning\_selection\_pairs);

// System.out.println("number\_pairs = " + number\_pairs);

// System.out.println("number\_active\_processors = " + number\_active\_processors);

// System.out.println("max\_number\_active\_processors = " + max\_number\_active\_processors);

// System.out.println("number\_iterations\_per\_processor = " + Arrays.toString(number\_iterations\_per\_processor));

if (rank < max\_number\_active\_processors) {

if (rank == 0) {

for (int j = 1; j < number\_active\_processors; j++)

for (int i = 0; i < number\_iterations\_per\_processor[j]; i++)

req[j - 1 + i \* number\_active\_processors] = MPI.COMM\_WORLD.Irecv(array, j \* 2 + i \* number\_active\_processors \* 2 + beginning\_selection\_pairs, 2, MPI.INT, j, 0);

Request.Waitall(req);

for (int i = 0; i < number\_iterations\_per\_processor[0]; i++) {

if (array[i \* number\_active\_processors \* 2 + beginning\_selection\_pairs] > array[i \* number\_active\_processors \* 2 + 1 + beginning\_selection\_pairs]) {

int tmp = array[i \* number\_active\_processors \* 2 + beginning\_selection\_pairs];

array[i \* number\_active\_processors \* 2 + beginning\_selection\_pairs] = array[i \* number\_active\_processors \* 2 + 1 + beginning\_selection\_pairs];

array[i \* number\_active\_processors \* 2 + 1 + beginning\_selection\_pairs] = tmp;

}

}

for (int j = 1; j < max\_number\_active\_processors; j++)

req1[j - 1] = MPI.COMM\_WORLD.Isend(array, 0, n, MPI.INT, j, 0);

Request.Waitall(req1);

} else {

if (rank < number\_active\_processors) {

for (int i = 0; i < number\_iterations\_per\_processor[rank]; i++) {

if (array[rank \* 2 + i \* number\_active\_processors \* 2 + beginning\_selection\_pairs] > array[rank \* 2 + i \* number\_active\_processors \* 2 + 1 + beginning\_selection\_pairs]) {

int tmp = array[rank \* 2 + i \* number\_active\_processors \* 2 + beginning\_selection\_pairs];

array[rank \* 2 + i \* number\_active\_processors \* 2 + beginning\_selection\_pairs] = array[rank \* 2 + i \* number\_active\_processors \* 2 + 1 + beginning\_selection\_pairs];

array[rank \* 2 + i \* number\_active\_processors \* 2 + 1 + beginning\_selection\_pairs] = tmp;

}

r = MPI.COMM\_WORLD.Isend(array, rank \* 2 + i \* number\_active\_processors \* 2 + beginning\_selection\_pairs, 2, MPI.INT, 0, 0);

r.Wait();

}

}

r = MPI.COMM\_WORLD.Irecv(array, 0, n, MPI.INT, 0, 0);

r.Wait();

}

flag = check\_array(array);

System.out.println("Processor " + rank + ": " + Arrays.toString(array));

}

else break;

}

// System.out.println("Processor " + rank + ", flag = " + flag);

}

}

public static void main(String[] args)

throws Exception {

int[] s = { 0 };

int[] data = { 2016 };

int[] buf = { 1, 3, 5 };

int count, TAG = 0;

Status st;

int[] mas = { 14, 5, 9, 0, 80, 3, 1, 15 };

MPI.Init(args);

int rank = MPI.COMM\_WORLD.Rank();

int size = MPI.COMM\_WORLD.Size();

if (size == 1) {

System.out.println("Error!");

MPI.Finalize();

return;

}

sort\_(mas, size,rank);

if (rank == 0)

System.out.println("Result: " + Arrays.toString(mas));

MPI.Finalize();

}

}

Вывод: при выполнении данной лабораторной работы были изучены методы реализации неблокирующих обменов и выполнена задача сортировки с использованием нескольких потоков.