МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Систем и Технологий

Кафедра «Информационные системы»

Дисциплина «Распределенные вычисления и приложения»

**ОТЧЕТ**

**по лаборатороной работе № 2**

Тема Разработка параллельного MPI приложения на языке Java

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_Кузнецов В. И.\_\_\_\_\_\_\_/

подпись инициалы, фамилия

Курс\_\_\_\_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_\_ Группа\_\_\_\_\_\_\_ПИбд-41\_\_\_\_\_\_\_\_\_\_

Направление/специальность\_\_\_09.03.04. Программная инженерия\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель Андреев И.А.

должность, ученая степень, ученое звание

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

фамилия, имя, отчество

Дата сдачи:

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_\_г.

Дата защиты:

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_\_г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

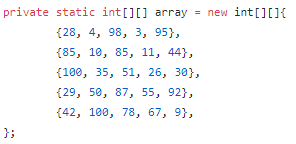
2020 г.

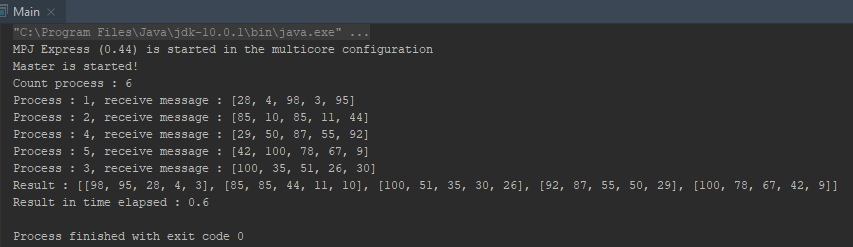
Задание:

Разработать параллельный вариант алгоритма с применением MPI и замерить время его работы.

Вариант: отсортировать строки матрицы по убыванию наибольших элементов.

*Исходная матрица*



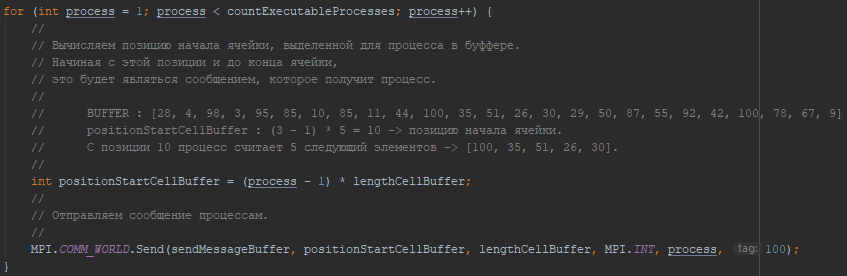


MPI.Init(args); - добавление процесса в Коммуникатор.

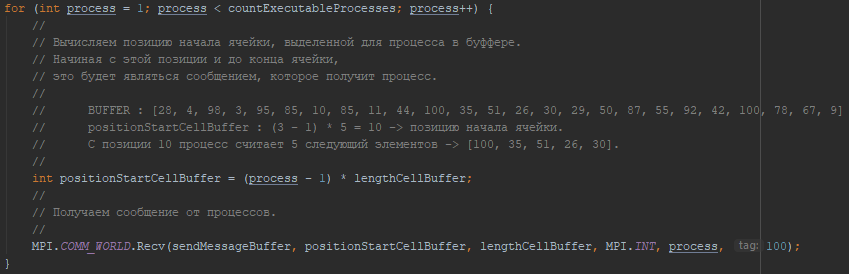
int currentExecutableProcessRank = MPI.COMM\_WORLD.Rank(); - Ранг текущего, запущеного процесса.

int lengthCellBuffer = array[0].length; - Длина ячейки в буффере, выделеной для каждого процесса.

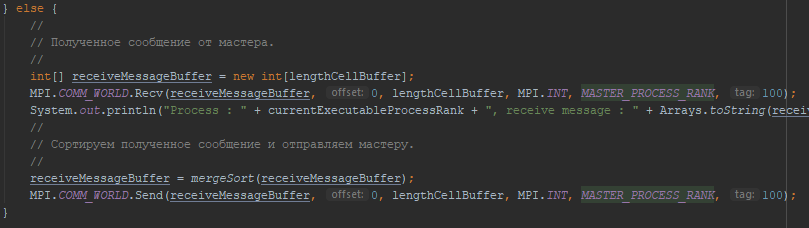
if (currentExecutableProcessRank == MASTER\_PROCESS\_RANK) – если процесс имеет ранг 0, то в таком случае выполняем отправку данных в буффер



Потом читаем данные из буффера и меняем строки матрицы.



Если процесс не является мастером, то принимаем данные из буффера, выполняем сортировку и отправляем данные обратно в буффер.



Листинг:

import mpi.\*;

import java.util.Arrays;

// ЧТО ДЕЛАЕТ ЛИБА?

// По сути она запускает несколько экземпляров написанной программы.

// Кол-во указывается с помощью флага -np в передаваемых аргументах программы.

//

// RUN -> EDIT CONFIGURATION ->

// VM options :

// -jar $MPJ\_HOME$\lib\starter.jar -np 6

// Environment variables :

// MPJ\_HOME=C:/Labs/rvip/rvip\_labs/java/labs\_libs/mpj

//

public class Main {

private static final int MASTER\_PROCESS\_RANK = 0;

private static int[][] array = new int[][]{

{28, 4, 98, 3, 95},

{85, 10, 85, 11, 44},

{100, 35, 51, 26, 30},

{29, 50, 87, 55, 92},

{42, 100, 78, 67, 9},

};

private static int[] matrixToSlice() {

int[] slice = new int[array.length \* array[0].length];

int sliceIndexPosition = 0;

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

for (int j = 0; j < array[i].length; j++) {

slice[sliceIndexPosition++] = array[i][j];

}

}

return slice;

}

private static void updateMatrix(int[] sortedSlice) {

int position = 0;

int row = 0;

for (int i = 0; i < sortedSlice.length; i++) {

if (position == 5) {

row++;

position = 0;

}

array[row][position] = sortedSlice[i];

position++;

}

}

public static void main(String[] args) throws Exception {

MPI.Init(args);

//

// Ранг текущего, запущеного процесса.

//

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

//

// Длина ячейки в буффере, выделеной для каждого процесса.

//

int lengthCellBuffer = array[0].length;

//

if (currentExecutableProcessRank == MASTER\_PROCESS\_RANK) {

//

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

System.out.println("Master is started!");

System.out.println("Count process : " + countExecutableProcesses);

//

// Переваричиваем матрицу в одномерный массив,

// он будет выступать буфером.

//

int[] sendMessageBuffer = matrixToSlice();

//

// Т.к. это мастер, то итерируемся с 1, по всему кол-ву процессов.

//

long start = System.currentTimeMillis();

for (int process = 1; process < countExecutableProcesses; process++) {

//

// Вычисляем позицию начала ячейки, выделенной для процесса в буффере.

// Начиная с этой позиции и до конца ячейки,

// это будет являться сообщением, которое получит процесс.

//

// BUFFER : [28, 4, 98, 3, 95, 85, 10, 85, 11, 44, 100, 35, 51, 26, 30, 29, 50, 87, 55, 92, 42, 100, 78, 67, 9]

// positionStartCellBuffer : (3 - 1) \* 5 = 10 -> позицию начала ячейки.

// С позиции 10 процесс считает 5 следующий элементов -> [100, 35, 51, 26, 30].

//

int positionStartCellBuffer = (process - 1) \* lengthCellBuffer;

//

// Отправляем сообщение процессам.

//

MPI.COMM\_WORLD.Send(sendMessageBuffer, positionStartCellBuffer, lengthCellBuffer, MPI.INT, process, 100);

}

//

// Т.к. это мастер, то итерируемся с 1, по всему кол-ву процессов.

//

for (int process = 1; process < countExecutableProcesses; process++) {

//

// Вычисляем позицию начала ячейки, выделенной для процесса в буффере.

// Начиная с этой позиции и до конца ячейки,

// это будет являться сообщением, которое получит процесс.

//

// BUFFER : [28, 4, 98, 3, 95, 85, 10, 85, 11, 44, 100, 35, 51, 26, 30, 29, 50, 87, 55, 92, 42, 100, 78, 67, 9]

// positionStartCellBuffer : (3 - 1) \* 5 = 10 -> позицию начала ячейки.

// С позиции 10 процесс считает 5 следующий элементов -> [100, 35, 51, 26, 30].

//

int positionStartCellBuffer = (process - 1) \* lengthCellBuffer;

//

// Получаем сообщение от процессов.

//

MPI.COMM\_WORLD.Recv(sendMessageBuffer, positionStartCellBuffer, lengthCellBuffer, MPI.INT, process, 100);

}

//

long finish = System.currentTimeMillis();

float timeElapsed = (finish - start) / 100F;

//

updateMatrix(sendMessageBuffer);

System.out.println("Result : " + Arrays.deepToString(array));

System.out.println("Result in time elapsed : " + timeElapsed);

} else {

//

// Полученное сообщение от мастера.

//

int[] receiveMessageBuffer = new int[lengthCellBuffer];

MPI.COMM\_WORLD.Recv(receiveMessageBuffer, 0, lengthCellBuffer, MPI.INT, MASTER\_PROCESS\_RANK, 100);

System.out.println("Process : " + currentExecutableProcessRank + ", receive message : " + Arrays.toString(receiveMessageBuffer));

//

// Сортируем полученное сообщение и отправляем мастеру.

//

receiveMessageBuffer = mergeSort(receiveMessageBuffer);

MPI.COMM\_WORLD.Send(receiveMessageBuffer, 0, lengthCellBuffer, MPI.INT, MASTER\_PROCESS\_RANK, 100);

}

MPI.Finalize();

}

private static int[] mergeSort(int[] input) {

if (input.length <= 1) {

return input;

}

//

int middle = input.length / 2;

int[] left = new int[middle];

int[] right = new int[input.length - middle];

//

for (int i = 0; i < middle; i++) {

left[i] = input[i];

}

for (int i = middle; i < input.length; i++) {

right[i - middle] = input[i];

}

//

return merge(mergeSort(right), mergeSort(left));

}

private static int[] merge(int[] right, int[] left) {

//

int size = right.length + left.length;

int i = right.length - 1;

int j = left.length - 1;

int[] sorted = new int[size];

//

for (int k = size - 1; k >= 0; k--) {

if (i == -1) {

sorted[k] = left[j];

j--;

continue;

}

if (j == -1) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] > left[j]) {

sorted[k] = left[j];

j--;

continue;

}

if (right[i] < left[j]) {

sorted[k] = right[i];

i--;

continue;

}

if (right[i] == left[j]) {

sorted[k] = left[j];

j--;

}

}

return sorted;

}

}