НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА АВТОМАТИЗАЦІЇ СИСТЕМ ОБРОБКИ ІНФОРМАЦІЇ ТА УПРАВЛІННЯ

Лабораторна робота №8

з дисципліни **«**Паралельне програмування**»**

Виконав:

студент 3 курсу

ФІОТ гр. ІП-31

Кахерський О.І.

Перевірив:

Корочкін О. В.

Київ – 2016 р.

**Тема:** Програмування для комп’ютерних систем з локальною пам’яттю. Бібліотека MPI.

**Технічне завдання:** Розробити програму для розв’язання ПКС із СП (структура на рис. 1) математичної задачі: A =

**Бібліотека: MPI.**

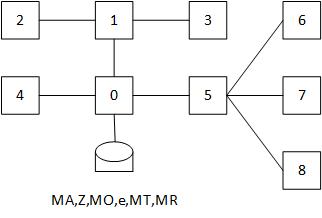


Рис.1 Структурна схема ПКС

**Виконання роботи**

**Етап 1.** Побудова паралельного алгоритму

2. =

**Етап 2. Розробка алогоритмів роботи кожного процесу**

i = (1 .. 8)

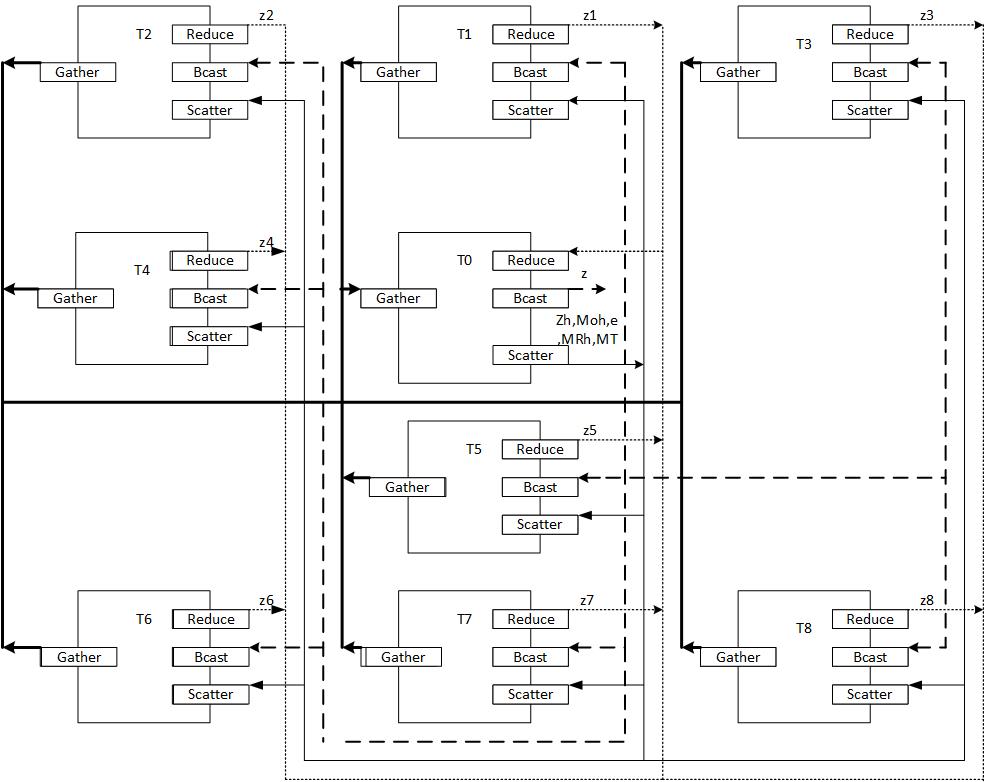
**Задача Т0**

* 1. Ввід МО, Z, МT, MR, e;
  2. упакувати , ,, MT, ;
  3. надіслати пакети всім процесам (враховуючи топологію);
  4. обчислити z0 = max(
  5. обчислити z = max(z,z0);
  6. Отримати zj від процесів Ti;
  7. Обчислити z = max(z,zi);
  8. Надіслати z всім процесам (враховуючи топологію);
  9. Обчислити =
  10. Отримати від всіх процесів
  11. Вивід MA

**Задача Тi**

1. Отримати упаковані, надіслані T0;
2. Розпакувати дані;
3. Обчислити zi = max(z,zi);
4. Надіслати zi процесу T0 (враховуючи топологію);
5. Отримати z від процесу T0;
6. Обчислити =
7. Надіслати процесу Т0 (враховуючи топологію).

**Етап 3. Розробка структурної схеми взаємодії задач**



**Етап 4. Лістинг коду**

import mpi.Graphcomm;

import mpi.MPI;

import java.util.Arrays;

/\*\*

\* Created by Oleh Kakherskyi, student of the KPI, FICT, IP-31 group (olehkakherskiy@gmail.com) on 21.04.2016.

\*/

**public class Main {**

private static int N;

private static int H;

private static int[][] MO, MR, MT, MA;

private static int[] Z;

private static int e, maxZ;

public static void main(String[] args) {

//размерность матриц и векторов

N = Integer.parseInt(args[args.length - 1]);

int p = 9;

H = N / p;

//инициализация библиотеки

MPI.Init(args);

//ранг процесса

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

//инициализация данных для топологии

int[] graphIndexes = new int[]{3, 5, 6, 7, 8, 9, 10, 11, 12};

int[] graphEdges = new int[]{1, 4, 5, 0, 2, 3, 1, 1, 0, 0, 6, 7, 8, 5, 5, 5};

//создание топологии на основе коммуникатора COMM\_WORLD

Graphcomm graphcomm = MPI.COMM\_WORLD.Create\_graph(graphIndexes, graphEdges, false);

//кол-во элементов в массивах и строк в матрицах, которые используются каждым рангом

int[] indexCount = getEachCount(p);

//смещение, с которого отсчитываются indexCount элементов для каждого ранга

int[] displacement = getEachOffset(p);

//ввод данных

if (rank == 0) {

MO = MatrixOperations.inputMatrix(N);

MR = MatrixOperations.inputMatrix(N);

MT = MatrixOperations.inputMatrix(N);

Z = MatrixOperations.inputVector(N);

e = MatrixOperations.inputConstant();

//установка метаданных для упаковщика

DataPackBuilder.setMetadata(p, indexCount, displacement);

}

//если ранг = 0, то упаковываются все матрицы, вектора и константы для отправки каждому процессу,

//в ином случае создается буфер, в который будет помещен принятый пакет

DataPack[] packs = rank == 0 ? DataPackBuilder.packData(MO, MT, MR, Z, e) : new DataPack[1];

//отправка пакетов всем процессам с процесса 0 согласно топологии

graphcomm.Scatter(packs, 0, 1, MPI.OBJECT, packs, 0, 1, MPI.OBJECT, 0);

//распаковка данных

MO = packs[0].getMO();

MR = packs[0].getMR();

MT = packs[0].getMT();

Z = packs[0].getVector();

e = packs[0].getE();

//вычисление локального максимума

int[] localMax = {MatrixOperations.max(Z, 0, Z.length)};

//принятие/передача локальных максимумов с автоматическим максимального значения

graphcomm.Reduce(localMax, 0, localMax, 0, 1, MPI.INT, MPI.MAX, 0);

//отправка/получение максимального значения всеми с процесса 0 всем процессам

graphcomm.Bcast(localMax, 0, 1, MPI.INT, 0);

maxZ = localMax[0];

//вычисление математического выражения

MA = MatrixOperations.addMatrix(MO, MatrixOperations.multMatrix(MT, MR), maxZ, e);

int[][] buf = new int[N][];

//отравка частичного результата процессу 0 всеми процессами с последущей автоматической упаковкой по номеру процесса

graphcomm.Gather(MA, 0, indexCount[rank], MPI.OBJECT, buf, displacement[rank], indexCount[rank], MPI.OBJECT, 0);

MA = buf;

if (rank == 0 && MA.length <= 18)

System.out.println(MatrixOperations.formattedDeepToString(MA));

System.out.println("Process " + rank + " is finished");

MPI.Finalize();

}

/\*\*

\* Вычисление количества элементов в векторах и кол-во строк в матрицах, которые будут переданы каждому процессу

\* для вычисления мат. выражения

\*

\* @param processCount количество процессов

\* @return количество элементов

\*/

private static int[] getEachCount(int processCount) {

int[] result = new int[processCount];

Arrays.fill(result, 1, result.length, H);

//первый процесс получит больше элементов, если их неравное количество на каждый процесс

result[0] = N - (result.length - 1) \* H;

return result;

}

/\*\*

\* Вычисление смещения в векторах и матрицах, начиная с которого будет передано {@link #getEachCount(int)}(rank)

\* элементов векторов и строк матриц процессу с номером rank

\*

\* @param processCount количество процессов

\* @return смещение в векторах и матрицах

\*/

private static int[] getEachOffset(int processCount) {

int[] result = new int[processCount];

result[0] = 0;

result[1] = N - (result.length - 1) \* H;

for (int i = 2; i < result.length; i++) {

result[i] = result[i - 1] + H;

}

return result;

}

}

import java.io.Serializable;

/\*\*

\* Created by Oleh Kakherskyi, student of the KPI, FICT, IP-31 group (olehkakherskiy@gmail.com) on 18.04.2016.

\*/

**public class DataPack implements Serializable** {

private int[][] MO;

private int[][] MR;

private int[][] MT;

private int[] vector;

private int e;

public DataPack() {

MO = new int[0][0];

MR = new int[0][0];

MT = new int[0][0];

vector = new int[0];

}

public int[][] getMO() {

return MO;

}

public void setMO(int[][] MO) {

this.MO = MO;

}

public int[][] getMR() {

return MR;

}

public void setMR(int[][] MR) {

this.MR = MR;

}

public int[][] getMT() {

return MT;

}

public void setMT(int[][] MT) {

this.MT = MT;

}

public int[] getVector() {

return vector;

}

public void setVector(int[] vector) {

this.vector = vector;

}

public int getE() {

return e;

}

public void setE(int e) {

this.e = e;

}

}

import java.util.Arrays;

/\*\*

\* Created by Oleh Kakherskyi, student of the KPI, FICT, IP-31 group (olehkakherskiy@gmail.com) on 18.04.2016.

\*/

**public class DataPackBuilder** {

private static DataPack[] preparedPacks;

public static int[] elementsCount;

public static int[] dataOffset;

public static void setMetadata(int processCount, int[] elemCount, int[] offset) {

preparedPacks = new DataPack[processCount];

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

preparedPacks[i] = new DataPack();

}

elementsCount = elemCount;

dataOffset = offset;

}

public static DataPack[] packData(int[][] MO, int[][] MT, int[][] MR, int[] Z, int e) {

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

preparedPacks[i].setMO(Arrays.copyOfRange(MO, dataOffset[i], dataOffset[i] + elementsCount[i]));

preparedPacks[i].setMR(MR);

preparedPacks[i].setMT(Arrays.copyOfRange(MT, dataOffset[i], dataOffset[i] + elementsCount[i]));

preparedPacks[i].setVector(Arrays.copyOfRange(Z, dataOffset[i], dataOffset[i] + elementsCount[i]));

preparedPacks[i].setE(e);

}

return preparedPacks;

}

}

import java.util.Arrays;

/\*\*

\* Created by oleg on 03.10.15.

\*/

**public class MatrixOperations** {

/\*\*

\* Генерує матрицю, заповнює одиницями

\*

\* @param n розмірність

\*/

public static int[][] inputMatrix(int n) {

int[][] result = new int[n][n];

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++) {

result[i][j] = 1;

}

return result;

}

/\*\*

\* Генерує вектор, заповнює одиницями

\*

\* @param n розмірність

\*/

public static int[] inputVector(int n) {

int[] result = new int[n];

// int start = 4;

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

result[i] = 1;

}

return result;

}

public static int inputConstant() {

return 1;

}

/\*\*

\* Додає матриці

\*

\* @param param1 матриця 1

\* @param param2 матриця 2

\* @return сума матриць

\*/

public static int[][] addMatrix(int[][] param1, int[][] param2, int const1, int const2) {

if (param1.length != param2.length) {

System.out.println("Нельзя суммировать матрицы с разным количеством строк");

return null;

}

int[][] result = new int[param1.length][param1[0].length];

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

result[i] = addVectors(param1[i], param2[i], const1, const2);

}

return result;

}

/\*\*

\* Додає вектори

\*

\* @param p1 вектор1

\* @param p2 вектор2

\* @return сума векторів

\*/

public static int[] addVectors(int[] p1, int[] p2, int const1, int const2) {

int[] result = new int[p1.length];

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

result[i] = p1[i] \* const1 + p2[i] \* const2;

}

return result;

}

/\*\*

\* Перемножає матриці

\*

\* @param param1 матриця-множник

\* @param param2 матриця-множене

\* @return добуток матриць, розмірність - nxn

\*/

public static int[][] multMatrix(int[][] param1, int[][] param2) {

if (param1[0].length != param2.length) {

System.out.println("Нельзя умножать матрицы, количество элементов в строке которой не равно кол-ву столбцов в другой");

return null;

}

int[][] result = new int[param1.length][param1[0].length];

for (int k = 0; k < param1.length; k++) {

for (int i = 0; i < param1[0].length; i++) {

for (int j = 0; j < param2.length; j++) {

result[k][i] += param1[k][j] \* param2[j][i];

}

}

}

return result;

}

public static String formattedDeepToString(int[][] matrix) {

StringBuilder result = new StringBuilder();

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

result.append(Arrays.toString(matrix[i])).append("\n");

}

return result.toString();

}

public static int max(int[] vector, int startIndex, int endIndex) {

int result = vector[startIndex];

for (int i = startIndex + 1; i < endIndex; i++) {

result = result < vector[i] ? vector[i] : result;

}

return result;

}

/\*\*

\* Метод выполняет обрезку строк матрицы. Возвращает строки в matrixToTruncate начиная с offset количеством rowCount

\*

\* @param matrixToTruncate матрица, с которой будут вырезаны строки

\* @param offset номер строки, с которой будет выполнена обрезка

\* @param rowCount количество строк

\* @return матрица, количество строк которой = rowCount и строки идентичны строкам в matrixToTruncate начиная с

\* offset позиции

\*/

public static int[][] truncateMatrix(int[][] matrixToTruncate, int offset, int rowCount) {

int result[][] = new int[rowCount][];

System.arraycopy(matrixToTruncate, offset, result, 0, rowCount);

return result;

}

}