**Додаток Ж. Лістинг коду програми ПРГ2**

import mpi.Graphcomm;  
import mpi.MPI;  
  
import java.util.Arrays;  
  
*/\*\*  
 \* Математичне завдання: MA = MB(MC+MO)a+min(Z)\*MK  
 \* Бібліотека: MPI  
 \* Виконав: Кахерський О.І., ІП-31, ФІОТ  
 \*/*public class Main {  
  
 private static int[][] *MA*, *MB*, *MO*, *MC*, *MK*;  
 private static int[] *Z*;  
 public static int *a*, *minZ*, *N*, *H*;  
 private static int *p*;  
  
 public static void main(String[] args) {  
 long startTime = System.*currentTimeMillis*();  
 long endTime = 0;  
 MPI.*Init*(args);  
  
 int rank = MPI.*COMM\_WORLD*.Rank();  
 int size = MPI.*COMM\_WORLD*.Size();  
  
 *//определяем размерность матриц - передана как последний параметр командной строки  
 N* = Integer.*parseInt*(args[args.length - 1]);  
 *H* = *N* / size;  
 *p* = size - 1;  
  
 if (size == 1) {  
 *initData*(rank);  
 *MA* = MatrixOperations.*addMatrix*(  
 MatrixOperations.*multMatrix*(MatrixOperations.*addMatrix*(*MC*, *MO*, 1, 1), *MB*), *MK*, *a*, *minZ*);  
 if (*MA*.length <= 16)  
 System.*out*.println(MatrixOperations.*formattedDeepToString*(*MA*));  
 System.*out*.println("Computation time (sec): " + ((System.*currentTimeMillis*() - startTime) / 1000.0));  
 return;  
 }  
  
 System.*out*.println("Process " + rank + " is started");  
  
 int power = (int) (Math.*log*(size) / Math.*log*(2)); *//степень каждой вершины  
  
 //создаем топологию на основе глобального коммуникатора - каждый процесс будет знать своих соседей  
 // (необходимо для передачи данных), таким образом реализовываем гиперкуб* Graphcomm graphcomm = MPI.*COMM\_WORLD*.Create\_graph(GraphTopology.*calculateIndexes*(power, size), GraphTopology.*calculateEdges*(power, size), false);  
  
 *//ввод данных  
 initData*(rank);  
  
 int[] indexCount = *getEachCount*(size);  
 int[] displacement = *getEachOffset*(size);  
  
 if (rank == 0 || rank == *p*) {  
 *//устанавливаем метаданные для упаковки данных* DataPackBuilder.*setMetadata*(size, *getEachCount*(size), *getEachOffset*(size));  
 }  
  
 *//упаковываем данные для отправки* DataPack[] sendPacks;  
 if (rank == 0)  
 sendPacks = DataPackBuilder.*packData*(*MB*, *MO*, null, *a*, true);  
 else if (rank == *p*)  
 sendPacks = DataPackBuilder.*packData*(*MC*, *MK*, *Z*, 0, false);  
 else sendPacks = new DataPack[0];  
  
 *//указываем кол-во отсылаемых элементов каждому процессу (1 и последний процесс отсылают по пакету на процесс,  
 // остальные - не отсылают данные)* int[] sendCount = new int[size];  
  
 *//смещение в отсылаемом буффере для каждого процесса* int[] sendDisplacement = new int[size];  
  
 if (rank == 0 || rank == *p*) {  
  
 *//1 и последний процесс отсылают по пакету на процесс* Arrays.*fill*(sendCount, 1);  
 *//указываем смещение в пакетах (1 процесс начиная с 0 позиции получает 1 пакет, 2 - начиная с первой  
 // позиции получает 1 пакет и тд)* for (int i = 0; i < sendDisplacement.length; i++) {  
 sendDisplacement[i] = i;  
 }  
 }  
  
 int[] receiveCount = new int[size];  
 receiveCount[0] = 1;  
 receiveCount[*p*] = 1;  
  
 int[] receiveDisplacement = new int[size];  
 receiveDisplacement[0] = 0;  
 receiveDisplacement[*p*] = 1;  
  
 *// массив полученых пакетов* DataPack[] receivePacks = new DataPack[2];  
 graphcomm.Alltoallv(sendPacks, 0, sendCount, sendDisplacement, MPI.*OBJECT*,  
 receivePacks, 0, receiveCount, receiveDisplacement, MPI.*OBJECT*);  
 *//распаковка данных с 0 процесса  
 MO* = receivePacks[0].getMatrix2();  
 *MB* = receivePacks[0].getMatrix1();  
 *a* = receivePacks[0].getConstant();  
  
 *//распаковка данных с p процесса  
 MC* = receivePacks[1].getMatrix1();  
 *MK* = receivePacks[1].getMatrix2();  
 *Z* = receivePacks[1].getVector();  
  
  
 *//вычисление локальных минимумов* int[] localMinimums = new int[1];  
 localMinimums[0] = MatrixOperations.*min*(*Z*, 0, *Z*.length);  
  
 long s1 = System.*currentTimeMillis*();  
 *//вычисление глобального минимума* graphcomm.Reduce(localMinimums, 0, localMinimums, 0, 1, MPI.*INT*, MPI.*MIN*, *p*);  
  
 *//расшаривание глобального минимума* graphcomm.Bcast(localMinimums, 0, 1, MPI.*INT*, *p*);  
 *minZ* = localMinimums[0];  
 *//вычисление частичных результатов задания  
 MA* = MatrixOperations.*addMatrix*(  
 MatrixOperations.*multMatrix*(MatrixOperations.*addMatrix*(*MC*, *MO*, 1, 1), *MB*), *MK*, *a*, *minZ*);  
  
 *//сборка частичных результатов мат. выражения со всех процессов в процесс 0* int[][] result = new int[*N*][];  
 graphcomm.Gatherv(*MA*, 0, indexCount[rank], MPI.*OBJECT*, result, 0, indexCount, displacement, MPI.*OBJECT*, 0);  
  
 *//вывод результатов на экран и завершение работы* if (rank == 0) {  
 if (result.length <= 16)  
 System.*out*.println(MatrixOperations.*formattedDeepToString*(result));  
 endTime = System.*currentTimeMillis*();  
 System.*out*.println("Computation time (sec): " + ((endTime - startTime) / 1000.0));  
 }  
 System.*out*.println("Process " + rank + " is finished");  
 MPI.*Finalize*();  
 }  
  
 private static void initData(int rank) {  
 if (rank == 0) {  
 *MB* = MatrixOperations.*inputMatrix*(*N*);  
 *MO* = MatrixOperations.*inputMatrix*(*N*);  
 *a* = MatrixOperations.*inputConstant*();  
  
 }  
 if (rank == *p*) {  
 *MC* = MatrixOperations.*inputMatrix*(*N*);  
 *MK* = MatrixOperations.*inputMatrix*(*N*);  
 *Z* = MatrixOperations.*inputVector*(*N*);  
 }  
 }  
  
 */\*\*  
 \* 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;  
 }  
}

/\*\*

\* Created by Oleh Kakherskyi, student of the KPI, FICT, IP-31 group (olehkakherskiy@gmail.com) on 16.04.2016.

\*/

public class GraphTopology {

/\*\*

\* Возврат массива степеней каждой вершины. Так как у нас k-мерный гиперкуб, то степени всех вершин одинаковы

\*

\* @param k степень выхода каждой вершины

\* @param n количество вершин

\* @return

\*/

public static int[] calculateIndexes(int k, int n) {

int[] result = new int[n];

for (int i = 1; i <= result.length; i++) {

result[i - 1] = k \* i;

}

return result;

}

public static int[] calculateEdges(int k, int n) {

int[][] buffer = new int[n][k];

for (int[] aResult : buffer) {

Arrays.fill(aResult, -1);

}

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

int neighbourBaseNumber = (int) Math.pow(2, i); //на сколько отличается номер соседей

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

if (buffer[j][i] == -1) {

buffer[j][i] = j + neighbourBaseNumber;

buffer[j + neighbourBaseNumber][i] = j;

}

}

int[] result = new int[k \* n];

//формируем массив edge

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

System.arraycopy(buffer[i], 0, result, k \* i, k);

}

// System.out.println(Arrays.toString(result));

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[][] matrix1;

private int[][] matrix2;

private int[] vector;

public DataPack() {

matrix1 = new int[0][0];

matrix2 = new int[0][0];

vector = new int[0];

}

public int getConstant() {

return constant;

}

public void setConstant(int constant) {

this.constant = constant;

}

public int[] getVector() {

return vector;

}

public void setVector(int[] vector) {

this.vector = vector;

}

public int[][] getMatrix2() {

return matrix2;

}

public void setMatrix2(int[][] matrix2) {

this.matrix2 = matrix2;

}

public int[][] getMatrix1() {

return matrix1;

}

public void setMatrix1(int[][] matrix1) {

this.matrix1 = matrix1;

}

private int constant;

}

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[][] matrix1, int[][] matrix2, int[] vector, int constant, boolean matrix1FullSize) {

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

if (matrix1FullSize)

preparedPacks[i].setMatrix1(matrix1);

else

preparedPacks[i].setMatrix1(Arrays.copyOfRange(matrix1, dataOffset[i], dataOffset[i] + elementsCount[i]));

preparedPacks[i].setMatrix2(Arrays.copyOfRange(matrix2, dataOffset[i], dataOffset[i] + elementsCount[i]));

if (vector != null)

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

preparedPacks[i].setConstant(constant);

}

return preparedPacks;

}

}

/\*\*

\* 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] = start + i;

}

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 min(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;

}

}