**Додаток В**

**Лістинг програми для системи зі спільною пам’яттю**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ПРО-2 \*

\* Курсова робота. Система с СП \*

\* MA = (MX \* MC) \* ME - MM \* MD \*

\* Захожий Ігор, група ІО-73 \*

\* 06.05.2010 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.util.Scanner;

public class Program {

public static void main(String[] args) {

int N;

int P;

int H;

Scanner input = new Scanner(System.in);

System.out.println("Please, enter the count of threads:");

System.out.print(">> ");

P = input.nextInt();

System.out.println("Please, enter the size of matrixes:");

System.out.print(">> ");

N = input.nextInt();

if (N < P) {

P = N;

}

H = N / P;

SynchronizationMonitor synchronizationMonitor = new SynchronizationMonitor(P);

DataMonitor dataMonitor = new DataMonitor();

Matrix MX = new Matrix(N);

Matrix MM = new Matrix(N);

Matrix MA = new Matrix(N);

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

CalcThread Ti = new CalcThread(i, synchronizationMonitor, dataMonitor, N, P, H, MX, MM, MA);

Ti.start();

}

}

}

public class SynchronizationMonitor {

private int FInput;

private int FCalc;

private int P;

public SynchronizationMonitor(int P) {

FInput = 0;

FCalc = 0;

this.P = P;

}

public synchronized void signalInput() {

FInput++;

notifyAll();

}

public synchronized void waitInput() {

if (FInput < 1) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public synchronized void signalCalc() {

FCalc++;

if (FCalc == (P - 1)) {

notify();

}

}

public synchronized void waitCalc() {

if (FCalc < (P - 1)) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class DataMonitor {

private Matrix MC;

private Matrix ME;

private Matrix MD;

public DataMonitor(){}

public synchronized Matrix getMC() {

return MC.copy();

}

public synchronized void setMC(Matrix MC) {

this.MC = MC.copy();

}

public synchronized Matrix getME() {

return ME.copy();

}

public synchronized void setME(Matrix ME) {

this.ME = ME.copy();

}

public synchronized Matrix getMD() {

return MD.copy();

}

public synchronized void setMD(Matrix MD) {

this.MD = MD.copy();

}

}

/////////////////////////////////////////////////////////////////////

public class CalcThread extends Thread {

private int Tid;

private SynchronizationMonitor synchronizationMonitor;

private DataMonitor dataMonitor;

private int N;

private int P;

private int H;

private Matrix MX;

private Matrix MM;

private Matrix MA;

private Matrix MCi;

private Matrix MEi;

private Matrix MDi;

public CalcThread(int Tid, SynchronizationMonitor synchronizationMonitor,

DataMonitor dataMonitor, int N, int P, int H,

Matrix MX, Matrix MM, Matrix MA) {

this.Tid = Tid;

this.synchronizationMonitor = synchronizationMonitor;

this.dataMonitor = dataMonitor;

this.N = N;

this.P = P;

this.H = H;

this.MX = MX;

this.MM = MM;

this.MA = MA;

MCi = new Matrix(N);

MEi = new Matrix(N);

MDi = new Matrix(N);

}

private Matrix multiplyMatrixesH(Matrix MM1, Matrix MM2, int Hfrom, int Hto) {

Matrix result = new Matrix(Hto - Hfrom, MM2.getColCount());

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

for (int j = 0; j < result.getColCount(); j++) {

for (int k = 0; k < MM1.getColCount(); k++) {

result.set(i, j,

result.get(i, j) + MM1.get(i + Hfrom, k) \* MM2.get(k, j));

}

}

}

return result;

}

public void run() {

System.out.println("Thread T" + String.valueOf(Tid + 1) + "has started.");

if (Tid == 0) {

MX.fillWithOnes();

MM.fillWithOnes();

MCi.fillWithOnes();

MEi.fillWithOnes();

MDi.fillWithOnes();

dataMonitor.setMC(MCi);

dataMonitor.setMD(MDi);

dataMonitor.setME(MEi);

synchronizationMonitor.signalInput();

}

else {

synchronizationMonitor.waitInput();

MCi = dataMonitor.getMC();

MDi = dataMonitor.getMD();

MEi = dataMonitor.getME();

}

System.out.println("Thread T" + String.valueOf(Tid + 1) +

"has begun to calculate the function.");

int Hfrom = Tid \* H;

int Hto;

if (Tid == (P - 1)) {

Hto = N;

}

else {

Hto = Tid \* H + H;

}

Matrix temp1 = multiplyMatrixesH(multiplyMatrixesH(MX, MCi, Hfrom, Hto),

MEi, 0, (Hto - Hfrom));

Matrix temp2 = multiplyMatrixesH(MM, MDi, Hfrom, Hto);

for (int i = Hfrom; i < Hto; i++) {

for (int j = 0; j < MA.getColCount(); j++) {

MA.set(i, j, temp1.get((i - Hfrom), j) - temp2.get((i - Hfrom), j));

}

}

System.out.println("Thread T" + String.valueOf(Tid + 1) +

"has ended to calculate the function.");

if (Tid == 0) {

synchronizationMonitor.waitCalc();

System.out.println("MA = (");

for (int i = 0; i < MA.getRowCount(); i++) {

for (int j = 0; j < MA.getColCount(); j++) {

System.out.print(String.valueOf(MA.get(i, j)) + "\t");

}

System.out.println();

}

System.out.println(")");

}

else {

synchronizationMonitor.signalCalc();

}

System.out.println("Thread T" + String.valueOf(Tid + 1) + "has finished.");

}

}

/////////////////////////////////////////////////////////////////////

public class Matrix {

private long[][] m;

public Matrix(int n) {

m = new long[n][];

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

m[i] = new long[n];

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

m[i][j] = 0;

}

}

}

public Matrix(int rows, int cols) {

m = new long[rows][];

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

m[i] = new long[cols];

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

m[i][j] = 0;

}

}

}

public long get(int i, int j) {

return m[i][j];

}

public void set(int i, int j, long e) {

m[i][j] = e;

}

public int getRowCount() {

return m.length;

}

public int getColCount() {

return m[0].length;

}

public void fillWithOnes() {

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

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

m[i][j] = 1;

}

}

}

public Matrix copy() {

Matrix copyMM = new Matrix(m.length, m[0].length);

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

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

copyMM.set(i, j, m[i][j]);

}

}

return copyMM;

}

}

**Лістинг програми для системи з локальною пам’яттю**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ПРО-2 \*

\* Курсова робота. Система с ЛП \*

\* MA = (MX \* MC) \* ME - MM \* MD \*

\* Захожий Ігор, група ІО-73 \*

\* 07.05.2010 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "stdafx.h"

#include <iostream>

#include <stdlib.h>

#include "mpi.h"

#include "Matrix.h"

using namespace std;

int N = 8;

int P;

int H;

Matrix multiplyMatrixesH(Matrix MM1, Matrix MM2, int Hfrom, int Hto)

{

Matrix result(Hto - Hfrom, MM2.getColCount());

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

{

for (int j = 0; j < result.getColCount(); j++)

{

for (int k = 0; k < MM1.getColCount(); k++)

{

result.set(i, j,

result.get(i, j) + MM1.get(i + Hfrom, k) \* MM2.get(k, j));

}

}

}

return result;

}

/////////////////////////////////////////////////////////////////////////

void T0(int Tid)

{

MPI\_Status status;

Matrix MA(N);

Matrix MX(N);

Matrix MC(N);

Matrix ME(N);

Matrix MM(N);

Matrix MD(N);

MX.fillWithOnes();

MC.fillWithOnes();

ME.fillWithOnes();

MM.fillWithOnes();

MD.fillWithOnes();

int buf1Size = MC.getSize() + ME.getSize() + MD.getSize();

long\* buf1 = new long[buf1Size];

MC.writeToBufer(buf1);

long\* ptr = (long\*) ((size\_t) buf1 + MC.getSize() \* sizeof(long));

ME.writeToBufer(ptr);

ptr = (long\*) ((size\_t) ptr + ME.getSize() \* sizeof(long));

MD.writeToBufer(ptr);

ptr = (long\*) ((size\_t) ptr + MD.getSize() \* sizeof(long));

int buf2Size;

long\* buf2;

int Trec = P / 2;

int prevTrec = P;

while (Trec >= 4)

{

buf2Size = (MX.getColCount() + MM.getColCount()) \* H \* (prevTrec - Trec);

buf2 = new long[buf2Size];

ptr = buf2;

for (int i = H \* Trec; i < (prevTrec \* H); i++)

{

for (int j = 0; j < MX.getColCount(); j++)

{

\*ptr = MX.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

for (int i = H \* Trec; i < (prevTrec \* H); i++)

{

for (int j = 0; j < MM.getColCount(); j++)

{

\*ptr = MM.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, Trec, Trec, MPI\_COMM\_WORLD);

MPI\_Send((void\*) buf2, buf2Size, MPI\_LONG, Trec, Trec, MPI\_COMM\_WORLD);

delete[] buf2;

prevTrec = Trec;

Trec /= 2;

}

buf2Size = (MX.getColCount() + MM.getColCount()) \* H \* 2;

buf2 = new long[buf2Size];

ptr = buf2;

for (int i = H; i < (3 \* H); i++)

{

for (int j = 0; j < MX.getColCount(); j++)

{

\*ptr = MX.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

for (int i = H; i < (3 \* H); i++)

{

for (int j = 0; j < MM.getColCount(); j++)

{

\*ptr = MM.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, 1, 1, MPI\_COMM\_WORLD);

MPI\_Send((void\*) buf2, buf2Size, MPI\_LONG, 1, 1, MPI\_COMM\_WORLD);

delete[] buf2;

buf2Size = (MX.getColCount() + MM.getColCount()) \* H;

buf2 = new long[buf2Size];

ptr = buf2;

for (int i = 3 \* H; i < (4 \* H); i++)

{

for (int j = 0; j < MX.getColCount(); j++)

{

\*ptr = MX.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

for (int i = 3 \* H; i < (4 \* H); i++)

{

for (int j = 0; j < MM.getColCount(); j++)

{

\*ptr = MM.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, 3, 3, MPI\_COMM\_WORLD);

MPI\_Send((void\*) buf2, buf2Size, MPI\_LONG, 3, 3, MPI\_COMM\_WORLD);

delete[] buf1;

delete[] buf2;

cout << "Thread T" << (Tid + 1) << " has begun to calculate the function." << endl;

Matrix temp1 = multiplyMatrixesH(multiplyMatrixesH(MX, MC, 0, H),

ME, 0, H);

Matrix temp2 = multiplyMatrixesH(MM, MD, 0, H);

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

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, temp1.get(i, j) - temp2.get(i, j));

}

}

cout << "Thread T" << (Tid + 1) << " has ended to calculate the function." << endl;

buf1Size = MA.getColCount() \* H;

buf1 = new long[buf1Size];

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, 3, Tid, MPI\_COMM\_WORLD, &status);

ptr = buf1;

for (int i = 3 \* H; i < (4 \* H); i++)

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, \*ptr);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

delete[] buf1;

buf1Size = MA.getColCount() \* 2 \* H;

buf1 = new long[buf1Size];

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, 1, Tid, MPI\_COMM\_WORLD, &status);

ptr = buf1;

for (int i = H; i < (3 \* H); i++)

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, \*ptr);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

delete[] buf1;

int Tsend = P / 2;

int prevTsend = P;

while (Tsend >= 4)

{

buf1Size = MA.getColCount() \* H \* (prevTsend - Tsend);

buf1 = new long[buf1Size];

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, Tsend, Tid, MPI\_COMM\_WORLD, &status);

ptr = buf1;

for (int i = Tsend \* H; i < (prevTsend \* H); i++)

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, \*ptr);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

delete[] buf1;

prevTsend = Tsend;

Tsend /= 2;

}

MA.writeToFile("MA.txt");

}

/////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////

void T0i(int Tid)

{

MPI\_Status status;

int Tprev = Tid;

int divider = 2;

int temp = P / divider;

while (temp != Tid)

{

temp += (P / divider);

if (temp >= P)

{

divider \*= 2;

temp = P / divider;

}

}

int prevDivider = divider / 2;

temp = 0;

while (temp < Tid)

{

temp += (P / prevDivider);

}

temp -= (P / prevDivider);

Tprev = temp;

Matrix MA((P / divider) \* H, N);

Matrix MX((P / divider) \* H, N);

Matrix MC(N);

Matrix ME(N);

Matrix MM((P / divider) \* H, N);

Matrix MD(N);

long\* ptr;

int buf1Size = MC.getSize() + ME.getSize() + MD.getSize();

long\* buf1 = new long[buf1Size];

int buf2Size = MX.getSize() + MM.getSize();

long\* buf2 = new long[buf2Size];

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, Tprev, Tid, MPI\_COMM\_WORLD, &status);

MPI\_Recv((void\*) buf2, buf2Size, MPI\_LONG, Tprev, Tid, MPI\_COMM\_WORLD, &status);

MC.readFromBufer(buf1);

ptr = (long\*) ((size\_t) buf1 + MC.getSize() \* sizeof(long));

ME.readFromBufer(ptr);

ptr = (long\*) ((size\_t) ptr + ME.getSize() \* sizeof(long));

MD.readFromBufer(ptr);

MX.readFromBufer(buf2);

ptr = (long\*) ((size\_t) buf2 + MX.getSize() \* sizeof(long));

MM.readFromBufer(ptr);

delete[] buf2;

int nextDivider = divider \* 2;

int Trec = Tid + (P / nextDivider);

while ((P / nextDivider) >= 4)

{

buf2Size = (MX.getColCount() + MM.getColCount()) \* H \* (P / nextDivider);

buf2 = new long[buf2Size];

ptr = buf2;

for (int i = H \* (Trec - Tid); i < ((Trec - Tid + P / nextDivider) \* H); i++)

{

for (int j = 0; j < MX.getColCount(); j++)

{

\*ptr = MX.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

for (int i = H \* (Trec - Tid); i < ((Trec - Tid + P / nextDivider) \* H); i++)

{

for (int j = 0; j < MM.getColCount(); j++)

{

\*ptr = MM.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, Trec, Trec, MPI\_COMM\_WORLD);

MPI\_Send((void\*) buf2, buf2Size, MPI\_LONG, Trec, Trec, MPI\_COMM\_WORLD);

delete[] buf2;

nextDivider \*= 2;

Trec = Tid + (P / nextDivider);

}

buf2Size = (MX.getColCount() + MM.getColCount()) \* H \* 2;

buf2 = new long[buf2Size];

ptr = buf2;

for (int i = H; i < (3 \* H); i++)

{

for (int j = 0; j < MX.getColCount(); j++)

{

\*ptr = MX.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

for (int i = H; i < (3 \* H); i++)

{

for (int j = 0; j < MM.getColCount(); j++)

{

\*ptr = MM.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, (Tid + 1), (Tid + 1), MPI\_COMM\_WORLD);

MPI\_Send((void\*) buf2, buf2Size, MPI\_LONG, (Tid + 1), (Tid + 1), MPI\_COMM\_WORLD);

delete[] buf2;

buf2Size = (MX.getColCount() + MM.getColCount()) \* H;

buf2 = new long[buf2Size];

ptr = buf2;

for (int i = 3 \* H; i < (4 \* H); i++)

{

for (int j = 0; j < MX.getColCount(); j++)

{

\*ptr = MX.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

for (int i = 3 \* H; i < (4 \* H); i++)

{

for (int j = 0; j < MM.getColCount(); j++)

{

\*ptr = MM.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, (Tid + 3), (Tid + 3), MPI\_COMM\_WORLD);

MPI\_Send((void\*) buf2, buf2Size, MPI\_LONG, (Tid + 3), (Tid + 3), MPI\_COMM\_WORLD);

delete[] buf1;

delete[] buf2;

cout << "Thread T" << (Tid + 1) << " has begun to calculate the function." << endl;

Matrix temp1 = multiplyMatrixesH(multiplyMatrixesH(MX, MC, 0, H),

ME, 0, H);

Matrix temp2 = multiplyMatrixesH(MM, MD, 0, H);

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

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, temp1.get(i, j) - temp2.get(i, j));

}

}

cout << "Thread T" << (Tid + 1) << " has ended to calculate the function." << endl;

buf1Size = MA.getColCount() \* H;

buf1 = new long[buf1Size];

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, (Tid + 3), Tid, MPI\_COMM\_WORLD, &status);

ptr = buf1;

for (int i = 3 \* H; i < (4 \* H); i++)

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, \*ptr);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

delete[] buf1;

buf1Size = MA.getColCount() \* 2 \* H;

buf1 = new long[buf1Size];

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, (Tid + 1), Tid, MPI\_COMM\_WORLD, &status);

ptr = buf1;

for (int i = H; i < (3 \* H); i++)

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, \*ptr);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

delete[] buf1;

nextDivider = divider \* 2;

int Tsend = Tid + (P / nextDivider);

while ((P / nextDivider) >= 4)

{

buf1Size = MA.getColCount() \* H \* (P / nextDivider);

buf1 = new long[buf1Size];

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, Tsend, Tid, MPI\_COMM\_WORLD, &status);

ptr = buf1;

for (int i = H \* (Tsend - Tid); i < ((Tsend - Tid + P / nextDivider) \* H); i++)

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, \*ptr);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

delete[] buf1;

nextDivider \*= 2;

Tsend = Tid + (P / nextDivider);

}

buf1Size = MA.getSize();

buf1 = new long[buf1Size];

MA.writeToBufer(buf1);

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, Tprev, Tprev, MPI\_COMM\_WORLD);

}

/////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////

void T2i3i(int Tid)

{

MPI\_Status status;

Matrix MA(H, N);

Matrix MX(H, N);

Matrix MC(N);

Matrix ME(N);

Matrix MM(H, N);

Matrix MD(N);

long\* ptr;

int buf1Size = MC.getSize() + ME.getSize() + MD.getSize();

long\* buf1 = new long[buf1Size];

int buf2Size = MX.getSize() + MM.getSize();

long\* buf2 = new long[buf2Size];

int Tsend;

if ((Tid % 4) == 2)

{

Tsend = Tid - 1;

}

else

{

Tsend = Tid - 3;

}

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, Tsend, Tid, MPI\_COMM\_WORLD, &status);

MPI\_Recv((void\*) buf2, buf2Size, MPI\_LONG, Tsend, Tid, MPI\_COMM\_WORLD, &status);

MC.readFromBufer(buf1);

ptr = (long\*) ((size\_t) buf1 + MC.getSize() \* sizeof(long));

ME.readFromBufer(ptr);

ptr = (long\*) ((size\_t) ptr + ME.getSize() \* sizeof(long));

MD.readFromBufer(ptr);

delete[] buf1;

MX.readFromBufer(buf2);

ptr = (long\*) ((size\_t) buf2 + MX.getSize() \* sizeof(long));

MM.readFromBufer(ptr);

delete[] buf2;

cout << "Thread T" << (Tid + 1) << " has begun to calculate the function." << endl;

Matrix temp1 = multiplyMatrixesH(multiplyMatrixesH(MX, MC, 0, H),

ME, 0, H);

Matrix temp2 = multiplyMatrixesH(MM, MD, 0, H);

for (int i = 0; i < MA.getRowCount(); i++)

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, temp1.get(i, j) - temp2.get(i, j));

}

}

cout << "Thread T" << (Tid + 1) << " has ended to calculate the function." << endl;

buf1Size = MA.getSize();

buf1 = new long[buf1Size];

MA.writeToBufer(buf1);

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, Tsend, Tsend, MPI\_COMM\_WORLD);

}

/////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////

void T1i(int Tid)

{

MPI\_Status status;

Matrix MA(2 \* H, N);

Matrix MX(2 \* H, N);

Matrix MC(N);

Matrix ME(N);

Matrix MM(2 \* H, N);

Matrix MD(N);

long\* ptr;

int buf1Size = MC.getSize() + ME.getSize() + MD.getSize();

long\* buf1 = new long[buf1Size];

int buf2Size = MX.getSize() + MM.getSize();

long\* buf2 = new long[buf2Size];

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, (Tid - 1), Tid, MPI\_COMM\_WORLD, &status);

MPI\_Recv((void\*) buf2, buf2Size, MPI\_LONG, (Tid - 1), Tid, MPI\_COMM\_WORLD, &status);

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, (Tid + 1), (Tid + 1), MPI\_COMM\_WORLD);

MC.readFromBufer(buf1);

ptr = (long\*) ((size\_t) buf1 + MC.getSize() \* sizeof(long));

ME.readFromBufer(ptr);

ptr = (long\*) ((size\_t) ptr + ME.getSize() \* sizeof(long));

MD.readFromBufer(ptr);

delete[] buf1;

MX.readFromBufer(buf2);

ptr = (long\*) ((size\_t) buf2 + MX.getSize() \* sizeof(long));

MM.readFromBufer(ptr);

delete[] buf2;

buf2Size = H \* (MX.getColCount() + MM.getColCount());

buf2 = new long[buf2Size];

ptr = buf2;

for (int i = H; i < MX.getRowCount(); i++)

{

for (int j = 0; j < MX.getColCount(); j++)

{

\*ptr = MX.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

for (int i = H; i < MM.getRowCount(); i++)

{

for (int j = 0; j < MM.getColCount(); j++)

{

\*ptr = MM.get(i, j);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

MPI\_Send((void\*) buf2, buf2Size, MPI\_LONG, (Tid + 1), (Tid + 1), MPI\_COMM\_WORLD);

delete[] buf2;

cout << "Thread T" << (Tid + 1) << " has begun to calculate the function." << endl;

Matrix temp1 = multiplyMatrixesH(multiplyMatrixesH(MX, MC, 0, H),

ME, 0, H);

Matrix temp2 = multiplyMatrixesH(MM, MD, 0, H);

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

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, temp1.get(i, j) - temp2.get(i, j));

}

}

cout << "Thread T" << (Tid + 1) << " has ended to calculate the function." << endl;

buf1Size = H \* MA.getColCount();

buf1 = new long[buf1Size];

MPI\_Recv((void\*) buf1, buf1Size, MPI\_LONG, (Tid + 1), Tid, MPI\_COMM\_WORLD, &status);

ptr = buf1;

for (int i = H; i < (MA.getRowCount()); i++)

{

for (int j = 0; j < MA.getColCount(); j++)

{

MA.set(i, j, \*ptr);

ptr = (long\*) ((size\_t) ptr + sizeof(long));

}

}

delete[] buf1;

buf1Size = MA.getSize();

buf1 = new long[buf1Size];

MA.writeToBufer(buf1);

MPI\_Send((void\*) buf1, buf1Size, MPI\_LONG, (Tid - 1), (Tid - 1), MPI\_COMM\_WORLD);

}

/////////////////////////////////////////////////////////////////////////

int main(int argc, char \*argv[])

{

int Tid;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &Tid);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &P);

H = N / P;

cout << "Thread T" << (Tid + 1) << " has started." << endl;

if (Tid == 0)

{

T0(Tid);

}

else

{

if ((Tid % 4) == 0)

{

T0i(Tid);

}

else

{

if (((Tid % 4) == 2) || ((Tid % 4) == 3))

{

T2i3i(Tid);

}

else

{

T1i(Tid);

}

}

}

cout << "Thread T" << (Tid + 1) << " has finished." << endl;

return 0;

}

#pragma once

#include "Vector.h"

class Matrix

{

public:

Matrix(int N);

Matrix(int H, int N);

~Matrix(void);

int getRowCount();

int getColCount();

int getSize();

long get(int i, int j);

void set(int i, int j, long e);

void fillWithOnes();

void writeToBufer(long\* buf);

void readFromBufer(long\* buf);

void writeToFile(char fileName[]);

private:

vector<Vector> m;

};

#include "StdAfx.h"

#include "Matrix.h"

#include <fstream>

#include <new>

Matrix::Matrix(int N)

{

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

{

m.push\_back(Vector(N));

}

}

Matrix::Matrix(int H, int N)

{

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

{

m.push\_back(Vector(N));

}

}

Matrix::~Matrix(void)

{

m.~vector();

}

int Matrix::getRowCount()

{

return m.size();

}

int Matrix::getColCount()

{

return m[0].getSize();

}

int Matrix::getSize()

{

return (getRowCount() \* getColCount());

}

long Matrix::get(int i, int j)

{

return m[i].get(j);

}

void Matrix::set(int i, int j, long e)

{

m[i].set(j, e);

}

void Matrix::fillWithOnes()

{

for (size\_t i = 0; i < m.size(); i++)

{

m[i].fillWithOnes();

}

}

void Matrix::writeToBufer(long\* buf)

{

int k = 0;

for (int i = 0; i < this->getRowCount(); i++)

{

for (int j = 0; j < this->getColCount(); j++)

{

buf[k] = this->get(i, j);

k++;

}

}

}

void Matrix::readFromBufer(long\* buf)

{

int k = 0;

for (int i = 0; i < this->getRowCount(); i++)

{

for (int j = 0; j < this->getColCount(); j++)

{

this->set(i, j, buf[k]);

k++;

}

}

}

void Matrix::writeToFile(char fileName[])

{

ofstream outFile(fileName);

for (size\_t i = 0; i < m.size(); i++)

{

for (size\_t j = 0; j < m[i].getSize(); j++)

{

outFile << m[i].get(j) << " ";

}

outFile << endl;

}

outFile.close();

}

#pragma once

#include <vector>

using namespace std;

class Vector

{

public:

Vector(int N);

~Vector(void);

size\_t getSize();

long get(int i);

void set(int i, long e);

void fillWithOnes();

private:

vector<long> v;

};

#include "stdafx.h"

#include "Vector.h"

#include <fstream>

#include <new>

Vector::Vector(int N)

{

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

{

v.push\_back(0);

}

}

Vector::~Vector(void)

{

v.~vector();

}

size\_t Vector::getSize()

{

return v.size();

}

long Vector::get(int i)

{

return v[i];

}

void Vector::set(int i, long e)

{

v[i] = e;

}

void Vector::fillWithOnes()

{

for (size\_t i = 0; i < v.size(); i++)

{

v[i] = 1;

}

}