НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №4**

з дисципліни **«**Паралельні та розподілені обчислення**»**

Виконав:

студент 3 курсу гр. ІО-42

Кочетов Данило

№ ЗК 4213

Перевірив:

Долголенко О. М.

Київ 2016 р.

***Завдання:***

1.13; 2.13; 3.13

F1: C = A\*(MA\*ME) + B + D

F2: ML = MIN(MF)\*MG + MAX(MH) \* (MK\*MF)

F3: T = (MO\*MP)\*S + MR\*SORT(S)

***Лістинг програми:***

***// Lab4.cpp***

#include "F1.h"

#include "F2.h"

#include "F3.h"

const int N = 500;

int main() {

cout << "Lab 4 start" << endl << endl;

DWORD tid[3];

HANDLE threads[3];

F1\* f1 = new F1(N);

F2\* f2 = new F2(N);

F3\* f3 = new F3(N);

threads[0] = CreateThread(NULL, 0, F1::startThread, f1, 0, &tid[0]);

threads[1] = CreateThread(NULL, 0, F2::startThread, f2, 0, &tid[1]);

threads[2] = CreateThread(NULL, 0, F3::startThread, f3, 0, &tid[1]);

SetThreadPriority(threads[0], THREAD\_PRIORITY\_LOWEST);

SetThreadPriority(threads[1], THREAD\_PRIORITY\_NORMAL);

SetThreadPriority(threads[2], THREAD\_PRIORITY\_HIGHEST);

WaitForMultipleObjects(3, threads, true, INFINITE);

cout << endl << "Lab 4 end" << endl << endl << "Press Enter...";

string t;

getline(cin, t);

delete f1;

delete f2;

delete f3;

}

***// F1.h***

#pragma once

#include <iostream>

#include <Windows.h>

#include "Matrix.h"

class F1 {

private:

Vector\* result;

int N;

public:

F1(int N);

Vector\* getResult();

static DWORD WINAPI startThread(void\* param);

DWORD run();

};

***// F1.cpp***

#include "F1.h"

F1::F1(int N) {

this->N = N;

}

Vector\* F1::getResult() {

return result;

}

DWORD WINAPI F1::startThread(void\* param) {

F1\* This = (F1\*) param;

return This->run();

}

DWORD F1::run() {

cout << "Task 1 start\n";

Vector \*A = new Vector(N), \*B = new Vector(N), \*D = new Vector(N);

Matrix \*MA = new Matrix(N), \*ME = new Matrix(N);

result = MA->multiply(ME)->multiply(A)->sum(B)->sum(D);

cout << "Task 1 end\n";

delete A;

delete B;

delete D;

delete MA;

delete ME;

return 0;

}

***// F2.h***

#pragma once

#include <iostream>

#include <Windows.h>

#include "Matrix.h"

class F2 {

private:

Matrix\* result;

int N;

public:

F2(int N);

Matrix\* getResult();

static DWORD WINAPI startThread(void\* param);

DWORD run();

};

***// F2.cpp***

#include "F2.h"

F2::F2(int N) {

this->N = N;

}

Matrix\* F2::getResult() {

return result;

}

DWORD WINAPI F2::startThread(void\* param) {

F2\* This = (F2\*) param;

return This->run();

}

DWORD F2::run() {

cout << "Task 2 start\n";

Matrix \*MF = new Matrix(N), \*MG = new Matrix(N), \*MH = new Matrix(N), \*MK = new Matrix(N);

result = MG->multiply(MF->get\_min())->sum(MK->multiply(MF)->multiply(MH->get\_max()));

cout << "Task 2 end\n";

delete MF;

delete MG;

delete MH;

delete MK;

return 0;

}

***// F3.h***

#pragma once

#include <iostream>

#include <Windows.h>

#include "Matrix.h"

class F3 {

private:

Vector\* result;

int N;

public:

F3(int N);

Vector\* getResult();

static DWORD WINAPI startThread(void\* param);

DWORD run();

};

***// F3.cpp***

#include "F3.h"

F3::F3(int N) {

this->N = N;

}

Vector\* F3::getResult() {

return result;

}

DWORD WINAPI F3::startThread(void\* param) {

F3\* This = (F3\*)param;

return This->run();

}

DWORD F3::run() {

cout << "Task 3 start\n";

Vector\* S = new Vector(N);

Matrix \*MO = new Matrix(N), \*MP = new Matrix(N), \*MR = new Matrix(N);

result = MO->multiply(MP)->multiply(S)->sum(MR->multiply(S->sort()));

cout << "Task 3 end\n";

delete S;

delete MO;

delete MP;

delete MR;

return 0;

}

***// Vector.h***

#pragma once

#include <random>

#include <ctime>

#include <string>

using namespace std;

class Vector {

private:

long\* grid;

int N;

public:

Vector();

Vector(int N);

Vector(long\* grid, int N);

~Vector();

int getSize();

long get(int i);

Vector\* sum(Vector\* v);

Vector\* sort();

string toString();

};

***// Vector.cpp***

#include "Vector.h"

Vector::Vector() {}

Vector::Vector(int N) {

this->N = N;

srand(time(NULL));

grid = new long[N];

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

grid[i] = rand() % 20;

}

Vector::Vector(long\* grid, int N) {

this->N = N;

this->grid = new long[N];

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

this->grid[i] = grid[i];

}

Vector::~Vector() {

delete[] grid;

}

int Vector::getSize() {

return N;

}

long Vector::get(int i) {

return grid[i];

}

Vector\* Vector::sum(Vector\* v) {

int N = getSize();

long\* newGrid = new long[N];

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

newGrid[i] = grid[i] + v->get(i);

Vector\* newVector = new Vector(newGrid, N);

delete[] newGrid;

return newVector;

}

Vector\* Vector::sort() {

int N = getSize();

long\* newGrid = new long[N];

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

newGrid[i] = grid[i];

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

for (int k = 0; k < N - i - 1; ++k) {

if (newGrid[k] > newGrid[k + 1]) {

long t = newGrid[k];

newGrid[k] = newGrid[k + 1];

newGrid[k + 1] = t;

}

}

}

Vector\* newVector = new Vector(newGrid, N);

delete[] newGrid;

return newVector;

}

string Vector::toString() {

string res = "";

int N = getSize();

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

res += grid[i] + " ";

return res;

}

***// Matrix.h***

#pragma once

#include <random>

#include <ctime>

#include "Vector.h"

class Matrix {

private:

long\*\* grid;

int N;

public:

Matrix(int N);

Matrix(long\*\* grid, int N);

~Matrix();

long get(int i, int k);

int getSize();

Matrix\* multiply(Matrix\* m);

Vector\* multiply(Vector\* v);

Matrix\* multiply(long a);

Matrix\* sum(Matrix\* m);

long get\_min();

long get\_max();

string toString();

};

***// Matrix.cpp***

#include "Matrix.h"

#include <iostream>

Matrix::Matrix(int N) {

this->N = N;

srand(time(NULL));

grid = new long\*[N];

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

grid[i] = new long[N];

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

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

grid[i][k] = rand() % 20;

}

Matrix::Matrix(long\*\* grid, int N) {

this->N = N;

this->grid = new long\*[N];

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

this->grid[i] = new long[N];

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

this->grid[i][k] = grid[i][k];

}

}

Matrix::~Matrix() {

int N = getSize();

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

delete[] grid[i];

delete[] grid;

}

long Matrix::get(int i, int k) {

return grid[i][k];

}

int Matrix::getSize() {

return N;

}

Matrix\* Matrix::multiply(Matrix\* m) {

int N = getSize();

long\*\* newGrid = new long\*[N];

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

newGrid[i] = new long[N];

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

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

newGrid[i][k] = 0;

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

newGrid[i][k] += grid[i][j] \* m->get(j, k);

}

}

}

Matrix\* newMatrix = new Matrix(newGrid, N);

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

delete[] newGrid[i];

delete[] newGrid;

return newMatrix;

}

Vector\* Matrix::multiply(Vector\* v) {

int N = getSize();

long\* newGrid = new long[N];

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

newGrid[i] = 0;

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

newGrid[i] += v->get(k) \* grid[i][k];

}

}

Vector\* newVector = new Vector(newGrid, N);

delete[] newGrid;

return newVector;

}

Matrix\* Matrix::multiply(long a) {

int N = getSize();

long\*\* newGrid = new long\*[N];

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

newGrid[i] = new long[N];

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

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

newGrid[i][k] = grid[i][k] \* a;

}

}

Matrix\* newMatrix = new Matrix(newGrid, N);

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

delete[] newGrid[i];

delete[] newGrid;

return newMatrix;

}

Matrix\* Matrix::sum(Matrix\* m) {

int N = getSize();

long\*\* newGrid = new long\*[N];

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

newGrid[i] = new long[N];

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

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

newGrid[i][k] = grid[i][k] + m->get(i, k);

}

}

Matrix\* newMatrix = new Matrix(newGrid, N);

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

delete[] newGrid[i];

delete[] newGrid;

return newMatrix;

}

long Matrix::get\_min() {

long res = grid[0][0];

int N = getSize();

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

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

if (res < grid[i][k])

res = grid[i][k];

}

}

return res;

}

long Matrix::get\_max() {

long res = grid[0][0];

int N = getSize();

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

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

if (res > grid[i][k])

res = grid[i][k];

}

}

return res;

}

string Matrix::toString() {

string res = "";

int N = getSize();

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

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

res += grid[i][k] + "\t";

}

res += "\n";

}

return res;

}