КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Кафедра інтелектуальних та інформаційних систем

Лабораторна робота № 5

з дисципліни

“Методи та системи паралельного програмування”

Виконав студент

групи КН-31

Пашковський Павло Володимирович

Київ-2020

#include <iostream>

#include <cmath>

#include <vector>

#include <algorithm>

#include <stdlib.h>

#include <cmath>

#include <set>

#include <mpi.h>

using namespace std;

int tempSize = 21;

int cnt = 0;

int cntAll = 0;

int SIZE = 7;

int m = SIZE - 1;

double\*\* matrix;

vector<vector<int> > a(2, vector<int>(0));

int\*\* list;

int\* idx;

#define MASTER 0

#define FROM\_MASTER 1

#define FROM\_WORKER 2

MPI\_Status status;

MPI\_Request request;

int numrank;

int numsize;

void initMatrix(int size) {

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

{

for (int j = i; j < size; j++)

{

if (i == j)

{

matrix[i][j] = 1;

}

else

{

int temp = rand() % 3 + 1;

if (temp==3) {

matrix[i][j] = 1 / (double(rand() % 10 + 1));

}

else

if (temp == 2)

{

matrix[i][j] = rand() % 10 + 1;

}

else matrix[i][j] = 0;

if (matrix[i][j] != 0)

matrix[j][i] = 1 / matrix[i][j];

else matrix[j][i] = 0;

a[0].push\_back(i);

a[1].push\_back(j);

}

}

}

}

void showMatrix(double\*\* matrix, int size)

{

cout << "MATRIX" << "\n";

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

{

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

{

cout << matrix[i][j] << " ";

}

cout << "\n";

}

}

void showVector(double\* vectorPriority, int SIZE) {

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

{

cout << vectorPriority[i] << " ";

}

}

void getVectorPriority(double\*\* matrix) {

double\* vectorPriorityArif = new double[SIZE];

double\* vectorPriorityGeom = new double[SIZE];

double\* geom = new double[SIZE];

double\* arif = new double[SIZE];

double sumVectorGeom = 0;

double sumVectorArif = 0;

cout << endl;

showMatrix(matrix, SIZE);

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

{

geom[i] = 1;

arif[i] = 0;

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

{

//logarifm

geom[i] \*= log(pow(5, matrix[i][j])) / log(5);

arif[i] += matrix[i][j];

}

geom[i] = pow(geom[i], 1 / SIZE);

arif[i] = arif[i] / SIZE;

sumVectorGeom += geom[i];

sumVectorArif += arif[i];

}

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

{

vectorPriorityGeom[i] = geom[i] / sumVectorGeom;

vectorPriorityArif[i] = arif[i] / sumVectorArif;

}

showVector(vectorPriorityArif, SIZE);

cout << endl;

showVector(vectorPriorityGeom, SIZE);

cout << endl;

delete[] vectorPriorityArif;

delete[] vectorPriorityGeom;

delete[] geom;

delete[] arif;

}

double sumVector(double\* vectorPriority)

{

double sum = 0;

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

{

sum += vectorPriority[i];

}

return sum;

}

bool NextSet(int n, int m)

{

int k = m;

for (int i = k - 1; i >= 0; --i)

if (idx[i] < n - k + i + 1)

{

++idx[i];

for (int j = i + 1; j < k; ++j)

idx[j] = idx[j - 1] + 1;

return true;

}

return false;

}

bool checkDerevo(int\*\* list) {

set <int, greater <int> > check;

check.insert(list[0][0]);

check.insert(list[1][0]);

vector<int> checkBool(m);

vector<int> temp(2);

checkBool[0] = 1;

for (int i = 1; i < m; i++)

{

temp[0] = list[0][i];

temp[1] = list[1][i];

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

{

if (i != j)

{

if (find(check.begin(), check.end(), temp[0]) != check.end()) {

check.insert(temp[1]);

checkBool[i] = 1;

}

else

{

if (find(check.begin(), check.end(), temp[1]) != check.end()) {

check.insert(temp[0]);

checkBool[i] = 1;

}

}

}

}

}

for (int i = 0; i < checkBool.size(); i++)

{

if (checkBool[i] == 0)

return false;

}

return true;

}

void resetAllVector() {

for (int i = 0; i < 2; i++)

{

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

{

list[i][j] = 0;

}

}

}

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

{

MPI\_Init(&argc, &argv);

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

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

idx = new int[tempSize];

list = new int\* [2];

for (int i = 0; i < 2; i++)

{

list[i] = new int[tempSize];

}

matrix = new double\* [SIZE];

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

{

matrix[i] = new double[SIZE];

}

initMatrix(SIZE);

int n = a[0].size();

if (numrank == 0) {

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

idx[i] = i + 1;

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

{

list[0][i] = a[0][idx[i] - 1];

list[1][i] = a[1][idx[i] - 1];

cout << list[0][i] << list[1][i] << " ";

}

cout << endl;

double t1, t2;

t1 = MPI\_Wtime();

double\*\* currentMatrix = new double\* [SIZE];

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

{

currentMatrix[i] = new double[SIZE];

}

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

{

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

{

currentMatrix[i][j] = 0;

}

}

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

{

for (int j = i; j < SIZE; j++)

{

if (i == j) {

currentMatrix[i][j] = 1;

}

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

{

if (i == list[0][k] && j == list[1][k])

{

currentMatrix[i][j] = matrix[i][j];

}

}

}

}

if (checkDerevo(list))

{

cnt++;

for (int i = 0; i < SIZE - 1; i++)

{

for (int j = i + 1; j < SIZE; j++)

{

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

{

if (currentMatrix[i][j] == 0 && matrix[i][k] != 0 && matrix[j][k] != 0)

{

currentMatrix[i][j] = matrix[i][k] \* matrix[j][k];

}

}

}

}

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

{

for (int j = i; j < SIZE; j++)

{

currentMatrix[j][i] = 1 / currentMatrix[i][j];

}

}

}

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

{

delete[] currentMatrix[i];

}

delete[] currentMatrix;

resetAllVector();

showMatrix(matrix, SIZE);

if (n >= m)

{

int step = (int)pow(SIZE, SIZE - 2) / (numsize-1);

for (int i = 1; i < numsize; i++)

{

MPI\_Send(&list[0][0], m, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

MPI\_Send(&list[1][0], m, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

MPI\_Send(&idx[0], tempSize, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

MPI\_Send(&cntAll, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

//MPI\_Barrier(MPI\_COMM\_WORLD);

}

t2 = MPI\_Wtime();

printf("All time is %f\n", t2 - t1);

cout << "Finish" << endl;

cout << cntAll << endl;

}

}

if(numrank>0){

MPI\_Recv(&list[0][0], m, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&list[1][0], m, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&idx[0], tempSize, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&cntAll, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

double t1, t2;

t1 = MPI\_Wtime();

for (int j = 0; j < pow(SIZE, SIZE - 2) / (numsize - 1); j++)

{

double\*\* currentMatrix = new double\* [SIZE];

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

{

currentMatrix[i] = new double[SIZE];

}

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

{

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

{

currentMatrix[i][j] = 0;

}

}

NextSet(n, m);

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

{

list[0][k] = a[0][idx[k] - 1];

list[1][k] = a[1][idx[k] - 1];

}

if (checkDerevo(list))

{

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

{

for (int j = i; j < SIZE; j++)

{

if (i == j) {

currentMatrix[i][j] = 1;

}

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

{

if (i == list[0][k] && j == list[1][k])

{

currentMatrix[i][j] = matrix[i][j];

}

}

}

}

for (int i = 0; i < SIZE - 1; i++)

{

for (int j = i + 1; j < SIZE; j++)

{

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

{

if (currentMatrix[i][j] == 0 && matrix[i][k] != 0 && matrix[j][k] != 0)

{

currentMatrix[i][j] = matrix[i][k] \* matrix[j][k];

}

}

}

}

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

{

for (int j = i+1; j < SIZE; j++)

{

currentMatrix[j][i] = 1 / currentMatrix[i][j];

}

}

cntAll++;

cout << "rank: " << numrank << endl;

getVectorPriority(currentMatrix);

//MPI\_Barrier(MPI\_COMM\_WORLD);

}

else {

cout << "It`s not a tree"<<endl;

}

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

{

delete[] currentMatrix[i];

}

delete[] currentMatrix;

t2 = MPI\_Wtime();

printf("Time is %f\n", t2 - t1);

resetAllVector();

//cout << cntAll << endl;

}

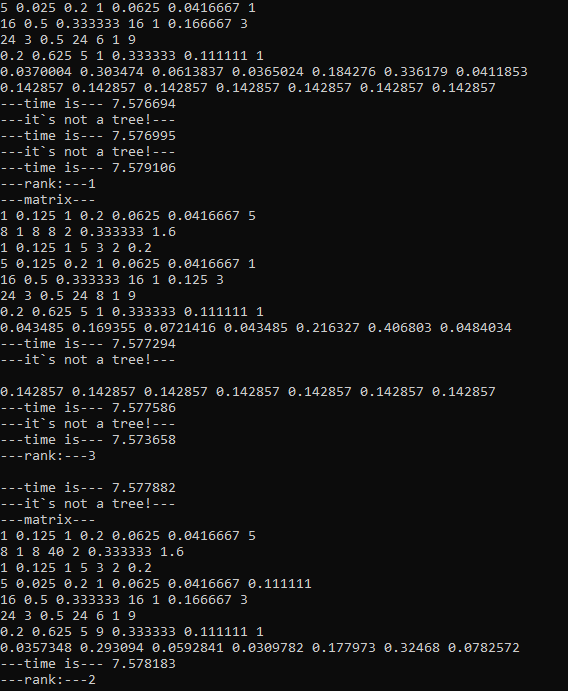
}

MPI\_Finalize();

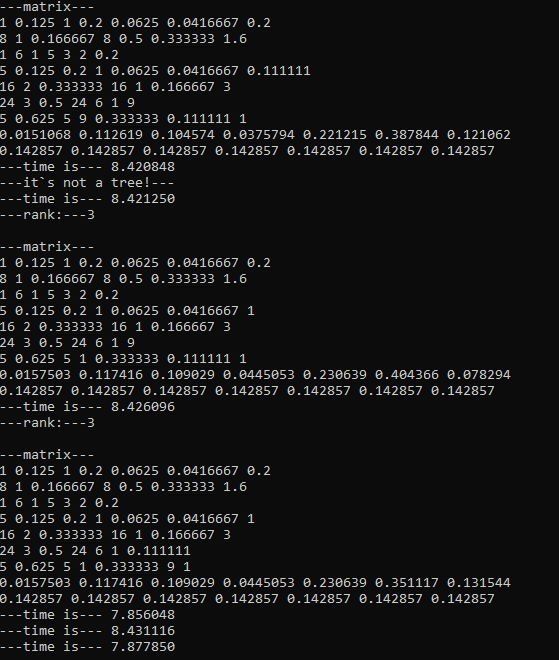
return 0;

}

**Результат виконання роботи:**



…



**Висновок**

В даній лабораторній роботі я освоїв алгоритмізацію паралельних обчислень та їх реалізацію за допомогою засобів технології MPI на практичному прикладі комбінаторного методу визначення вектора пріоритетів на основі неповної мультиплікативної матриці експертних парних порівнянь; набула практичних навичок створення ефективних паралельних програм з використанням технології MPI; навчилася налаштовувати обчислювальний кластер на базі однорідної комп’ютерної мережі.