Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра ЕОМ



**Звіт**

з лабораторної роботи №1

з дисципліни: “ Паралельні та розподілені обчислення”

на тему:

“ **ВИКОРИСТАННЯ ФУНКЦІОНАЛЬНОЇ ДЕКОМПОЗИЦІЇ ДЛЯ РОЗВ’ЯЗКУ ОБЧИСЛЮВАЛЬНИХ ЗАДАЧ.**”

Виконав: ст. гр. КІ-34

Гриценко В.В.

Прийняв: викл.

Козак Н.Б.

Львів

2020

**Тема:** Використання функціональної декомпозиції для розв’язку обчислювальних задач.

**Мета роботи:** Вивчити методи декомпозицій задач. Набути навиків розв’язування задач з використанням функціональної декомпозиції.

**Завдання:**

Вираз, який слід обрахувати, заданий наступним чином:



При чому елементи визначаються згідно правил:



, де bi=*i*2/12 для парних і bi=*i* для непарних *і*



*y2 =* A1(12b1-c1)

Y3 = A2(B2-C2) , де Cij=1/(i+j2)

**Хід роботи:**

**1.  Аналіз завдання.**

Для заданого виразу вхідними даними є:

розмірність матриць – n;

матриці ;



вектори-стовпці .



Ці параметри повинні вводитися з клавіатури, або генеруватися випадковим чином (крім розмірності). При чому, елементи всіх матриць та векторів є цілими додатними числами, більшими за нуль.

Вектор-стовпець та матриця обраховуються, виходячи з уведеної розмірності, зауважимо, що значення їх елементів завжди менші одиниці і різко спадають зі збільшенням розмірності.



При утворенні враховуємо, що результатом множення матриці А на вектор-стовпець b є вектор-стовпець, елементи якого будуть раціональними числами(тобто матимуть значущу дробову частину).



При утворенні враховуємо, що результатом віднімання двох векторів-стовпців є вектор-стовпець, елементи якого можуть бути меншими за нуль цілими числами. Далі, при множенні цілочисельної додатної матриці А1 на результат віднімання, отримаємо вектор-стовпець з цілочисельними елементами довільного знаку.



При утворенні враховуємо, що присутні лише операції додавання та множення, а тому вихідний результат завжди буде додатнім і завжди матиме значущу дробову частину.



Таким чином, згідно поставленої задачі, в обчисленні загального виразу приймають участь три різні елементи – два вектори стовпці та матриця .



Оскільки, згідно правил матричних обчислень, добуток не є комутативною операцією, всі множення слід виконувати в тій послідовності, яка задана. Результатом множення рядка на стовпець є число, а матриці на матрицю - матриця. Тому, в загальному, перший доданок буде рядком. Аналогічний аналіз можна застосувати і до другого доданку виразу.

Таким чином, з попереднього випливає, що остаточний результат є рядком, елементи якої можуть бути як додатними так і від’ємними і завжди мають дробову частину.

**2.  Декомпозиція задачі.**

Однозначно, всі обчислення безпосередньо залежать від розмірності даних, тому найперше, слід забезпечити ввід змінної n, що визначає цю розмірність. Далі, можна паралельно виконувати обчислення значень вектора b та матриці С2, оскільки вони незалежні від інших параметрів. Крім того, на тому ж рівні декомпозиції слід визначати вхідні дані, тобто вводити з клавіатури, або генерувати випадковим чином матриці та вектори-стовпці . Наступний рівень декомпозиції – це знаходження елементів виразу. Значення залежить від введеної матриці А та обрахованого вектора b. Значення залежить від введеної А1 та різниці векторів b1 і c1, тому знайти його можна лише після обчислення (b1 - c1). Зауважимо, що множення на константу не є окремою операцією, як і транспонування векторів. Аналогічно, знаходимо . Подальша декомпозиція відбувається згідно заданої послідовності операцій та врахування залежностей отриманих на кожному рівні даних. Повна схема декомпозиції обчислення заданого виразу приведена нижче.



Ввід n

A

A1

A2

B2

b1

Cij = 1/(i+j2)

c1

bi=*i*2/12 or bi=*i*

(B2-C2)

12b1-c1

Y3 = A2(B2-C2)

*y2 =* A1(12b1-c1)

y1 = A \* b

Y32

y1’

y2’

Y32\*y1

y1\*y2’

Y32\*y1 + y2

y1\*(Y32\*y1 + y2)

y1\*(Y32\*y1 + y2)\* Y3

y1\*(Y32\*y1 + y2)\* Y3+ y1\*y2’

(y1\*(Y32\*y1 + y2)\* Y3+ y1\*y2’)\* y2

((y1\*(Y32\*y1 + y2)\* Y3+ y1\*y2’)\* y2)’

y2’+((y1\*(Y32\*y1 + y2)\* Y3+ y1\*y2’)\* y2)’

**Результат виконання:**

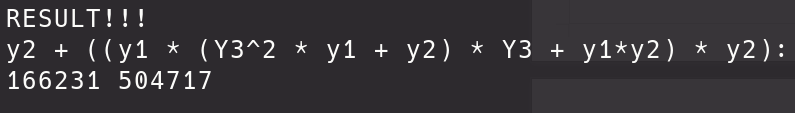


Рис.1 При n=2 і вручну заповнених матрицях, для векторів елементи = (1, 2), для матриць елементи = (1, 2, 3, 4)

**Вміст файлу log.txt з детальною інформаціею про виконання:**

> cat log.txt

Enter n:

myScanned value: 2

Generate data randomly? (1 - yeah(default), 2 - nope):

myScanned value: 2

c1:

1 2

A:

1 2

3 4

A1:

1 2

3 4

A2:

1 2

3 4

B2:

1 2

3 4

b1:

0 1

C2:

0 1

1 0.5

B2 - C2:

1 1

2 3.5

12b1 - c1:

-1 10

y1:

2 4

y2:

19 37

Y3:

5 11

8 17

Y3 ^ 2:

5 11

8 17

y1 \* y2:

38 148

Y3^2 \* y1:

38 148

Y3^2 \* y1 + y2:

57 185

y1 \* (Y3^2 \* y1 + y2):

114 740

y1 \* (Y3^2 \* y1 + y2) \* Y3:

8710 13492

y1 \* (Y3^2 \* y1 + y2) \* Y3 + y1\*y2:

8748 13640

(y1 \* (Y3^2 \* y1 + y2) \* Y3 + y1\*y2) \* y2:

166212 504680

RESULT!!!

y2 + ((y1 \* (Y3^2 \* y1 + y2) \* Y3 + y1\*y2) \* y2):

166231 504717

**Код програми:**

**main.cpp**

#include <iostream>

#include <fstream>

#include <sstream>

#include <ctime>

#include <thread>

#include <mutex>

#include "funcs.cpp"

static fstream file("log.txt",

ios::in | ios::out | ios::trunc);

static stringstream ss;

static thread th1, th2;

static mutex mtx;

void myPrint(string msg);

void myScan(double &value);

void fout\_matrix(vector<vector<double>>& matrix);

void fout\_vector(vector<double>& vector);

void doRandomly();

void doManually();

// ----- DATA -----

double n;

vector<vector<double>> A, A1, A2, B2, C2, Y3;

vector<double> b1, c1, y1, y2;

int main()

{

myPrint(string("Enter n: "));

myScan(n);

// ------------------------- 1 ----------------------------

double choice = 1;

myPrint(string("Generate data randomly? (1 - yeah(default), 2 - nope): "));

myScan(choice);

switch (int(choice)) {

case 2:

{

doManually();

} break;

default:

{

doRandomly();

} break;

}

file << "c1: " << endl;

fout\_vector(c1);

file << "A: " << endl;

fout\_matrix(A);

file << "A1: " << endl;

fout\_matrix(A1);

file << "A2: " << endl;

fout\_matrix(A2);

file << "B2: " << endl;

fout\_matrix(B2);

// b1

th1 = thread([]()

{

b1 = vector<double>(n, 0);

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

if(i % 2)

b1[i] = i;

else

b1[i] = i \* i / 12;

}

});

// C2

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

C2.push\_back(vector<double>(n, 0));

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

if(!(j || i))

{

C2[i][j] = 0;

continue;

}

C2[i][j] = 1.0 / double(i + j \* j);

}

}

th1.join();

file << "b1: " << endl;

fout\_vector(b1);

file << "C2: " << endl;

fout\_matrix(C2);

// ------------------------- 2 ----------------------------

vector<vector<double>> B2\_min\_C2(n, vector<double>(n));

vector<double> \_12b1\_min\_c1(n);

// B2 - C2

th1 = thread([&B2\_min\_C2]()

{

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

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

B2\_min\_C2[i][j] = B2[i][j] - C2[i][j];

}

}

});

// 12b1 - c1

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

{

\_12b1\_min\_c1[i] = 12 \* b1[i] - c1[i];

}

th1.join();

file << "B2 - C2:" << endl;

fout\_matrix(B2\_min\_C2);

file << "12b1 - c1: " << endl;

fout\_vector(\_12b1\_min\_c1);

// ------------------------- 3 ----------------------------

// Y3

th1 = thread([&B2\_min\_C2]()

{

Y3 = A\_mul\_B(&A2, &B2\_min\_C2);

});

// y2

th2 = thread([&\_12b1\_min\_c1]()

{

y2 = A\_mul\_b(&A1, &\_12b1\_min\_c1);

});

// y1

y1 = A\_mul\_b(&A, &b1);

th1.join();

th2.join();

file << "y1: " << endl;

fout\_vector(y1);

file << "y2: " << endl;

fout\_vector(y2);

file << "Y3: " << endl;

fout\_matrix(Y3);

// ------------------------- 4 ----------------------------

vector<vector<double>> Y3\_pow\_2(n, vector<double>(n));

// Y3^2

Y3\_pow\_2 = A\_mul\_B(&Y3, &Y3);

file << "Y3 ^ 2: " << endl;

fout\_matrix(Y3);

// ------------------------- 5 ----------------------------

vector<double> y1\_mul\_y2(n), Y3\_pow\_2\_mul\_y1(n);

// y1 \* y2

th1 = thread([&y1\_mul\_y2]()

{

y1\_mul\_y2 = a\_mul\_b(&y1, &y2);

});

// Y3^2 \* y1

Y3\_pow\_2\_mul\_y1 = a\_mul\_b(&y1, &y2);

th1.join();

file << "y1 \* y2: " << endl;

fout\_vector(y1\_mul\_y2);

file << "Y3^2 \* y1: " << endl;

fout\_vector(Y3\_pow\_2\_mul\_y1);

// ------------------------- 6 ----------------------------

vector<double> temp(n);

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

temp[i] = Y3\_pow\_2\_mul\_y1[i] + y2[i];

file << "Y3^2 \* y1 + y2: " << endl;

fout\_vector(temp);

// ------------------------- 7 ----------------------------

temp = a\_mul\_b(&y1, &temp);

file << "y1 \* (Y3^2 \* y1 + y2): " << endl;

fout\_vector(temp);

// ------------------------- 8 ----------------------------

temp = a\_mul\_B(&temp, &Y3);

file << "y1 \* (Y3^2 \* y1 + y2) \* Y3: " << endl;

fout\_vector(temp);

// ------------------------- 9 ----------------------------

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

temp[i] += y1\_mul\_y2[i];

file << "y1 \* (Y3^2 \* y1 + y2) \* Y3 + y1\*y2: " << endl;

fout\_vector(temp);

// ------------------------- 10 ----------------------------

temp = a\_mul\_b(&temp, &y2);

file << "(y1 \* (Y3^2 \* y1 + y2) \* Y3 + y1\*y2) \* y2: " << endl;

fout\_vector(temp);

// ------------------------- 11 ----------------------------

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

temp[i] += y2[i];

file << "RESULT!!!" << endl;

file << "y2 + ((y1 \* (Y3^2 \* y1 + y2) \* Y3 + y1\*y2) \* y2): " << endl;

fout\_vector(temp);

return(0);

}

void myPrint(string msg)

{

cout << msg << endl;

file << msg << endl << endl;

}

void myScan(double &value)

{

cin >> value;

file << "myScanned value: " << value << endl << endl;

}

void doRandomly()

{

srand(time(NULL));

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

// RANDOM Vectors

c1.push\_back(double(rand() % 10 + 1));

// RANDOM Matrices

A.push\_back(vector<double>());

A1.push\_back(vector<double>());

A2.push\_back(vector<double>());

B2.push\_back(vector<double>());

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

A[i].push\_back(double(rand() % 10 + 1));

A1[i].push\_back(double(rand() % 10 + 1));

A2[i].push\_back(double(rand() % 10 + 1));

B2[i].push\_back(double(rand() % 10 + 1));

}

}

}

void doManually()

{

int k;

ss << "Enter c1 (" << n << " elements): ";

cout << ss.str();

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

cin >> k;

c1.push\_back(k);

}

cout << endl;

ss.str(std::string());

ss << "Enter A (" << n\*n << " elements): ";

cout << ss.str();

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

A.push\_back(vector<double>());

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

cin >> k;

A[i].push\_back(k);

}

cout << endl;

}

cout << endl;

ss.str(std::string());

ss << "Enter A1 (" << n\*n << " elements): ";

cout << ss.str();

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

A1.push\_back(vector<double>());

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

cin >> k;

A1[i].push\_back(k);

}

cout << endl;

}

cout << endl;

ss.str(std::string());

ss << "Enter A2 (" << n\*n << " elements): ";

cout << ss.str();

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

A2.push\_back(vector<double>());

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

cin >> k;

A2[i].push\_back(k);

}

cout << endl;

}

cout << endl;

ss.str(std::string());

ss << "Enter B2 (" << n\*n << " elements): ";

cout << ss.str();

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

B2.push\_back(vector<double>());

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

cin >> k;

B2[i].push\_back(k);

}

cout << endl;

}

cout << endl;

ss.str(std::string());

}

void fout\_matrix(vector<vector<double>>& matrix)

{

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

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

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

}

file << endl;

}

file << endl;

}

void fout\_vector(vector<double>& vector)

{

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

file << vector[i] << " ";

}

file << endl << endl;

}

**funcs.cpp**

#include <string>

#include <vector>

using namespace std;

// returns sum up of all vector elements

double sum\_up(vector<double>\* vec)

{

double res = 0;

for (double& el : \*vec)

res += el;

return res;

}

// returns transposed matrix

vector<vector<double>> trans(vector<vector<double>>\* matrix)

{

vector<vector<double>> result; //<-- AGAIN SIGSEGV ERROR

for (int i = 0; i < matrix->size(); ++i) {

result.push\_back(vector<double>(matrix->size(), 0.0));

for (int j = 0; j < matrix->size(); ++j) {

result.at(i).at(j) += matrix->at(j).at(i);

}

}

return result;

}

vector<double> a\_mul\_b(

vector<double>\* a,

vector<double>\* b)

{

vector<double> res;

for (int i = 0; i < a->size(); ++i)

res.push\_back(a->at(i) \* b->at(i));

return res;

}

vector<double> A\_mul\_b(

vector<vector<double>>\* A,

vector<double>\* b)

{

vector<double> res;

for (int i = 0; i < A->size(); ++i) {

auto temp = a\_mul\_b(&A->at(i), b);

res.push\_back(

sum\_up(

&temp

));

}

return res;

}

vector<double> a\_mul\_B(

vector<double>\* a,

vector<vector<double>>\* B)

{

vector<double> res;

for (int i = 0; i < a->size(); ++i) {

auto temp = a\_mul\_b(a, &B->at(i));

res.push\_back(

sum\_up(

&temp

));

}

return res;

}

// if we mul 2 matrices A and B => we will get a raw matrix with columns (A\_mul\_b(A, tB->at(i)), ->->->)

vector<vector<double>> A\_mul\_B(

vector<vector<double>>\* A,

vector<vector<double>>\* B)

{

vector<vector<double>> tB(trans(B)); // transposed B

vector<vector<double>> res;

for (int i = 0; i < A->size(); ++i)

res.push\_back(

A\_mul\_b(A, &tB.at(i))

);

return res;

}