Міністерствоосвіти і науки України

Національний університет „Львівська політехніка”

Кафедра ЕОМ



**Лабораторна робота №2**

З дисципліни:”Паралельні та розподіленні обчислення ”

На тему:”Паралельне представлення алгоритмів.”

Виконав: ст.гр. КІ-33

Мельник А.О.

Прийняв: Козак Н.Б.

Львів 2020

**Мета роботи.** Вивчити можливості паралельного представлення алгоритмів. Набути навиків такого представлення.

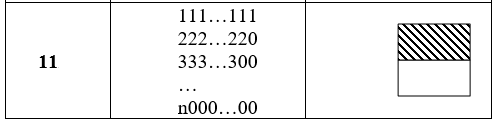
**Завдання:**

Запропонувати та реалізувати локально-рекурсивний алгоритм обчислення виразу:  , де А та В матриці з елементами  та , відповідно(),тобто:  () .

Матриця А задається однозначно і залежить лише від розмірності даних.

Для матриці В: заштрихована область – довільні цілі числа, відмінні від нуля, а незаштрихована область – нулі.

**Варіант 11**



**Порядок виконання роботи:**

Рекурсивні рівняння:

,

де= A[i][j],

= B[i][j],

r – індекс рекурсії.

Локалізований граф залежностей (n=3):

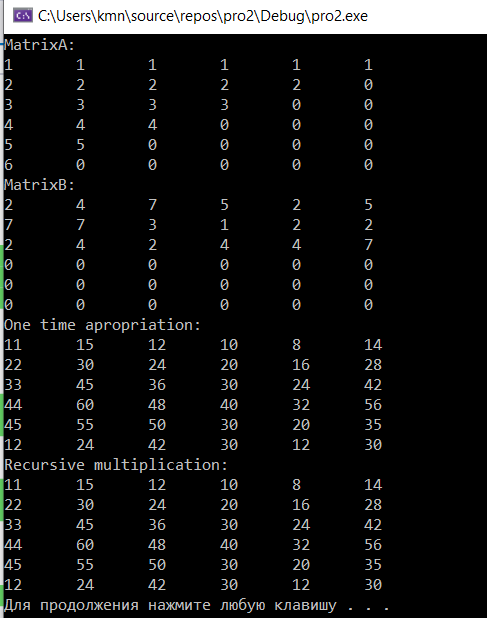


Оптимізований граф залежностей (n=3):



При використанні неоптимізованого локального рекурсивного алгоритму кількість арифметичних операцій приблизно у два рази більша, ніж при роботі алгоритму, де усунуто зайві обчислення(враховуючи тип вхідних даних). Приріст продуктивності зростає при збільшенні кількості вхідних даних. Так, для розмірності n=5 продуктивність зростає приблизно в 2 рази, а для n=9 – майже в 3 рази.

**Результати виконання роботи**



**Код програми:**

#include "pch.h"

#include <iostream>

#define MATRIX\_SIZE 6

using namespace std;

int res[MATRIX\_SIZE][MATRIX\_SIZE];

int printMatrix(int mas[MATRIX\_SIZE][MATRIX\_SIZE])

{

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

cout << mas[i][j] << "\t";

}

cout << endl;

}

return true;

}

void matrixMultiply(int a[MATRIX\_SIZE][MATRIX\_SIZE], int b[MATRIX\_SIZE][MATRIX\_SIZE]) {

static int i = 0, j = 0, k = 0;

if (i < MATRIX\_SIZE) {

if (j < MATRIX\_SIZE) {

if (k < MATRIX\_SIZE) {

res[i][j] += a[i][k] \* b[k][j];

k++;

matrixMultiply(a, b);

}

k = 0;

j++;

matrixMultiply(a, b);

}

j = 0;

i++;

matrixMultiply(a, b);

}

}

int main()

{

int matrixA[MATRIX\_SIZE][MATRIX\_SIZE];

int matrixB[MATRIX\_SIZE][MATRIX\_SIZE];

int matrixMulRes[MATRIX\_SIZE][MATRIX\_SIZE];

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

if (i +j<=MATRIX\_SIZE-1)

{

matrixA[i][j] = i+1;

}

else

{

matrixA[i][j] = 0;

}

if (MATRIX\_SIZE /2>i)

{

matrixB[i][j] = rand() % (9 - 1)+1;

}

else

{

matrixB[i][j] = 0;

}

}

}

cout << "MatrixA: " << endl;

printMatrix(matrixA);

cout << "MatrixB: " << endl;

printMatrix(matrixB);

for (int i = 0; i < MATRIX\_SIZE; i++)

{

for (int j = 0; j < MATRIX\_SIZE; j++)

{

matrixMulRes[i][j] = 0;

for (int k = 0; k < MATRIX\_SIZE; k++)

{

matrixMulRes[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

}

cout << "One time apropriation: " << endl;

printMatrix(matrixMulRes);

cout << "Matrix recursive multiplication: " << endl;

matrixMultiply(matrixA, matrixB);

printMatrix(res);

system("pause");

return 0;

}

**Висновок:** виконуючи дану лабораторну роботу вивчив можливості паралельного представлення алгоритмів. Набув навиків такого представлення.