Міністерствоосвіти і науки України

Національний університет „Львівська політехніка”

Кафедра ЕОМ



**Лабораторна робота №2**

З дисципліни:”Паралельні та розподілені обчислення ”

На тему:” **ПАРАЛЕЛЬНЕ ПРЕДСТАВЛЕННЯ АЛГОРИТМІВ**”

Виконав: ст.гр. КІ-34 Когут А.І.

Прийняв: Козак Н.Б.

Львів 2020

МЕТА РОБОТИ. Вивчити можливості паралельного представлення алгоритмів. Набути навиків такого представлення.

ВАРІАНТИ ЗАВДАНЬ

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 8**

|  |  |  |
| --- | --- | --- |
| **8** | 1 2 3 … n-1 n  2 1 2 … n-2 n-1  3 2 1 … n-3 n-2  …  n-1 n-2 n-3 … 1 2  n n-1 n-2 … 2 1 |  |

**Теоретична частина:**

**Векторизація** - це процес генерації паралельних машинних кодів на основі послідовного алгоритму, записаного на деякій мові програмування. Вона виконується, як правило векторизуючим компілятором (автоматично) і полягає у виявленні та аналізі залежностей між операторами з метою паралельного виконання незалежних, невпорядкованих дій.

**Рекурсивний алгоритм** – це алгоритм, який визначається за допомогою правила одноразового присвоювання і є стислим представленням багатьох алгоритмів. Побудова рекурсивного алгоритму зводиться до виведення рекурсивних рівнянь. Дії паралельних алгоритмів адекватно описуються в рекурсивних рівняннях з просторово-часовими індексами якщо один індекс використовується для часу, а інші – для простору (надалі – індексний простір).

**Граф залежностей** (ГЗ) - це граф, який описує залежність обчислень в алгоритмі. ГЗ може розглядатися як графічне представлення алгоритму з одноразовим присвоєнням. ГЗ називається повним, якщо він визначає всі залежності між всіма змінними в індексному просторі. Переважно, операції в вузлах графу не розкриваються, оскільки будуть виконуватися незалежними обчислювальними засобами (часто – процесорними елементами) і граф є скороченим.

**Локалізований Граф Залежностей**. Алгоритм є локалізований, якщо всі змінні безпосередньо залежать лише від змінних в сусідніх вузлах. Дані, що пересилаються незмінними до всіх вершин графу називаються передаваними, в іншому випадку – це непередавані дані.

Локально-рекурсивний алгоритм – це алгоритм, відповідний ГЗ якого має лише локальні залежності, тобто розмір задачі не впливає на довжину кожної дуги і більшість вузлів ГЗ складається з операцій одного типу.

**Хід роботи:**

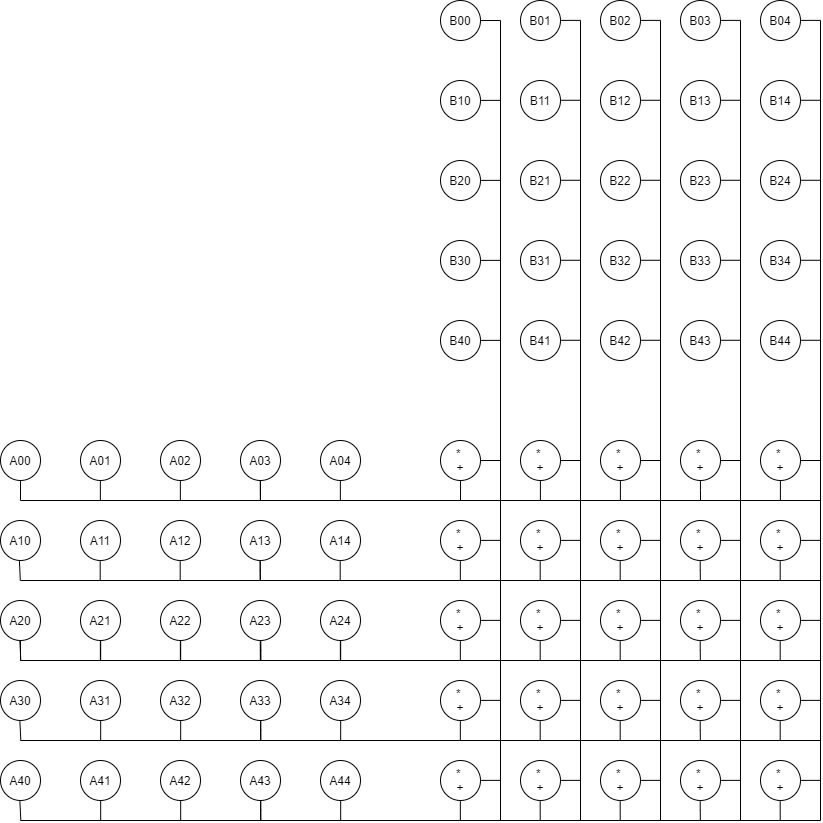


Рис.1. Граф залежностей оптимізованого множення матриці

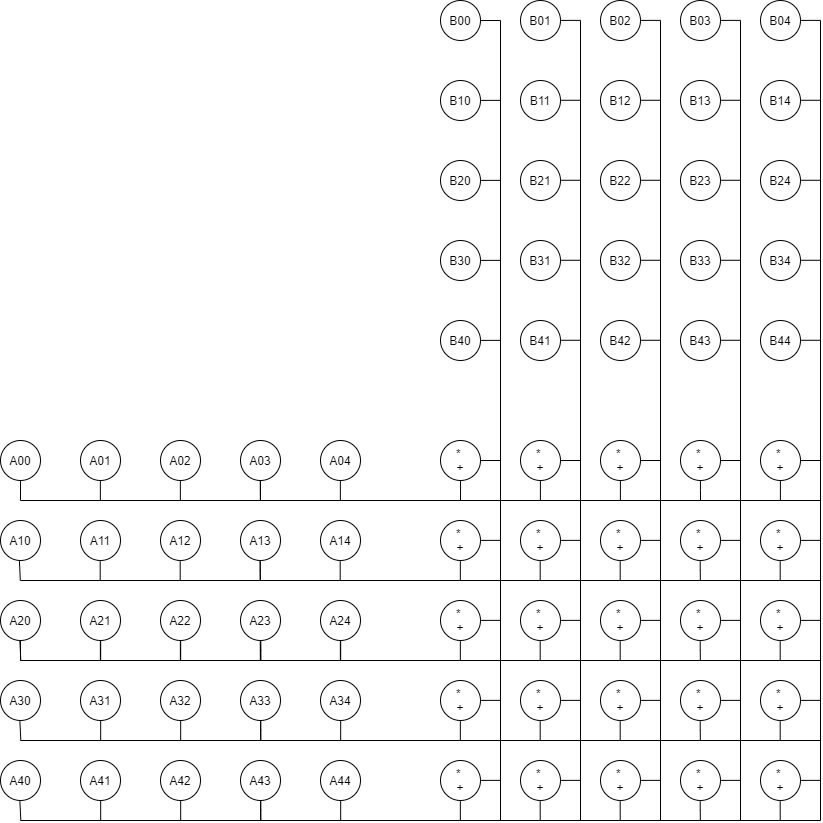


Рис.2 Граф залежностей звичайного множення матриці

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.

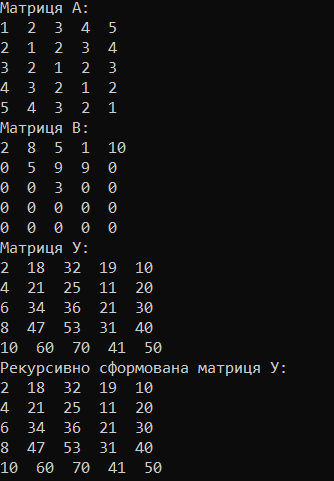


Рис.3. Результат обчислень виразу

**Висновок:** завдяки цій лабораторній роботі я зрозумів, яким чином можна представити алгоритм паралельно. Я застосував паралельне представлення. Цей метод передбачає написання рекурсивних рівнянь, побудову локалізованого та нелокалізованого графів залежностей. Виконавши оптимізацію алгоритму, я збільшив його продуктивніть більш, ніж у два рази.

Лістинг:

#include <iostream>

#include <conio.h>

#include <Windows.h>

#include "cmath"

using namespace std;

static const int n = 5;

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

static double res[n][n];

void rec(double \*a[], double \*b[])

{

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

if (i < n)

{

if (j < n)

{

if (k < n)

{

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

k++;

rec(a,b);

}

k = 0;

j++;

rec(a, b);

}

j = 0;

i++;

rec(a, b);

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

i = 0;

double\*\* A = new double\* [n];

A[i] = new double[n];

double\*\* B = new double\* [n];

B[i] = new double[n];

double\*\* Y = new double\* [n];

Y[i] = new double[n];

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

{

A[i] = new double[n];

B[i] = new double[n];

Y[i] = new double[n];

}

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

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

A[i][j] = abs(i - j) + 1;

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

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

if ((j - i >= 0) && (i + j <= n-1)) B[i][j] = rand() % 10+1;

else B[i][j] = 0;

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

{

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

{

Y[i][j] = 0;

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

{

Y[i][j] += A[i][k] \* B[k][j];

}

}

}

rec(A,B);

cout << "Матриця А:" << endl;

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

{

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

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

cout << endl;

}

cout << "Матриця В:" << endl;

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

{

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

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

cout << endl;

}

cout << "Матриця У:" << endl;

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

{

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

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

cout << endl;

}

cout << "Рекурсивно сформована матриця У:" << endl;

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

{

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

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

cout << endl;

}

}