**МІНІСТЕРСТВО ОСВІТИ 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

Заяць А.М.

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 2**

|  |  |  |
| --- | --- | --- |
| **7** | 1 2 3 … n  ….  n-2 n-1 n ....  n-1 n 0 . ..0  n 0 .0 ... 0 |  |

**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:**

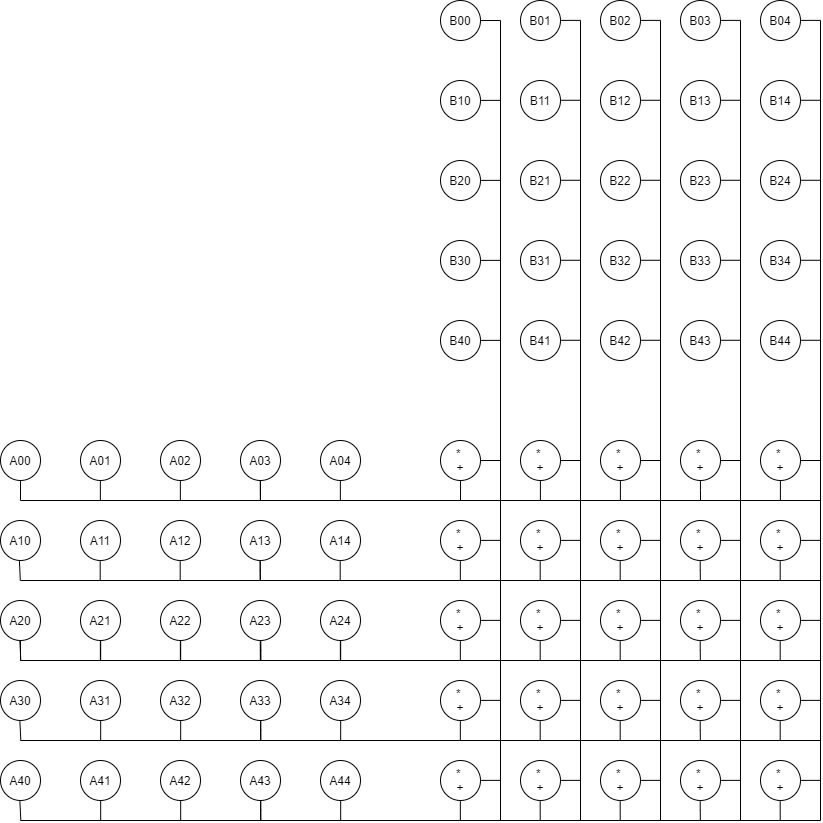


Рис.1. Граф залежностей звичайного множення матриць

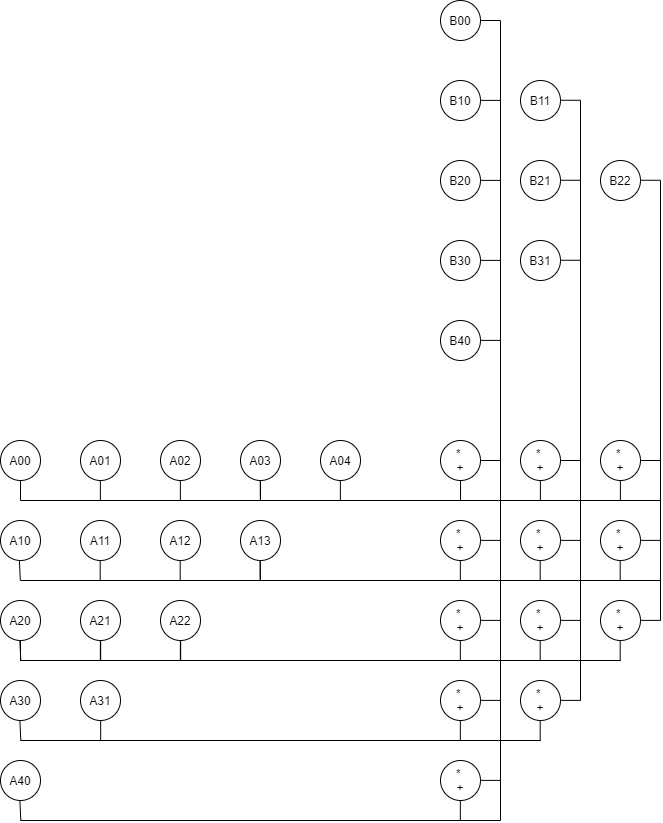


Рис.2 Граф залежностей оптимізованого множення матриці

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.

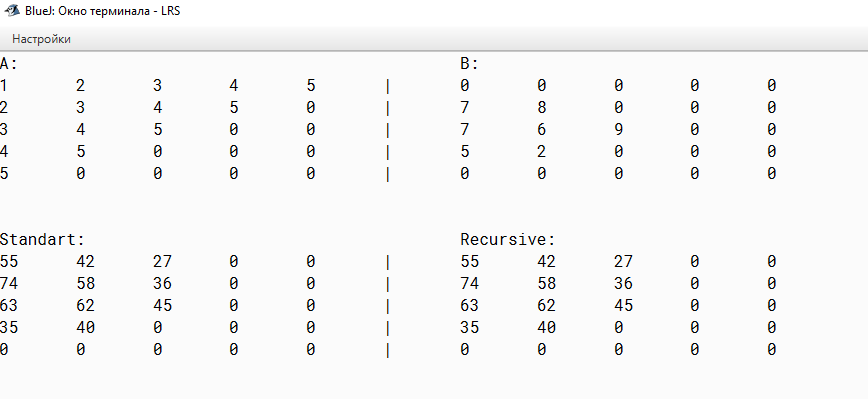


Рис.3. Результат обчислень виразу

Висновок: на цій лабораторній роботі я вивчив можливості паралельного представлення алгоритмів. Виходячи з оптимізованого графу залежностей ми отримали високу ефективність обчислень.

Лістинг:

package LRKozak;

public class LR2

{

private static int N = 5;

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

static int [][] res = new int [N][N];

static int factorial(int x)

{

int result=1;

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

{

result \*= i;

}

return result;

}

static void rec(int [][] a, int [][] b)

{

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);

}

}

static void printMatrix(int [][] mas, int [][] mas1)

{

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

{

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

{

System.out.print(mas[i][j]);

System.out.print("\t");

}

System.out.print("|\t");

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

{

System.out.print(mas1[i][j]);

System.out.print("\t");

}

System.out.print("\n");

}

}

public static void main(String[] args)

{

int [][] matrixA = new int [N][N];

int [][] matrixB = new int [N][N];

int [][] matrixMulRes = new int [N][N];

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

{

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

{

if (N - i - 1 >= j)

{

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

}

else

{

matrixA[i][j] = 0;

}

if (j <= 3)

{

if ((j <= i) & (j < 3) )

matrixB[i][j] = (int)(Math.random() \* 10);

else

matrixB[i][j] = 0;

if (((j == 1) & (i == 4)) | ((j == 2) & (i == 4)) | ((j == 2) & (i == 3)))

matrixB[i][j] = 0;

}

else

{

matrixB[i][j] = 0;

}

}

}

System.out.print("A:\t\t\t\t\t\tB: \n");

printMatrix(matrixA,matrixB);

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

{

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

{

matrixMulRes[i][j] = 0;

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

{

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

}

}

}

System.out.print("\n\nStandart:\t\t\t\t\tRecursive: \n");

rec(matrixA, matrixB);

printMatrix(matrixMulRes,res);

}

}