5МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

Кафедра дискретного аналізу та інтелектуальних систем

ПАРАЛЕЛЬНІ ТА РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ
ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №5
“АЛГОРИТМ ФЛОЙДА”

Роботу виконав:
Сенюк Віталій Васильович
Студент групи ПМІ-33с

Перевірив:
Доц. Пасічник Т.В.
кафедри програмування
Львівського національного
Університету імені Івана Франка

Львів – 2023

Тема: алгоритм Флойда.

Мета роботи: Розпаралелити виконання алгоритму Флойда.

Завдання

Для орієнтованого зваженого графа G(V,F), де V={a0, a1,…an} – множина вершин (n –велике число), а F множина орієнтованих ребер (шляхів) між вершинами, використовуючи алгоритм Флойда, знайти найкоротший шлях між заданими вузлами a та в.

Для різної розмірності графів та довільних вузлів a та в порахувати час виконання програми без потоків та при заданих k потоках розпаралелення.

Виконання роботи

1. Створюю клас для роботи з графом у виді матриці.

```
public class Graph
{
    private int[,] adjacencyMatrix;

    1 reference
    public Graph(int numberOfVertices)
    {
        Random random = new Random();

        // Initialize the adjacency matrix with random weights for edges
        adjacencyMatrix = new int[numberOfVertices, numberOfVertices];
        for (int i = 0; i < numberOfVertices; i++)
        {
            for (int j = 0; j < numberOfVertices; j++)
            {
                // Assign a random weight between 0 and 100 for the edges
                adjacencyMatrix[i, j] = random.Next(101); // 0 to 100
            }
        }
    }

    1 reference
    public int[,] GetAdjacencyMatrix()
    {
        return adjacencyMatrix;
    }
}
```

2. Створюю алгоритм для лінійного обчислення алгоритму Флойда.

```
static int[,] FloydWarshall(int[,] graph, int n)
{
    int[,] dist = new int[n, n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            dist[i, j] = graph[i, j];
        }
    }
    for (int k = 0; k < n; k++)
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (dist[i, k] != int.MaxValue && dist[k, j] != int.MaxValue &&
                    dist[i, k] + dist[k, j] < dist[i, j])
                {
                    dist[i, j] = dist[i, k] + dist[k, j];
                }
            }
        }
    }
    return dist;
}
```

3. Створюю паралельний алгоритм.

```
static int[,] ParallelFloydWarshall(int[,] graph, int n, int numberOfThreads)
{
    int[,] dist = new int[n, n];
    Parallel.For(0, n, i =>
    {
        for (int j = 0; j < n; j++)
        {
            dist[i, j] = graph[i, j];
        }
    });
            Parallel.For(0, n, new ParallelOptions { MaxDegreeOfParallelism = numberOfThreads }, k =>
    {
        Parallel.For(0, n, new ParallelOptions { MaxDegreeOfParallelism = numberOfThreads }, i =>
        {
            for (int j = 0; j < n; j++)
            {
                if (dist[i, k] != int.MaxValue && dist[k, j] != int.MaxValue &&
                    dist[i, k] + dist[k, j] < dist[i, j])
                {
                    dist[i, j] = dist[i, k] + dist[k, j];
                }
            }
        });
    });

    return dist;
}
```

З використанням Parallel.For та ParallelOptions алгоритм легко розпаралелюється.

4. Час та параметри на малому графі розміром 100 вершин.

```
GraphSize 100
Iterational 'a' (0) and 'b' (3) is: 2, time:00:00:00.0222338
Parallel t=2 'a' (0) and 'b' (3) is: 2 time: 00:00:00.0481263 acceleration: 0,46198855927008725 efficiency: 0,23099427963504363
Parallel t=4 'a' (0) and 'b' (3) is: 2 time: 00:00:00.0455969 acceleration: 0,48761648269948177 efficiency: 0,12190412067487044
Parallel t=8 'a' (0) and 'b' (3) is: 2 time: 00:00:00.0178386 acceleration: 1,246387048310966 efficiency: 0,15579838103887075
```

Як бачимо, на малих даних прискорення ледве відчувається при 8 потоках, а при 4 та 2 – взагалі менше 1. Даний алгоритм розпаралелення не ефективний на малих графах.

5. Час та параметри на середньому графі розміром 500 вершин.

```
GraphSize 500
Iterational 'a' (0) and 'b' (3) is: 0, time:00:00:03.3412148
Parallel t=2 'a' (0) and 'b' (3) is: 0 time: 00:00:01.0951486 acceleration: 3,050923682868243 efficiency: 1,5254618414341214
Parallel t=4 'a' (0) and 'b' (3) is: 0 time: 00:00:00.9552254 acceleration: 3,497828680016256 efficiency: 0,874457170004064
Parallel t=8 'a' (0) and 'b' (3) is: 0 time: 00:00:01.0052943 acceleration: 3,3236185662248356 efficiency: 0,41545232077810446
```

Як бачимо, прискорення відчувається набагато більше, слід зазначити що складність алгоритму O(n^3), тому якщо збільшити розмір графа всього в 5 разі, час збільшиться у 3. Прискорення залишається однаковим у всіх варіантах (враховуючи погрішність), то ефективність починає падати.

6. Час та параметри на великому графі розміром 2500 вершин.

```
GraphSize 2500
Iterational 'a' (0) and 'b' (3) is: 0, time:00:03:42.3864261
Parallel t=2 'a' (0) and 'b' (3) is: 0 time: 00:02:09.5243395 acceleration: 1,7169469997567524 efficiency: 0,8584734998783762
Parallel t=4 'a' (0) and 'b' (3) is: 0 time: 00:01:38.8636736 acceleration: 2,249425071940681 efficiency: 0,5623562679851702
Parallel t=8 'a' (0) and 'b' (3) is: 0 time: 00:01:36.2596896 acceleration: 2,310275744957316 efficiency: 0,2887844681196645
```

У випадку з великим графом, прискорення та ефективність зменшуються порівняно з середнім графом. Це відбувається через те, що при довгому навантаженні процесора, його частота зменшується до базової 2.4 ghz. Так як при меншому графі частота ще не встигла перескочити з 3+ ghz на 2.4 ghz, то отримуємо кращий результат.

Висновки

У результаті виконання роботи:

1. Дізнався про Parralel.For.
2. Використав Parralel.For для обчислення зображення найкоротшого шляху між двома вершинами у графі.

Список використаних джерел

1. Вільний простір інтернету.