Міністерство освіти і науки України

Львівський національний університет Ім. І. Франка

Факультет прикладної математики та інформатики

Кафедра дискретного аналізу та інтелектуальних систем

Паралельні та розподілені обчислення

Лабораторна робота №5

Алгоритм Флойда

Роботу виконала:

Студентка ПМІ-33

Багінська Маргарита

Прийняв:

доц. Пасічник Т.В.

Львів 2023

Тема: розв'язання задачі про найкоротший шлях в орієнтованому зваженому графі з додатними або від'ємними вагами ребер за допомогою алгоритму Флойда-Воршелла.

Мета: Реалізувати послідовний та паралельний алгоритм розв'язання задачі знаходження найкоротшого шляху в орієнтованому зваженому графі з додатними або від'ємними вагами ребер за допомогою алгоритму Флойда-Воршелла.

Хід роботи

Загалом програма ініціалізує спрямований, зважений граф G двовимірним масивом, що представляє вагу ребер між кожною парою вершин. Використовую значення INF (99999) для представлення «відсутності ребра» між вузлами та 0 для петель.

Допоміжна функція:

```
static int INF = 99999;
private static int[,] GenerateGraph(int size)
{
    int[,] graph = new int[size, size];
    Random rnd = new Random();

    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (i == j)
            {
                graph[i, j] = 0;
            }
            else
            {
                // random weights from -5 to 10, or set to INF with a 50% probability
                graph[i, j] = rnd.Next(-5, 11);
                if (rnd.NextDouble() < 0.5)
                {
                    graph[i, j] = INF;
                }
            }
        }
    }

    return graph;
}
```

Функція GenerateGraph приймає цілочисельний розмір як вхідні дані та ініціалізує порожній граф N x N. Потім проходить по графу, призначаючи випадкові ваги від -5 до 10 ребрам або встановлюючи для них значення INF (імовірність 50%), щоб відобразити, що немає ребра між двома вершинами графу. На петлі у вершинах встановлюється на 0.

Приклад згенерованих даних:

```
int[,] graph = {
     {0, 5, INF, 10},
     {INF, 0, 3, INF},
     {INF, INF, 0, 1},
     {INF, INF, INF, 0}
 };
```

Послідовний алгоритм

```
public static TimeSpan FloydOneThread(int[,] graph, int a, int b)
{
    Stopwatch stopWatch = new Stopwatch();
    stopWatch.Start();

    int n = graph.GetLength(0);
    int[,] dist = new int[n, n];

    // Initialize the distance matrix with the input graph
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            dist[i, j] = graph[i, j];
        }
    }

    // Main loop: Floyd-Warshall algorithm
    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] + dist[k, j] < dist[i, j])
                {
                    dist[i, j] = dist[i, k] + dist[k, j];
                }
            }
        }
    }

    stopWatch.Stop();

    Console.WriteLine("Sync time ~ " + stopWatch.Elapsed.ToString());
    Console.WriteLine("Shortest path between " + a + " and " + b + ": " + dist[a, b] + "\n");
    return stopWatch.Elapsed;
}
```

Метод floydWarshall бере вхідний граф і обчислює найкоротший шлях між усіма парами вершин за допомогою алгоритму Floyd-Warshall. Метод повертає двовимірний масив найкоротших шляхів (матрицю відстаней) між усіма парами вершин.

Масив відстаней dist ініціалізується вагами ребер вхідного графа. Це робиться за допомогою двох вкладених циклів for, які перебирають кожен елемент у графі матриці суміжності.

Алгоритм Флойда-Воршалла реалізовано за допомогою трьох вкладених циклів for. Для кожної пари вершин (i, j), якщо існує кращий шлях через вершину k, кращий шлях оновлюється в масиві dist, тобто dist[i,j] = dist[i,k] + dist[k ,j].

Найкоротший шлях між вузлами a і b можна знайти, звернувшись до відповідного запису в масиві найкоротших шляхів dist[a][b].

Переконаємось у правильності обчислень для графу згаданого вище:

```
Sync time ~ 00:00:00.0001836
Shortest path between 1 and 3: 4
```

Алгоритм працює коректно.

Паралельний алгоритм

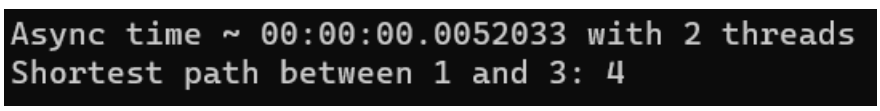
```
public static TimeSpan FloydMultiThread(int[,] graph, int ThreadNum, int a, int b)
 {

     Stopwatch stopWatch = new Stopwatch();
     stopWatch.Start();
     int n = graph.GetLength(0);
     int[,] dist = new int[n, n];

     // Initialize the distance matrix with the input graph
     for (int i = 0; i < n; i++)
     {
         for (int j = 0; j < n; j++)
         {
             dist[i, j] = graph[i, j];
         }
     }

     // Main loop: Floyd-Warshall algorithm with parallelization
     for (int k = 0; k < n; k++)
     {
         Parallel.For(0, n, new ParallelOptions { MaxDegreeOfParallelism = ThreadNum }, i =>
         {
             for (int j = 0; j < n; j++)
             {
                 if (dist[i, k] + dist[k, j] < dist[i, j])
                 {
                     dist[i, j] = dist[i, k] + dist[k, j];
                 }
             }
         });
     }

     stopWatch.Stop();

     Console.WriteLine($"Async time ~ {stopWatch.Elapsed} with {ThreadNum} threads");
     Console.WriteLine("Shortest path between " + a + " and " + b + ": " + dist[a, b]+"\n");
     return stopWatch.Elapsed;
}
```

Функція виконує алгоритм Флойда-Варшалла, використовуючи кілька потоків на заданому зваженому орієнтованому графі. Кількість потоків, які використовуються для паралельного виконання, визначається параметром ThreadNum.

Масив відстаней dist ініціалізується вагами ребер вхідного графа. Це робиться за допомогою двох вкладених циклів for, які перебирають кожен елемент у графі матриці суміжності.

Розпаралелений алгоритм Флойда-Варшалла реалізовано за допомогою методу Parallel.For із простору імен System.Threading.Tasks із зазначенням властивості MaxDegreeOfParallelism для керування кількістю потоків (ThreadNum). Основний цикл алгоритму включає в себе зовнішній цикл, що виконується по k, і внутрішній вкладений цикл, який ітерує по парах вершин (i, j). Якщо існує кращий шлях через вершину k, кращий шлях оновлюється в масиві dist, тобто dist[i,j] = dist[i,k] + dist[k,j].

Переконаємось у правильності обчислень для графу згаданого вище:



Паралельний алгоритм також працює коректно.

Прискорення Sp для паралельного алгоритму визначається відношенням часової складності послідовного T1 та паралельного алгоритмів для p процесорів Sp = T1 / Tp. (Sp > 1 Оптимально).

Ефективність Ep для паралельного алгоритму визначається прискоренням цього алгоритму відносно кількості процесорів: Ep= Sp/p Ідеал: Ep(n) = 1.

Результати

На малій розмірності графа розпаралелення не є оптимальним, як і у попередніх лабораторних роботах.

```
N = 5
Sync time ~ 00:00:00.0001914
Shortest path between 3 and 2: 8

Async time ~ 00:00:00.0530993 with 3 threads
Shortest path between 3 and 2: 8

Acceleration ~ 0,0036045672918475387
Efficiency ~ 0,0012015224306158462
```

Зі збільшенням розмірності паралельність ефективніша.

```
N = 150
Sync time ~ 00:00:00.0192064
Shortest path between 1 and 3: 2

Async time ~ 00:00:00.0191442 with 3 threads
Shortest path between 1 and 3: 2

Acceleration ~ 1,0032490258146 07
Efficiency ~ 0,3344163419382023
```

```
N = 900
Sync time ~ 00:00:03.9678620
Shortest path between 1 and 4: 3

Async time ~ 00:00:01.0581651 with 10 threads
Shortest path between 1 and 4: 3

Acceleration ~ 3,7497570086180314
Efficiency ~ 0,37497570086180315
```

Можемо спостерігати ефективність паралельного алгоритму при великих об’ємах та розумній кількості потоків.

```
N = 2500
Sync time ~ 00:01:31.3474379
Shortest path between 1 and 3: 1

Async time ~ 00:00:27.9385861 with 8 threads
Shortest path between 1 and 3: 1

Acceleration ~ 3,2695798410500094
Efficiency ~ 0,4086974801312512
```

Висновок: У результаті виконання лабораторної роботи було реалізовано послідовний та паралельний алгоритм розв'язання задачі знаходження найкоротшого шляху в орієнтованому зваженому графі з додатними або від'ємними вагами ребер за допомогою алгоритму Флойда-Воршелла мовою програмування C# та за допомогою методу Parallel.For із простору імен System.Threading.Tasks. Переконались у ефективності розпаралелення процесу у даному випадку.