МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Паралельні та розподілені обчислення
ЛАБОРАТОРНА РОБОТА №5
«Алгоритм Флойда»

Виконала:
студентка групи ПМі-31
Дудчак Валентина Юріївна

Львів 2024

Тема: Розпаралелення алгоритму Флойда-Варшалла

Мета: Написати програми розв’язування алготму Флойда (послідовний та паралельний алгоритми). Для орієнтованого зваженого графа G(V,F) знайти найкоротший шлях між заданими вузлами *а* та *в*. Порахувати час роботи кожної з програм, обчислити прискорення та ефективність роботи паралельного алгоритму.

Хід роботи:

Для задачі я подаю граф у вигляді двовимірної квадратної матриці n x n, де індекси i та j – вершини, а значення на перетині – вага ребра. Для вершин, між якими немає ребра, присвоюється значення MAX_VALUE. Для петель – значення 0.

Послідовний алгоритм

Послідовний алгоритм Флойда реалізовано у методі SequentialFloydAlgorithm.

У ньому обчислюється найкоротший шлях між усіма парами вершин за допомогою алгоритму Флойда-Варшалла та зберігається у матрицю. Використано також об’єкт класу Stopwatch для засікання часу. Метод повертає об’єкт stopwatch та значення відстані між двома заданими вершинами:

```
1 reference
public static (Stopwatch, int) SequentialFloydAlgorithm(int[,] graph, int a = 0, int b = 0)
{
  int n = graph.GetLength(0);

  Stopwatch stopWatch = new Stopwatch();
  stopWatch.Start();

  int[,] resultDistances = new int[n, n];

  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      resultDistances[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 (resultDistances[i, k] + resultDistances[k, j] < resultDistances[i, j])
        {
          resultDistances[i, j] = resultDistances[i, k] + resultDistances[k, j];
        }
      }
    }
  }

  stopWatch.Stop();
  return (stopWatch, resultDistances[a, b]);
}
```

Крім цього, я перевірила, чи результати обчислюються правильно:

```
0 18 18 14 17
10 0 10 19 12
3 10999 0 10999 14
10 10999 12 0 1
12 10999 18 5 0

n = 5, a = 1, b = 2
Sequential time: 0 ms, dist: 10
Parallel time: 32 ms, threads: 10, dist: 10
```

Паралельний алгоритм

Паралельний алгоритм Флойда реалізовано у методі ParallelFloydAlgorithm.

Використано об’єкт класу Tasks, зокрема метод Parallel.For, для розпаралелення обчислення найкоротшого шляху:

```
1 reference
public static (Stopwatch, int) ParallelFloydAlgorithm(int[,] graph, int threadCount, int a = 0, int b = 0)
{
  int n = graph.GetLength(0);

  Stopwatch stopWatch = new Stopwatch();
  stopWatch.Start();

  int[,] resultDistances = new int[n, n];

  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n; j++)
    {
      resultDistances[i, j] = graph[i, j];
    }
  }

  for (int k = 0; k < n; k++)
  {
    Parallel.For(0, n, new ParallelOptions { MaxDegreeOfParallelism = threadCount }, i =>
    {
      for (int j = 0; j < n; j++)
      {
        if (resultDistances[i, k] + resultDistances[k, j] < resultDistances[i, j])
        {
          resultDistances[i, j] = resultDistances[i, k] + resultDistances[k, j];
        }
      }
    });
  }

  stopWatch.Stop();
  return (stopWatch, resultDistances[a, b]);
}
```

Метод також працює правильно (видно у результатах вище)

Аналіз результатів

На малих розмірах графа та великій кількості потоків розпаралелення неефективне:

```
n = 5, a = 1, b = 2
Sequential time: 0 ms, dist: 12
Parallel time: 22 ms, threads: 5, dist: 12
Acceleration: 0.012109114515480853
Efficiency: 0.0024218229030961705
```

```
n = 10, a = 1, b = 2
Sequential time: 0 ms, dist: 6
Parallel time: 29 ms, threads: 5, dist: 6
Acceleration: 0.009116938169107879
Efficiency: 0.0018233876338215756
```

```
n = 100, a = 1, b = 2
Sequential time: 14 ms, dist: 2
Parallel time: 33 ms, threads: 10, dist: 2
Acceleration: 0.4248640402639015
Efficiency: 0.04248640402639015
```

На більших розмірностях розпаралелення дає значно кращий результат:

```
n = 300, a = 1, b = 2
Sequential time: 242 ms, dist: 3
Parallel time: 160 ms, threads: 10, dist: 3
Acceleration: 1.5128438420448305
Efficiency: 0.15128438420448304
```

```
n = 900, a = 1, b = 2
Sequential time: 5943 ms, dist: 1
Parallel time: 3300 ms, threads: 10, dist: 1
Acceleration: 1.8009256119960702
Efficiency: 0.180092561199607
```

```
n = 1500, a = 1, b = 2
Sequential time: 27629 ms, dist: 2
Parallel time: 14803 ms, threads: 20, dist: 2
Acceleration: 1.8663544803517145
Efficiency: 0.09331772401758573
```

При великій кількості потоків ефективність трохи покращується:

```
n = 900, a = 1, b = 2
Sequential time: 5896 ms, dist: 3
Parallel time: 3271 ms, threads: 111, dist: 3
Acceleration: 1.802466714192711
Efficiency: 0.016238438866601
```

Коли розмірності графа не кратна кількості потоків, ефективність незначно погіршується:

```
n = 400, a = 1, b = 2
Sequential time: 527 ms, dist: 2
Parallel time: 299 ms, threads: 10, dist: 2
Acceleration: 1.759651418971997
Efficiency: 0.1759651418971997
```

```
n = 400, a = 1, b = 2
Sequential time: 552 ms, dist: 3
Parallel time: 315 ms, threads: 11, dist: 3
Acceleration: 1.751976802139702
Efficiency: 0.15927061837633655
```

Висновок:

Виконуючи лабораторну роботу, я навчилася розпаралелювати алгоритми за допомогою Tasks.Parallel.For і взнала, у яких випадках це ефективно робити.