МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Паралельні та розподілені обчислення

ЛАБОРАТОРНА РОБОТА №1
«Додавання та віднімання матриць»

Виконала:
студентка групи ПМі-31
Дудчак Валентина Юріївна

Львів 2024

Тема: Розпаралелення додавання/віднімання матриць

Мета: Написати програми обчислення суми/різниці двох матриць (послідовний та паралельний алгоритми). Порахувати час роботи кожної з програм, обчислити прискорення та ефективність роботи паралельного алгоритму.

Послідовний алгоритм:

Послідовний алгоритм додавання/віднімання матриць реалізовано у методах SequentialMatrixAddition та SequentialMatrixSubstraction. У них використані вкладені цикли для обрахунків матриць та об’єкт класу Stopwatch для засікання часу (у подальшому потрібно для аналізу результатів):

```
1 reference
public static TimeSpan SequentialMatrixAddition(int[,] matrixA, int[,] matrixB)
{
  int n = matrixA.GetLength(0);
  int m = matrixA.GetLength(1);
  int[,] resultMatrix = new int[n, m];

  Stopwatch stopWatch = new Stopwatch();
  stopWatch.Start();
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < m; j++)
    {
      resultMatrix[i, j] = matrixA[i, j] + matrixB[i, j];
    }
  }
  stopWatch.Stop();

  return stopWatch.Elapsed;
}
```

Паралельний алгоритм:

Паралельний алгоритм додавання/віднімання матриць реалізовано у методі ParallelMatrixComputation. Використано об’єкт Thread для розпаралелення процесу обчислення на рядки та функції SumMatrixRows/SubMatrixRows, у яких обчислюються лише потрібні рядки матриці. Кожен потік обчислює цілу частку від ділення кількості рядків на кількість потоків + перші потоки обчислюють ще по одному рядку з залишкових для кращої рівномірності.

```
1 reference
public static TimeSpan ParallelMatrixComputation(int[,] matrixA, int[,] matrixB, int threadCount, Boolean isSum)
{
  int n = matrixA.GetLength(0);
  int m = matrixA.GetLength(1);
  int[,] resultMatrix = new int[n, m];

  Thread[] threads = new Thread[threadCount];
  Stopwatch stopWatch = new Stopwatch();
  stopWatch.Start();

  int baseRowsPerThread = n / threadCount;
  int extraRows = n % threadCount;
```

```
  void SumMatrixRows(int startRow, int endRow)
  {
    for (int i = startRow; i < endRow; i++)
    {
      for (int j = 0; j < m; j++)
      {
        resultMatrix[i, j] = matrixA[i, j] + matrixB[i, j];
      }
    }
  }

  void SubMatrixRows(int startRow, int endRow)
  {
    for (int i = startRow; i < endRow; i++)
    {
      for (int j = 0; j < m; j++)
      {
        resultMatrix[i, j] = matrixA[i, j] - matrixB[i, j];
      }
    }
  }
```

```
Action<int, int> operation = isSum ? (Action<int, int>)SumMatrixRows : SubMatrixRows;

for (int i = 0; i < threadCount; i++)
{
  int startRow = i * baseRowsPerThread + Math.Min(i, extraRows);
  int endRow = startRow + baseRowsPerThread + (i < extraRows ? 1 : 0);

  threads[i] = new Thread(() => operation(startRow, endRow));
  threads[i].Start();
}

foreach (var thread in threads)
{
  thread.Join();
}

stopWatch.Stop();
return stopWatch.Elapsed;
}
```

Крім цих двох методів, реалізована також функція **PrintMatrix**, за допомогої якої я перевірила, чи функції додавання/віднімання обчислюються правильно:

```
0 references
public static void PrintMatrix(int[,] m)
{
  for (int i = 0; i < m.GetLength(0); i++)
  {
    for (int j = 0; j < m.GetLength(1); j++)
    {
      Console.Write(m[i, j] + " ");
    }
    Console.WriteLine();
  }
  Console.WriteLine();
}
```

```
11 64 3
16 74 17
   +
83 65 83
33 93 44
   =
94 129 86
49 167 61
```

```
51 66 46
96 60 43
   -
88 58 74
76 61 31
   =
-37 8 -28
20 -1 12
```

Аналіз результатів:

На малих розмірах матриці розпаралелення для додавання матриць неефективне, і стає все більш неефективним зі зростанням кількості потоків:

```
n = 10, m = 10
Sequential time: 1989 ticks
Parallel time: 6560 ticks, threads: 1
Acceleration of parallel: 0.303201219512195 14
Efficiency of parallel: 0.303201219512195 14
```

```
n = 10  m = 10
Sequential time: 1863 ticks
Parallel time: 5893 ticks, threads: 3
Acceleration of parallel: 0.316137790599015 76
Efficiency of parallel: 0.105379263533005 25
```

```
n = 10, m = 10
Sequential time: 1665 ticks
Parallel time: 12105 ticks, threads: 10
Acceleration of parallel: 0.137546468401487
Efficiency of parallel: 0.0137546468401487
```

На більших розмірностях розпаралелення дає значно кращий результат:

```
n = 10000, m = 10000
Sequential time: 792 ms
Parallel time: 310 ms, threads: 10
Acceleration of parallel: 2.554712240892089
Efficiency of parallel: 0.255471224089208 86
```

```
n = 30000, m = 20000
Sequential time: 752 ms
Parallel time: 49 ms, threads: 10
Acceleration of parallel: 3.2954729202446265
Efficiency of parallel: 0.32954729202446265
```

При ~10 потоках для ~20000 рядків ефективність є найбільшою, проте з більшим зростанням кількості потоків погіршується:

```
n = 30000, m = 20000
Sequential time: 670 ms
Parallel time: 111 ms, threads: 50
Acceleration of parallel: 3.158187938672758
Efficiency of parallel: 0.063163758773455 16
```

Додатково я порівняла те, як на ефективність впливає кратність розмірності матриці до кількості потоків. Бачимо, що не кратні значення погіршують ефективність паралельного алгоритму, хоча додалася невелика кількість потоків:

```
n = 30000, m = 20000
Sequential time: 675 ms
Parallel time: 85 ms, threads: 100
Acceleration of parallel: 3.2007213222873907
Efficiency of parallel: 0.032007213222873906
```

```
n = 30000, m = 20000
Sequential time: 574 ms
Parallel time: 107 ms, threads: 101
Acceleration of parallel: 3.1188875841072954
Efficiency of parallel: 0.030880075090171242
```

```
n = 30000, m = 20000
Sequential time: 597 ms
Parallel time: 179 ms, threads: 111
Acceleration of parallel: 3.0273242854562086
Efficiency of parallel: 0.027273191760866743
```

Для алгоритму віднімання матриць результати такі ж самі:

```
n = 10, m = 10
Sequential time: 1579 ticks
Parallel time: 6239 ticks, threads: 1
Acceleration of parallel: 0.25308543035742 91
Efficiency of parallel: 0.25308543035742 91
```

```
n = 10, m = 10
Sequential time: 2321 ticks
Parallel time: 13685 ticks, threads: 10
Acceleration of parallel: 0.16960175374497624
Efficiency of parallel: 0.016960175374497625
```

```
n = 10000, m = 10000
Sequential time: 784 ms
Parallel time: 297 ms, threads: 10
Acceleration of parallel: 2.6407192220412226
Efficiency of parallel: 0.26407192220412223
```

```
n = 30000, m = 20000
Sequential time: 596 ms
Parallel time: 26 ms, threads: 10
Acceleration of parallel: 3.255895926136939
Efficiency of parallel: 0.3255895926136939
```

```
n = 30000, m = 20000
Sequential time: 666 ms
Parallel time: 112 ms, threads: 111
Acceleration of parallel: 3.1559093891051098
Efficiency of parallel: 0.02843161611806405
```

Висновок:

Виконуючи лабораторну роботу, я навчилася розпаралелювати алгоритми за допомогою Thread та взнала, у яких випадках це ефективно робити.