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 №1

Додавання (віднімання) матриць

Роботу виконала:

Студентка ПМІ-33

Багінська Маргарита

Прийняв:

доц. Пасічник Т.В.

Львів 2023

Тема: Додавання (віднімання) матриць.

Мета: Реалізувати послідовний та паралельний алгоритм додавання (віднімання) двох матриць.

Послідовний алгоритм

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

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

    Console.WriteLine("Sync time ~ " + stopWatch.Elapsed.ToString());
    return stopWatch.Elapsed;
}
```

У послідовному алгоритмі проходимось по двом матрицям двома вкладеними циклами та результат додавання записуємо у результуючу матрицю, засікаємо час.

Паралельний алгоритм

```
public static TimeSpan AsyncMethod(int[,] matrixA, int[,] matrixB, int threadNum)
 {

     int n = matrixA.GetLength(0);
     int m = matrixA.GetLength(1);
     int[,] result = new int[n, m];

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

     Thread[] threads = new Thread[threadNum];
     Stopwatch stopWatch = new Stopwatch();
     stopWatch.Start();
     int rowsPerThread = n / threadNum;
     for (int i = 0; i < threadNum; i++)
     {
         int startRow = i * rowsPerThread;
         int endRow = (i == threadNum - 1) ? n : (i + 1) * rowsPerThread;

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

     foreach (var thread in threads)
     {
         thread.Join();
     }
     stopWatch.Stop();
     Console.WriteLine($"Async time ~ {stopWatch.Elapsed} with {threadNum} threads");
     return stopWatch.Elapsed;
}
```

У розпаралеленому алгоритмі додавання двох матриць за допомогою потоків Threads ми використовуємо цикл for, щоб розділити обчислення рядків матриці між потоками. Кожен потік обчислює певний діапазон рядків (підматрицю) від startRow до endRow, де startRow та endRow обчислюються на основі кількості потоків та розмірності матриці.

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

```
N = 20  M = 30
Sync time ~ 00:00:00.0001612
Async time ~ 00:00:00.0100642 with 3 threads
Acceleration ~ 0,016017169770076112
Efficiency ~ 0,005339056590025371
```

На малих розмірностях матриць розпаралелення не є оптимальним.

```
N = 2000  M = 3000
Sync time ~ 00:00:00.0392642
Async time ~ 00:00:00.0269286 with 3 threads
Acceleration ~ 1,4580854556122487
Efficiency ~ 0,48602848520408287
```

Зі збільшенням розмірності паралельність ефективніша.

```
N = 20000  M = 30000
Sync time ~ 00:00:03.9485778
Async time ~ 00:00:01.8722769 with 10 threads
Acceleration ~ 2,108971060851095
Efficiency ~ 0,21089710608510953
```

```
N = 20000  M = 30000
Sync time ~ 00:00:04.1895597
Async time ~ 00:00:01.2035514 with 9 threads
Acceleration ~ 3,480997737196766
Efficiency ~ 0,38677752635519624
```

```
N = 20000  M = 30000
Sync time ~ 00:00:03.9793755
Async time ~ 00:00:01.9056139 with 30 threads
Acceleration ~ 2,088238073830171
Efficiency ~ 0,06960793579433903
```

Висновок: У результаті виконання лабораторної роботи було реалізовано послідовний та паралельний алгоритм додавання (віднімання) двох матриць мовою програмування C# та класу Thread. Переконались у ефективності розпаралелення процесу у даному випадку.