МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Паралельні та розподілені обчислення

ЛАБОРАТОРНА РОБОТА №2
«Множення матриць»

Виконала:
студентка групи ПМі-31
Дудчак Валентина Юріївна

Львів 2024

Тема: Розпаралелення множення матриць

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

Послідовний алгоритм

Послідовний алгоритм множення матриць реалізовано у методі SequentialMatrixMultiplication.

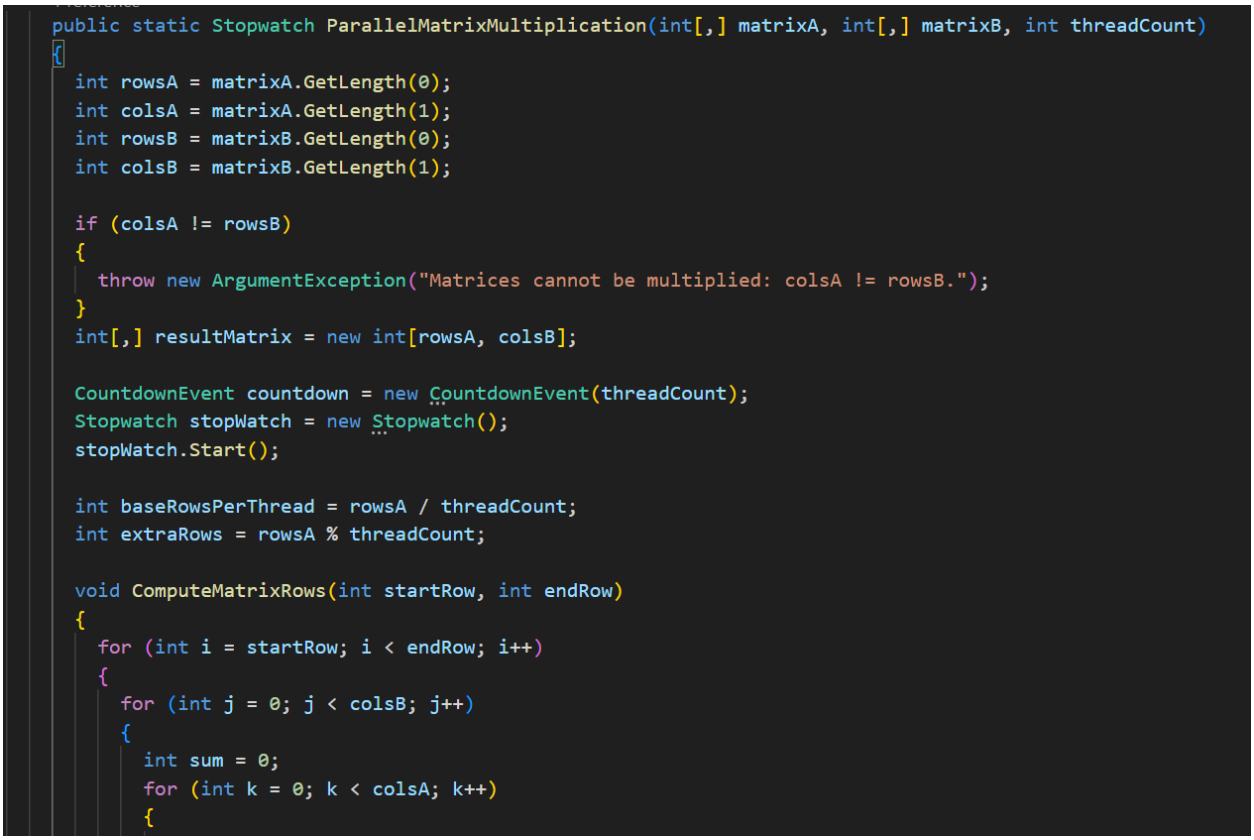
У ньому використані вкладені цикли для обрахунків та об’єкт класу Stopwatch для засікання часу. Також перевіряється, чи кількість колонок першої матриці дорівнює кількості рядків другої:

```
public static Stopwatch SequentialMatrixMultiplication(int[,] matrixA, int[,] matrixB)
{
  int rowsA = matrixA.GetLength(0);
  int colsA = matrixA.GetLength(1);
  int rowsB = matrixB.GetLength(0);
  int colsB = matrixB.GetLength(1);

  if (colsA != rowsB)
  {
    throw new ArgumentException("Matrices cannot be multiplied: colsA != rowsB.");
  }
  int[,] resultMatrix = new int[rowsA, colsB];

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

  for (int i = 0; i < rowsA; i++)
  {
    for (int j = 0; j < colsB; j++)
    {
      int sum = 0;
      for (int k = 0; k < colsA; k++)
      {
        sum += matrixA[i, k] * matrixB[k, j];
      }
      resultMatrix[i, j] = sum;
    }
  }
  stopWatch.Stop();
  return stopWatch;
```

Паралельний алгоритм

Паралельний алгоритм множення матриць реалізовано у методі ParallelMatrixMultiplication.

Використано об’єкт Thread для розпаралелення процесу обчислення на рядки та об’єкт класу CountdownEvent, щоб упевнитися, що всі потоки завершили свою роботу. Кожен потік обчислює цілу частку від ділення кількості рядків на кількість потоків + перші потоки обчислюють ще по одному рядку з залишкових для кращої рівномірності.



```
public static Stopwatch ParallelMatrixMultiplication(int[,] matrixA, int[,] matrixB, int threadCount)
{
  int rowsA = matrixA.GetLength(0);
  int colsA = matrixA.GetLength(1);
  int rowsB = matrixB.GetLength(0);
  int colsB = matrixB.GetLength(1);

  if (colsA != rowsB)
  {
    throw new ArgumentException("Matrices cannot be multiplied: colsA != rowsB.");
  }
  int[,] resultMatrix = new int[rowsA, colsB];

  CountdownEvent countdown = new CountdownEvent(threadCount);
  Stopwatch stopWatch = new Stopwatch();
  stopWatch.Start();

  int baseRowsPerThread = rowsA / threadCount;
  int extraRows = rowsA % threadCount;

  void ComputeMatrixRows(int startRow, int endRow)
  {
    for (int i = startRow; i < endRow; i++)
    {
      for (int j = 0; j < colsB; j++)
      {
        int sum = 0;
        for (int k = 0; k < colsA; k++)
        {
```

```
            sum += matrixA[i, k] * matrixB[k, j];
          }
          resultMatrix[i, j] = sum;
        }
      }
      countdown.Signal();
    }

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

      new Thread(() => ComputeMatrixRows(startRow, endRow)).Start();
    }

    countdown.Wait();

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

(local variable) int extraRows

Крім цих двох методів, реалізована також функція **PrintMatrix**, за допомогої якої я перевірила, чи множення обчислюється 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();
}
```

```
9 67
53 41

20 77
19 97

1453 7192
1839 8058
```

Аналіз результатів

На малих розмірах матриць та великій кількості потоків розпаралелення множення матриць неефективне:

```
nA = 2, mA = 3, nB = 3, mB = 2
Sequential time: 0 ms
Parallel time: 0 ms, threads: 3
Acceleration of parallel: 0.2544145379735985
Efficiency of parallel: 0.08480484599119949
```

```
nA = 100, mA = 100, nB = 100, mB = 100
Sequential time: 5 ms
Parallel time: 8 ms, threads: 101

Acceleration of parallel: 0.7365286408663017
Efficiency of parallel: 0.0072923627808544715
```

На більших розмірностях розпаралелення дає значно кращий результат:

```
nA = 1000, mA = 1000, nB = 1000, mB = 1000
Sequential time: 5313 ms
Parallel time: 1460 ms, threads: 10
Acceleration of parallel: 3.639527121942546
Efficiency of parallel: 0.3639527121942546
```

```
nA = 2000, mA = 2000, nB = 2000, mB = 2000
Sequential time: 42752 ms
Parallel time: 12773 ms, threads: 10

Acceleration of parallel: 3.347093900790494
Efficiency of parallel: 0.3347093900790494
```

При ~10 потоках для ~1-2,000 рядків ефективність є найбільшою, проте з більшим зростанням кількості потоків погіршується:

```
nA = 1500, mA = 1500, nB = 1500, mB = 1500
Sequential time: 18328 ms
Parallel time: 5613 ms, threads: 50

Acceleration of parallel: 3.264795455020333
Efficiency of parallel: 0.065295909100406665
```

Додатково я порівняла те, як на ефективність впливає кратність розмірності матриці до кількості потоків. Бачимо, що не кратні значення погіршують ефективність паралельного алгоритму, але не суттєво, оскільки, за формулою, надлишкові рядки рівномірно розподіляються між першими потоками:

```
nA = 800, mA = 800, nB = 800, mB = 800
Sequential time: 2534 ms
Parallel time: 758 ms, threads: 10

Acceleration of parallel: 3.339697920836989
Efficiency of parallel: 0.3339697920836989
```

```
nA = 800, mA = 800, nB = 800, mB = 800
Sequential time: 2502 ms
Parallel time: 780 ms, threads: 13

Acceleration of parallel: 3.2049684811639993
Efficiency of parallel: 0.246536037012615 34
```

Висновок:

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