МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Паралельні та розподілені обчислення

ЛАБОРАТОРНА РОБОТА №3

«Розв’язування системи лініних алгебрагічних рівнянь»

Виконала:
студентка групи ПМі-31
Дудчак Валентина Юріївна

Львів 2024

Тема: Розпаралелення розв’язування СЛАР

Мета: Написати програми розв’язування СЛАР (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 SequentialGaussElimination.

У ньому матриця спершу зводиться до верхньої трикутної, потім перевіряється, чи на основній діагоналі немає нульових елементів, і потім обчислюються значення змінних у зворотньому обході. Використано також об’єкт класу Stopwatch для засікання часу:

```
1 reference
public static Stopwatch SequentialGaussElimination(double[,] matrix)
{
  int n = matrix.GetLength(0);

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

  try {
    for (int curRow = 0; curRow < n; curRow++)
    {
      for (int i = curRow + 1; i < n; i++)
      {
        double multiplier = matrix[i, curRow] / matrix[curRow, curRow];
        for (int j = curRow; j <= n; j++)
        {
          matrix[i, j] -= multiplier * matrix[curRow, j];
        }
      }
    }

    CheckNonZeroDiagonal(matrix);
    double[] resultArray = new double[n];
```

```
    for (int curRow = n - 1; curRow >= 0; curRow--)
    {
      resultArray[curRow] = matrix[curRow, n];
      for (int j = curRow + 1; j < n; j++)
      {
        resultArray[curRow] -= matrix[curRow, j] * resultArray[j];
      }
      resultArray[curRow] /= matrix[curRow, curRow];
    }
  }
  catch (DivideByZeroException ex)
  {
    throw new InvalidOperationException("Error: Division by zero encountered during Gaussian elimination.", ex);
  }

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

Крім цього, я перевірила, чи результати обчислюються правильно:

```
n = 5
1 7 6 9 6 | 0
1 5 9 8 9 | 5
4 5 8 8 3 | 5
2 2 1 8 2 | 5
1 7 1 0 5 | 8
```

```
3.7920238538948947
-0.7277301528140141
-1.0678345136041751
-0.526090197540067
2.0739843458814766
```

Результати в онлайн-калькуляторі СЛАР:

$$\begin{cases} 1x_1 + 7x_2 + 6x_3 + 9x_4 + 6x_5 = 0 \\ 1x_1 + 5x_2 + 9x_3 + 8x_4 + 9x_5 = 5 \\ 4x_1 + 5x_2 + 8x_3 + 8x_4 + 3x_5 = 5 \\ 2x_1 + 2x_2 + 1x_3 + 8x_4 + 2x_5 = 5 \\ 1x_1 + 7x_2 + 1x_3 + 0x_4 + 5x_5 = 8 \end{cases}$$

$$\begin{cases} x_1 = \frac{10174}{2683} \\ x_2 = -\frac{3905}{5366} \\ x_3 = -\frac{2865}{2683} \\ x_4 = -\frac{2823}{5366} \\ x_5 = \frac{11129}{5366} \end{cases}$$

Паралельний алгоритм

Паралельний алгоритм множення матриць реалізовано у методі ParallelGaussElimination.

Використано об’єкт Thread для розпаралелення процесу обчислення на рядки та об’єкт класу CountdownEvent, щоб упевнитися, що всі потоки завершили свою роботу. При зведенні матриці до трикутної залишкові рядки розподіляються по першим потокам для кращої рівномірності. Потім в одному потоці обчислюється результат.

```
public static Stopwatch ParallelGaussElimination(double[,] matrix, int threadCount)
{
  int n = matrix.GetLength(0);

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

  CountdownEvent countdownEvent = new CountdownEvent(threadCount);
  int baseRowsPerThread = n / threadCount;
  int extraRows = n % threadCount;

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

    Thread thread = new Thread(() =>
    {
      for (int curRow = startRow; curRow < endRow; curRow++)
      {
        for (int j = curRow + 1; j < n; j++)
        {
          double multiplier = matrix[j, curRow] / matrix[curRow, curRow];
          for (int k = curRow; k <= n; k++)
          {
            matrix[j, k] -= multiplier * matrix[curRow, k];
          }
        }
      }
      countdownEvent.Signal();
    });
```

```
    thread.Start();
  }

  countdownEvent.Wait();
  CheckNonZeroDiagonal(matrix);
  double[] resultArray = new double[n];

  for (int curRow = n - 1; curRow >= 0; curRow--)
  {
    resultArray[curRow] = matrix[curRow, n];
    for (int j = curRow + 1; j < n; j++)
    {
      resultArray[curRow] -= matrix[curRow, j] * resultArray[j];
    }
    resultArray[curRow] /= matrix[curRow, curRow];
  }

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

Також обчислюється правильно:

```
n = 5, k = 5
2 4 8 0 8 | 0
5 5 9 0 3 | 9
1 1 1 8 3 | 8
6 8 9 1 2 | 6
0 9 7 7 8 | 5
```

```
-2.1696035242290694
-1.8182819383259925
4.0974669603524205
1.9785242290748892
-2.645925110132157
```

Аналіз результатів

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

```
n = 5
Sequential time: 0 ms
Parallel time: 1 ms, threads: 5

Acceleration of parallel: 0.337932723709993 44
Efficiency of parallel: 0.06758654474199868
```

```
n = 100
Sequential time: 2 ms
Parallel time: 2 ms, threads: 5

Acceleration of parallel: 1.0814986162171094
Efficiency of parallel: 0.2162997232434219
```

На більших розмірностях розпаралелення дає значно кращий результат:

```
n = 1000
Sequential time: 1994 ms
Parallel time: 793 ms, threads: 10

Acceleration of parallel: 2.5122299718250094
Efficiency of parallel: 0.25122299718250096
```

```
n = 2000
Sequential time: 14648 ms
Parallel time: 5750 ms, threads: 10

Acceleration of parallel: 2.5473257439386616
Efficiency of parallel: 0.25473257439386615
```

```
n = 2000
Sequential time: 14716 ms
Parallel time: 4366 ms, threads: 50

Acceleration of parallel: 3.3704754220581896
Efficiency of parallel: 0.06740950844116379
```

При надмірній кількості потоків погіршується:

```
n = 500
Sequential time: 250 ms
Parallel time: 81 ms, threads: 100

Acceleration of parallel: 3.0775801768590942
Efficiency of parallel: 0.030775801768590943
```

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:

```
n = 500
Sequential time: 250 ms
Parallel time: 81 ms, threads: 100

Acceleration of parallel: 3.0775801768590942
Efficiency of parallel: 0.030775801768590943
```

```
n = 500
Sequential time: 251 ms
Parallel time: 84 ms, threads: 111

Acceleration of parallel: 2.9630178562156906
Efficiency of parallel: 0.026693854560501717
```

Висновок:

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