


Лабораторна робота №3. Паралельне представлення алгоритмів

Варіант №8

Мета роботи: Вивчити можливості паралельного обчислення алгоритмів. Набути навичок такого представлення.

Завдання.

8.	1	2	3 ... n-1	
	n			
	2	1	2 ... n-2	
	n-1			
	3	2	1 ... n-3	
	n-2			
	...			
	n-1	n-2	n-3 ... 1	
	2			
	n	n-1	n-2 ... 2	
	1			

Для цього завдання написати програму з одноразовим присвоєння, та рекурсивним обчисленням.

Код програми.

```
using System;
using System.Diagnostics;
using System.Linq;

namespace lab3
{
    class Launcher
    {
        static void Main(string[] args)
        {
            int N = CLI.GetInt("Enter size.", 5);
            Console.WriteLine("Welcome!");
            SquareMatrix aMatrix = new SquareMatrix(N);
            aMatrix.Fill((i, j) => 1 + Math.Abs(i - j));
            Console.WriteLine("Here is A matrix:\n{0}", aMatrix);
            SquareMatrix bMatrix = new SquareMatrix(N);
            bMatrix.Fill((i, j) => (i + j < N && i <= j)?
                CLI.GetDouble(String.Format("Enter [{0},{1}] element", i, j), 1) : 0);

            Console.WriteLine("Here is B matrix:\n{0}", bMatrix);
        }
    }
}
```

```

        // RECURSION PART ...
        Stopwatch recursionStopwatcher = new Stopwatch();
        recursionStopwatcher.Start();
        SquareMatrix recursiveResult =
aMatrix.RecursiveMultiplication(bMatrix);

        recursionStopwatcher.Stop();
        int recursiveCalculations = aMatrix.Calculations;
        long recursionMilliseconds =
recursionStopwatcher.ElapsedMilliseconds;

        long recursionTicks = recursionStopwatcher.ElapsedTicks;
        // SINGLE ASSIGNMENT PART ...
        Stopwatch saStopwatcher = new Stopwatch();
        saStopwatcher.Start();
        SquareMatrix saResult =
aMatrix.SingleAssignmentMultiplication(bMatrix);

        saStopwatcher.Stop();

        int saCalculations = aMatrix.Calculations;
        long saMilliseconds = saStopwatcher.ElapsedMilliseconds;
        long saTicks = saStopwatcher.ElapsedTicks;

        Console.WriteLine("Recursive result:\n{0}",
recursiveResult);
        Console.WriteLine("Amount of calculations took: {0}",
recursiveCalculations);
        Console.WriteLine("Time took to calculate: {0}ms",
recursionMilliseconds);
        Console.WriteLine("Ticks took to calculate: {0}",
recursionTicks);

        Console.WriteLine("Simple multiplication result:\n{0}",
saResult);
        Console.WriteLine("Amount of calculations took: {0}",
saCalculations);
        Console.WriteLine("Time took to calculate: {0}ms",
saMilliseconds);
        Console.WriteLine("Ticks took to calculate: {0}", saTicks);

    }

}

```

```

class SquareMatrix {

    public int Calculations;
    int size;
    public double[,] body = null;
    bool needStringRefresh = true;
    string matrixRepr = null;

    public SquareMatrix(int size) {
        this.size = size;
        body = new double[size,size];
    }
    public void Fill(Func<int, int, double> filler) {
        for(int i = 0; i < size; i++) {
            for(int j = 0; j < size; j++) {
                body[i,j] = filler(i, j);
            }
        }
        needStringRefresh = true;
    }
    public override string ToString() {
        if (needStringRefresh)
            refreshRepr();
        return matrixRepr;
    }
    public SquareMatrix SingleAssignmentMultiplication(SquareMatrix
other) {
        Calculations = 0;
        SquareMatrix result = new SquareMatrix(size);
        for(int i = 0; i < size; i++) {
            for(int j = 0; j < size; j++) {
                for(int k = 0; k < size; k++) {
                    result.body[i, j] += this.body[i, k] *
other.body[k, j];

                    Calculations += 1;
                }
            }
        }
        return result;
    }
    public SquareMatrix RecursiveMultiplication(SquareMatrix other)
{
        Calculations = 0;

```

```

        SquareMatrix result = new SquareMatrix(size);

        Func<int, int, int, double> summator = null;
        summator = (i, j, k) => {
            Calculations += 1;
            return (k == size-1)? (body[i, k] * other.body[k, j] +
result.body[i,j]) : (body[i, k] * other.body[k, j] + summator(i, j,
k+1));
        };

        for(int i = 0; i < size; i++)
            for(int j = 0; j < size; j++)
                result.body[i, j] = summator(i, j, 0);
        return result;
    }

    public double GetMaxElement() {
        Func<int, int, double> maxer = null;
        maxer = (i, j) => {
            return Math.Max(Math.Max(body[i,j], (i < size-1) ?
(maxer(i+1, j)) : (body[i,j])),
                (j < size-1) ? (maxer(i, j+1)) :
(body[i,j]));
        };
        return maxer(0, 0);
    }

    private void refreshRepr() {
        string[] rows = new string[size];
        string[] tempRow = new string[size];
        double maxElement = GetMaxElement();
        string formatting = String.Format("{0,{0}}",
Math.Floor(maxElement).ToString().Length);
        for(int i = 0; i < size; i++) {
            for(int j = 0; j < size; j++) {
                tempRow[j] = String.Format(formatting, body[i, j]);
            }
            rows[i] = String.Join(" ", tempRow);
        }
        matrixRepr = String.Join("\n", rows);
        needStringRefresh = false;
    }
}

public class CLI {

```

```

static void showPrompt(string prompt, string current) {
    Console.WriteLine(prompt);
    Console.Write("[{0}]> ", current);
}

static string[] BOOL_YES = {"yes", "y", "true", "da", "sure"};
static string[] BOOL_NO = {"no", "n", "false", "net", "not
sure"};

public static bool GetBool(string prompt, bool d) {
    showPrompt(prompt, d.ToString());
    string s = Console.ReadLine().Trim().ToLower();
    if (Array.Exists(BOOL_YES, e => s.Equals(e))) {
        return true;
    }
    else if (Array.Exists(BOOL_NO, e => s.Equals(e))) {
        return false;
    }
    return d;
}

public static int GetInt(string prompt, int d) {
    showPrompt(prompt, d.ToString());
    return ParseInt(d);
}

public static double GetDouble(string prompt, double d) {
    showPrompt(prompt, d.ToString());
    return ParseDouble(d);
}

public static int ParseInt(int d) {
    int result = -1;
    bool success = Int32.TryParse(Console.ReadLine(), out
result);
    if (success && result > 0)
        return result;
    return d;
}

public static double ParseDouble(double d) {
    double result = -1;
    bool success = Double.TryParse(Console.ReadLine(), out
result);
    if (success && result > 0)

```

```

        return result;
    return d;
}

}

}

```

Результат виконання.

```

C:\Windows\System32\cmd.exe
Q:\NUWM\shared_calculations\lab3>dotnet run lab3
Enter size.
[5]> 6
Welcome!
Here is A matrix:
1 2 3 4 5 6
2 1 2 3 4 5
3 2 1 2 3 4
4 3 2 1 2 3
5 4 3 2 1 2
6 5 4 3 2 1
Enter [0,0] element
[1]> 6
Enter [0,1] element
[1]> 5
Enter [0,2] element
[1]> 4
Enter [0,3] element
[1]> 7
Enter [0,4] element
[1]> 3
Enter [0,5] element
[1]> 8
Enter [1,1] element
[1]> 34
Enter [1,2] element
[1]> 78
Enter [1,3] element
[1]> 3
Enter [1,4] element
[1]> 6
Enter [2,2] element
[1]> 1
Enter [2,3] element
[1]> 90
Here is B matrix:
6 5 4 7 3 8
0 34 78 3 6 0
0 0 1 90 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
Recursive result:
6 73 163 283 15 8
12 44 88 197 12 16
18 83 169 117 21 24
24 122 252 217 30 32
30 161 335 317 39 40
36 200 418 417 48 48
Amount of calculations took: 216
Time took to calculate: 0ms
Ticks took to calculate: 2834
Simple multiplication result:
6 73 163 283 15 8
12 44 88 197 12 16
18 83 169 117 21 24
24 122 252 217 30 32
30 161 335 317 39 40
36 200 418 417 48 48
Amount of calculations took: 216
Time took to calculate: 0ms
Ticks took to calculate: 1943
Q:\NUWM\shared_calculations\lab3>

```

Висновок.

Як бачимо з результатів виконання моєї програми - рекурсія це більш дорога операція, яка використовує більше ресурсів системи, а також забиває стек викликів. Рекурсія може бути корисною при вирішенні окремої групи задач (наприклад обхід графу в глибину).

Додаткова інформація.

Виконав Лук'яничук Олексій.

Вихідні коди також можна знайти на: https://github.com/alexei-alexov/shared_calculations