

Лабораторна робота №4. Можливості використання паралельних алгоритмів.

Варіант № 8.

Мета роботи: Дослідити можливості розв'язання різноманітних задач за допомогою паралельних алгоритмів. Навчитися виділяти незалежні гілки обчислень та виконувати їх паралельно.

Завдання.

В полі 8*8 кліток зображено кілька прямокутників, кожен з яких складається з кліток, різні прямокутники не перетинаються і не доторкаються один до одного. Задана квадратна матриця порядку 8, в якій елемент рівний нулю, якщо відповідна клітина належить прямокутнику і відмінний від нуля, в іншому випадку. Визначити кількість прямокутників. Початковими даними вважати матрицю елементів, яка повинна вводитися під час виконання програми. Графічно відобразити вхідні дані.

Аналіз задачі

Для простого вирішення цієї проблеми можна використати біжучий елемент який проходить послідовно по всьому полю з верхнього лівого кута до нижнього правого кута в пошуках клітинок які належать прямокутнику. У випадку знаходження клітинки яка належить прямокутнику - це буде верхній лівий кут прямокутника, для знаходження границь прямокутника використаємо бігунець який буде спускатись з початкової знайденої точки - в нижній правий кут, доти, доки не буде знайдено нижній правий кут прямокутника. Знайшовши границі прямокутника - змалюймо його, для того щоб не порахувати його більше одного разу, та збільшуємо лічильник прямокутників на 1.

Для модифікації нашого алгоритму введемо додаткове поняття **вертикальний бігунець** - це бігунець який рухається в одній вертикальній в пошуках прямокутників на цій вертикалі. Зроблено це для того щоб вертикальний бігунець міг розділяти поле на два незалежних один від одного поля прибираючи при цьому всі прямокутники які знаходяться на межі. Для коректної роботи цього бігунця ми повинні модифікувати алгоритм який знаходить межі прямокутника, для того щоб він знаходив окрім нижнього правого кута ще й верхній лівий. Отже, суть паралельності алгоритму полягає у тому щоб використовувати вертикальний бігунець для поділу поля на незалежні частини, та горизонтальний бігунців, для перевірки цих незалежних частин поля одночасно.

Текст програми

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

namespace lab4
```

```

{

    public class Seeker
    {

        static string welcome_str = @"
Welcome to rectangle seeker!

";

        static string type_ask = @"
Please, choose input type:
1 - file
2 - stdin

";

        static string split_ask = @"
Please, enter amount of field splits or choose recommended.

";

        static string subseeker_ask = @"
Would you like to use additional threads for rectangle corners seek?

";

        static string final_out = @"
Result: {0} rectangle(s).
Time took: {1}ms.
Ticks took: {2}.
Threads created: {3}.

";

        static string size_ask = @"
Please, enter size of matrix.

";

        static string matrix_ask = @"Enter matrix.";

        static string filename_ask = @"Enter filename with matrix.";

        static readonly int TYPE_FILE = 1;
        static readonly int TYPE_STDIN = 2;

        static readonly int RECT = 0;

```

```

static readonly int REC_SPLIT = 8;

int[,] field;
int[,] _field;
int size;

bool sub_threads;

public int threads_created;

int rect_amount;

List<Thread> thread_pool;

static void Main(string[] args)
{

    int[,] field;
    int size;
    Seeker sk;

    if (args.GetLength(0) > 1 && args[1] == "testing") {
        bool verbose = false;
        // bool verbose = (args[3] == "+");
        int test_amount = Int32.Parse(args[2]);
        Console.WriteLine("Testing {0} amount ...\nGetting file
from testing folder.", test_amount);
        string[] files_to_test = Directory.GetFiles(@"tests");
        Stopwatch s = new Stopwatch();

        long min_ticks;
        long max_ticks;
        long min_ms;
        long max_ms;
        long temp_ticks;
        long temp_ms;
        long total_ticks;
        long total_ms;
        long temp_avg_ticks;
        long temp_avg_ms;

        long best_ticks;
        long best_ms;
        long best_ticks_split;

```

```

        long best_ms_split;

        foreach(string filename in files_to_test) {
            field = Seeker.ReadFile(filename);
            size = field.GetLength(0) / 2;
            sk = new Seeker(field);
            best_ticks = Int64.MaxValue;
            best_ms = Int64.MaxValue;
            best_ticks_split = 0;
            best_ms_split = 0;

            Console.WriteLine("Filename: {0} Size: {1}",
filename, size);

            for(int test_split = 0; test_split < size;
test_split++) {

                min_ticks = Int64.MaxValue;
                max_ticks = 0;
                min_ms = Int64.MaxValue;
                max_ms = 0;
                total_ticks = 0;
                total_ms = 0;

                for(int i = 0; i < test_amount; i++) {
                    s.Start();
                    sk.seek(test_split);
                    s.Stop();
                    temp_ticks = s.ElapsedTicks;
                    temp_ms = s.ElapsedMilliseconds;
                    s.Reset();

                    if (temp_ticks < min_ticks)min_ticks =
temp_ticks;

                    if (temp_ticks > max_ticks)max_ticks =
temp_ticks;

                    if (temp_ms < min_ms)min_ms = temp_ms;
                    if (temp_ms > max_ms)max_ms = temp_ms;

                    total_ticks += temp_ticks;total_ms +=
temp_ms;

                }

```

```

        temp_avg_ticks = total_ticks / test_amount;
        temp_avg_ms = total_ms / test_amount;

        if (best_ms > temp_avg_ms) {
            best_ms = temp_avg_ms;
            best_ms_split = test_split;
        }
        if (best_ticks > temp_avg_ticks) {
            best_ticks = temp_avg_ticks;
            best_ticks_split = test_split;
        }
        if (verbose)
            Console.WriteLine(@"
=====
Splits: {0}
== Ticks ==
Max: {2}
Avg: {3}
Min: {1}
== Ms ==
Max: {5}
Avg: {6}
Min: {4}
", test_split, min_ticks, max_ticks, temp_avg_ticks, min_ms, max_ms,
temp_avg_ms);

        }
        Console.WriteLine(@"
=====
Best split according to avg. ticks: {0}
Avg. ticks made: {1}
=====
Best split according to avg. ms: {2}
Avg ms took: {3} ms.
=====
", best_ticks_split, best_ticks, best_ms_split, best_ms);
    }
    return;
}

Console.WriteLine(Seeker.welcome_str);

int input_type = CLI.GetInt(Seeker.type_ask, 1);

if (input_type == TYPE_STDIN) {

```

```

        size = CLI.GetInt(size_ask, 8);
        field = CLI.GetMatrix(matrix_ask, 8);
    }
    else if (input_type == TYPE_FILE) {
        while(true) {
            try {
                Console.WriteLine(filename_ask);
                string filename = Console.ReadLine();
                field = Seeker.ReadFile(filename);
                size = field.GetLength(0);
                break;
            }
            catch(Exception e) {
                Console.WriteLine("Check file and try again.
{0}", e);
            }
        }
    }
    else
        return;

    Console.WriteLine(field);
    sk = new Seeker(field);
    sk.PrintField();

    int splits = CLI.GetInt(Seeker.split_ask, size / REC_SPLIT);
    bool sub_threads = CLI.GetBool(Seeker.subseeker_ask, false);
    Stopwatch watcher = new Stopwatch();
    watcher.Start();
    int squares = sk.seek(splits, sub_threads);
    watcher.Stop();

    Console.WriteLine(final_out, squares,
watcher.ElapsedMilliseconds, watcher.ElapsedTicks, sk.threads_created);

}

public Seeker(int[,] field) {
    this._field = field;
    this.size = field.GetLength(0);
    thread_pool = new List<Thread>();
    Reset();
}

```

```

private void Reset() {
    rect_amount = 0;
    field = _field.Clone() as int[,];
    thread_pool.Clear();
    threads_created = 0;
}

public int seek(int splits=1, bool sub_threads = false) {
    Reset();
    this.sub_threads = sub_threads;

    if (splits > 0)
        VSeeker(0, size, splits);
    else
        HSeeker(0, size);

    // wait for all thread to finish the job.
    while(true) {
        try {
            foreach (Thread thr in thread_pool) thr.Join();
            break;
        }
        catch(Exception) {}
    }
    return rect_amount;
}

private void VSeeker(int x0, int x1, int splits=1) {
    int width = x1 - x0;
    int middle = x0 + width / 2;
    bool double_w_seek = (width & 1) != 1;
    if (double_w_seek) middle -= 1;

    for(int y = 0; y < size; y++) {
        if (field[middle, y] == RECT) {
            Interlocked.Increment(ref rect_amount);
            FillField(FindCorners(middle, y));
        }
        if (double_w_seek) {
            if (field[middle+1, y] == RECT) {
                Interlocked.Increment(ref rect_amount);
                FillField(FindCorners(middle+1, y));
            }
        }
    }
}

```

```

    }
}

int left_width = middle - x0;
int right_width = x1 - middle - 1;
if (double_w_seek) right_width -= 1;

if ((left_width > 0 && splits < 2) || left_width == 1) {
    Thread h_left_thr = new Thread(() => HSeeker(x0, x0 +
left_width));
    Interlocked.Increment(ref threads_created);
    thread_pool.Add(h_left_thr);
    h_left_thr.Start();
}

if ((right_width > 0 && splits < 2) || right_width == 1) {
    Thread h_right_thr = new Thread(() => HSeeker(x1 -
right_width, x1));
    Interlocked.Increment(ref threads_created);
    thread_pool.Add(h_right_thr);
    h_right_thr.Start();
}

if (splits > 1 && left_width > 1) {
    Thread v_left_thread = new Thread(() => VSeeker(x0, x0 +
left_width, splits-1));
    Interlocked.Increment(ref threads_created);
    thread_pool.Add(v_left_thread);
    v_left_thread.Start();
}

if (splits > 1 && right_width > 1) {
    Thread v_right_thread = new Thread(() => VSeeker(x1 -
right_width, x1, splits-1));
    Interlocked.Increment(ref threads_created);
    thread_pool.Add(v_right_thread);
    v_right_thread.Start();
}

}

private void HSeeker(int x0, int x1) {
    for(int y = 0; y < size; y++) {
        for(int x = x0; x < x1; x++) {
            if (field[x, y] == RECT) {

```



```

        Interlocked.Increment(ref rect_amount);
        FillField(FindCorners(x, y));
    }
}

}

}

private void FillField(int[] corners) {
    for(int x = corners[0]; x <= corners[2]; x++)
        for(int y = corners[1]; y <= corners[3]; y++)
            field[x, y] += 1;
}

/// Find Top-Left and Bottom-Right corners of rectangle.
/// Can use threads (false by default)
/// Result: [top-left-x, top-left-y, bottom-right-x,
bottom-right-y]
private int[] FindCorners(int x, int y) {
    int[] result = new int[4];
    result[0] = result[2] = x;
    result[1] = result[3] = y;

    if (field[x, y] != RECT) return result;

    Action top_left_seeker = () => {
        while(result[0] > 0 && field[result[0]-1, result[1]] ==
RECT) result[0]--;
        while(result[1] > 0 && field[result[0], result[1]-1] ==
RECT) result[1]--;
    };

    Action bottom_right_seeker = () => {
        while(result[2] < size-1 && field[result[2]+1,
result[3]] == RECT) result[2]++;
        while(result[3] < size-1 && field[result[2],
result[3]+1] == RECT) result[3]++;
    };

    if (sub_threads) {
        Thread tl_thr = new Thread(() => top_left_seeker());
        Thread br_thr = new Thread(() => bottom_right_seeker());
        Interlocked.Increment(ref
threads_created); Interlocked.Increment(ref threads_created);
        tl_thr.Start();
        br_thr.Start();
    }
}

```

```

        tl_thr.Join();
        br_thr.Join();
    }
    else {
        top_left_seeker();
        bottom_right_seeker();
    }
    return result;
}

public void PrintField() {
    string[] rows = new string[size];
    string[] tempRow = new string[size];
    string formatting = "{0,1}";

    string wall = "■"; string rect = " ";
    for(int i = 0; i < size; i++) {
        for(int j = 0; j < size; j++) {
            tempRow[j] = String.Format(formatting, field[j, i]
== RECT ? rect : (field[j, i] > 1? "#" : wall));
        }
        rows[i] = String.Join("", tempRow);
    }
    Console.WriteLine(String.Join("\n", rows));
}

private static int[,] ReadFile(string filename) {
    StreamReader file = new StreamReader(filename);

    // read first line to get size...
    string line = file.ReadLine();
    string[] splitted = line.Split(" ");

    int size = splitted.GetLength(0);
    int[,] field = new int[size, size];

    for(int j = 0; j < size; j++)
        field[0,j] = Int32.Parse(splitted[j]);

    for(int i = 1; i < size; i++) {
        splitted = file.ReadLine().Split(" ");
        for(int j = 0; j < size; j++)
            field[i,j] = Int32.Parse(splitted[j]);
    }
}

```

```

        return field;
    }
}

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)
            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;
    }

    public static int[,] GetMatrix(string prompt, int size) {
        int d = 1;
        int[,] result = new int[size, size];
        Console.WriteLine(prompt);

        for(int i = 0; i < size; i++) {
            for(int j = 0; j < size; j++) {
                PrintArray(result);
                showPrompt("Enter element.", String.Format("{0},
{1}", (j+1), (i+1)));
                result[i,j] = ParseInt(d);
            }
        }
        return result;
    }

    public static void PrintArray(int[,] array) {
        for(int i = 0; i < array.GetLength(0); i++) {
            for(int j = 0; j < array.GetLength(1); j++) {
                Console.Write(array[i,j]);
            }
            Console.WriteLine();
        }
    }

}
}

```


Результати роботи програми

```

C:\Windows\System32\cmd.exe
Q:\NUWM\shared_calculations\lab4>dotnet run lab4

Welcome to rectangle seeker!

Please, choose input type:
1 - file
2 - stdin

[1]>
Enter filename with matrix.
8
System.Int32[,]


Please, enter amount of field splits or choose recommended.

[1]>

Would you like to use additional threads for rectangle corners seek?

[False]>

Result: 32 rectangle(s).
Time took: 3ms.
Ticks took: 16591.
Threads created: 2.

Q:\NUWM\shared_calculations\lab4>_

```

Висновки

Як було мною написано в попередніх звітах, створення нового потоку - операція дорога, тому, нажаль, результати використання паралельних обчислень - це дали очікуваного приросту швидкодії, навіть навпаки, занадто велика кількість потоків сповільнювала роботу програми. Отже, перейдемо до конкретних значень. Для підбору статистики я написав підтримку тестів, які перевіряються програмою, та виводять основні статистичні дані. Результати тестів наведено в таблиці нижче.

Розмір поля	Кількість поділів	Максимальний час (мс)	Середній час (мс)	Мінімальний час (мс)
8x8	0	0,0117	0,01074	0,0095
	1	0,3439	0,257272	0,2214
	2	2,9154	0,682776	0,5337
16x16	0	0,1841	0,037216	0,0254
	1	0,664	0,319332	0,2384
	2	0,9823	0,616936	0,5487
	4	1,9455	1,237552	1,0572
32x32	0	0,3532	0,122788	0,0997
	1	0,7234	0,3435	0,2881
	2	2,5592	0,747708	0,5359
	4	7,2022	3,015624	2,2387
64x64	0	1,2046	0,459224	0,3954
	1	0,8471	0,623148	0,494
	2	1,2394	0,838628	0,6673
	4	4,9368	3,124796	2,2973
	8	9,4246	6,494816	5,1389
128x128	0	4,3472	1,8888	1,6687
	1	3,3469	1,86686	1,4417
	2	2,6521	2,201008	1,9723
	4	4,0305	3,33158	2,6984
	8	16,5638	13,939968	12,1884
	16	16,8491	14,675584	12,5419