Національний Технічний Університет України
«Київський Політехнічний Інститут імені Ігоря Сікорського»
Інститут Прикладного Системного Аналізу
Кафедра Системного Проектування

Паралельні обчислення

Лабораторна робота №1

Роботу виконала:
Желєзнова В.С.
Група: ДА-81
Перевірив: Яременко В. С.

Київ – 2021

ЗМІСТ

Мета роботи

Розробка та реалізація паралельного алгоритму для задач із паралелізмом даних.

Завдання

1. Розробити послідовну та багатопоточну програми, які реалізують варіант індивідуального завдання (мова програмування обирається студентом).
2. Порівняти правильність виконання, порівнявши послідовний та паралельний розв’язки.
3. Виміряти час розрахунку для послідовного та паралельного розв’язків при різних значеннях SIZE та NUMBER_THREADS для власного варіанту, знайти значення, при яких кожен із розв’язків (послідовний чи паралельний) буде виконуватися швидше за інший, зробити таблицю та графічно представити результат.

Варіант 10.

Створити двовимірний масив розмірності n×m. Знайти максимальне та мінімальне значення масиву. (n, m ≥ 100).

1. Лістинг програми мовою Java

```
package com.company;

import java.util.Arrays;

public class ThreadWork {
    public static final int ROWS = 100;
    public static final int COLS = 100;
    public static final int NUMBER_THREADS = 4;
    public static final int RAND_MIN = 1;
    public static final int RAND_MAX = 250000;
    public static final int TESTS = 10;
    public static final int TABLE_HEADER = 2;
    public static final int TABLE_FOOTER = 2;

    public static void main(String[] args) throws InterruptedException {

        final Object[][] resultsTable = new String[TABLE_HEADER + TESTS +
TABLE_FOOTER][];
        resultsTable[0] = new String[]{"", "Serial", "",  "", "Parallel",
""};
        resultsTable[1] = new String[]{"min", "max", "time (us)", "min",
"max", "time (us)"};
        long[] serialTimes = new long[TESTS];
        long[] parallelTimes = new long[TESTS];

        for (int test_i = 0; test_i < TESTS; test_i++) {
            int[][] array = new int[ROWS][COLS];
            for (int i = 0; i < ROWS; i++) {
                for (int j = 0; j < COLS; j++) {
                    array[i][j] = RAND_MIN + (int) (Math.random() *
RAND_MAX);
                }
            }
            long start = System.nanoTime();

            int serialMax = RAND_MIN - 1;
            int serialMin = RAND_MAX + 1;
            int localMax;
            int localMin;
            for (int i = 0; i < ROWS; i++) {
                localMax = Arrays.stream(array[i]).max().orElseThrow();
                localMin = Arrays.stream(array[i]).min().orElseThrow();
                if (localMax > serialMax) {
                    serialMax = localMax;
                }
                if (localMin < serialMin) {
                    serialMin = localMin;
                }
            }
            long end = System.nanoTime();
            long serialTime = (end - start) / 1000;
            serialTimes[test_i] = serialTime;


            start = System.nanoTime();
            ThreadCalc[] ThreadArray = new ThreadCalc[NUMBER_THREADS];
```

```
            for (int i = 0; i < NUMBER_THREADS; i++) {
                ThreadArray[i] = new ThreadCalc(array,
                        ROWS / NUMBER_THREADS * i,
                        i == (NUMBER_THREADS - 1) ? ROWS : ROWS /
NUMBER_THREADS * (i + 1));
                ThreadArray[i].start();
            }
            for (int i = 0; i < NUMBER_THREADS; i++) {
                ThreadArray[i].join();
            }

            int parallelMax = RAND_MIN - 1;
            int parallelMin = RAND_MAX + 1;
            for (int i = 0; i < NUMBER_THREADS; i++) {
                localMax = ThreadArray[i].getMax();
                localMin = ThreadArray[i].getMin();
                if (localMax > parallelMax) {
                    parallelMax = localMax;
                }
                if (localMin < parallelMin) {
                    parallelMin = localMin;
                }
            }

            end = System.nanoTime();

            long parallelTime = (end - start) / 1000;
            parallelTimes[test_i] = parallelTime;

            resultsTable[TABLE_HEADER + test_i] = new
String[]{String.valueOf(serialMin), String.valueOf(serialMax),
String.valueOf(serialTime), String.valueOf(parallelMin),
String.valueOf(parallelMax), String.valueOf(parallelTime)};
        }

        resultsTable[TABLE_HEADER + TESTS] = new String[]{"", "", "", "", "",
""};
        resultsTable[TABLE_HEADER + TESTS + 1] = new String[]{"", "Average:
", String.valueOf(Arrays.stream(serialTimes).average().orElse(-1)), "",
"Average:", String.valueOf(Arrays.stream(parallelTimes).average().orElse(-
1))};
        System.out.println("Rows = " + String.valueOf(ROWS));
        System.out.println("Cols = " + String.valueOf(COLS));
        System.out.println("Number of threads  = " +
String.valueOf(NUMBER_THREADS));
        System.out.println("");

        for (final Object[] row: resultsTable) {
            System.out.format("%10s%10s%10s%10s%10s%10s%n", row);

        }
    }
}
```

Лістинг 1. ThreadSample.java

```
import java.util.Arrays;
```

```
class ThreadCalc extends Thread{

    int[][] array;
    int startIndex;
    int endIndex;
    int max = ThreadWork.RAND_MIN - 1;
    int min = ThreadWork.RAND_MAX + 1;

    public ThreadCalc(int[][] array, int startIndex, int endIndex) {
        this.array = array;
        this.startIndex = startIndex;
        this.endIndex = endIndex;
    }


    public int getMax() {
        return max;
    }
    public int getMin() { return min; }

    @Override
    public void run(){
        int localMax;
        int localMin;
        for(int i = startIndex; i<endIndex; i++ ){
            localMax= Arrays.stream(array[i]).max().orElseThrow();
            localMin= Arrays.stream(array[i]).min().orElseThrow();
            if (localMax > max){
                max = localMax;
            }
            if (localMin < min) {
                min = localMin;
            }
        }
    }
}
```

Лістинг 2. ThreadCalc.java

2. Порівняння реалізацій

При значеннях:

ROWS = 100; COLS = 100; NUMBER_THREADS = 4;

```
Rows = 100
Cols = 100
Number of threads  = 4

             Serial                        Parallel
       min       max time (us)       min       max time (us)
         2    249973      8473         2    249973     16097
        22    249998      4365        22    249998     20403
        34    249943      2218        34    249943     13652
         7    249990      1736         7    249990      3529
        14    249989       161        14    249989      1662
        10    249948       552        10    249948      5765
        79    249999       193        79    249999     17152
         6    249938       127         6    249938      2302
        14    249998       143        14    249998      2857
        27    249996       171        27    249996     12430

          Average:     1813.9           Average:    9584.9
```

Рисунок 2.1 Таблиця з порівнянням для матриці 100×100 та 4-ма потоками

При значеннях:

ROWS = 100; COLS = 100; NUMBER_THREADS = 2;

```
Rows = 100
Cols = 100
Number of threads  = 2

             Serial                        Parallel
       min        max time (us)       min        max time (us)
        38     249999       7659         38     249999       9332
         3     249983        689          3     249983      18549
         4     249984       1168          4     249984      16922
         9     249981        214          9     249981       4212
        22     249943        163         22     249943      10140
        33     249991        161         33     249991       6540
        27     249984        181         27     249984       6266
        37     249954        150         37     249954        893
         6     249996        148          6     249996        948
        20     249988        149         20     249988       2413

           Average:     1068.2            Average:    7621.5
```

Рисунок 2.2 Таблиця з порівнянням для матриці 100×100 та 2-ма потоками

При значеннях:

ROWS = 1000; COLS = 1200; NUMBER_THREADS = 4;

```
Rows = 1000
Cols = 1200
Number of threads  = 4

             Serial                        Parallel
       min       max time (us)       min       max time (us)
         5   2500000     53196         5   2500000      3838
         2   2499998      9226         2   2499998      9501
         1   2499999      8050         1   2499999      6066
         5   2500000      2519         5   2500000     16736
         2   2499997      2391         2   2499997      2973
         1   2499999      2019         1   2499999      3082
         2   2499996      9821         2   2499996      3634
         3   2500000      2621         3   2500000      3404
         3   2499998      2198         3   2499998      1932
         2   2499998      2110         2   2499998      2469

           Average:     9415.1           Average:    5363.5
```

Рисунок 2.3 Таблиця з порівнянням для матриці 1000×1200 та 4-ма потоками

При значеннях:

ROWS = 1000; COLS = 1200; NUMBER_THREADS = 2;

```
Rows = 1000
Cols = 1200
Number of threads  = 2

             Serial                      Parallel
       min       max time (us)       min       max time (us)
         1   2499996     55953         1   2499996      3992
         1   2499999      7898         1   2499999      4809
         3   2499999      3930         3   2499999      2578
         1   2499998      3202         1   2499998      3658
         4   2500000      4133         4   2500000     32938
         2   2499998      8322         2   2499998      2545
         3   2499998      2643         3   2499998      3225
         4   2500000      2924         4   2500000     22100
        13   2499999      3539        13   2499999      4413
         2   2500000      3083         2   2500000      4463

           Average:     9562.7           Average:    8472.1
```

Рисунок 2.4 Таблиця з порівнянням для матриці 1000×1200 та 2-ма потоками

3. Порівняння часу виконання

Таблиця 3.1 Порівняння часу виконання алгоритму

ROWS	COLS	Serial	2 Threads	4 Threads
100	120	1234.1	5239.9	4651.7
1000	1200	9562.7	8472.1	5363.5
2500	3000	20388.3	11229.2	10398.0
5000	6000	54398.3	29600.9	19759.3
10000	12000	183199.9	89374.0	54575.9

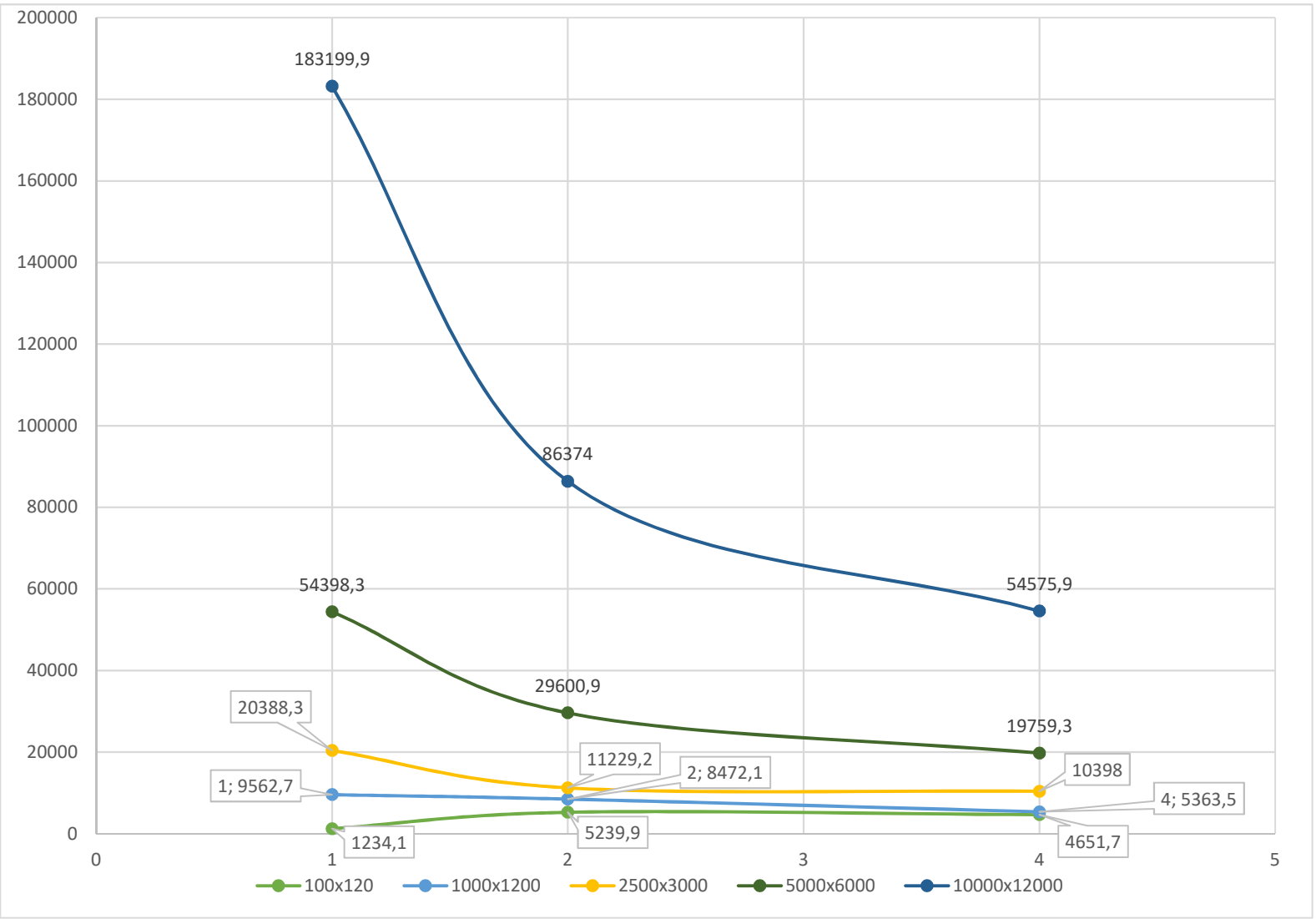

Рисунок 3.1 Графічне зображення порівняння часу виконання алгоритму

Висновки

У ході виконання даної лабораторної роботи я розробила програму, що реалізує варіант мого завдання як послідовно, так і паралельно, а саме пошук максимального та мінімального числа у масиві розмірністю n×m (n, m ≥ 100). Також було проведено порівняння результатів аби впевнитися, що результати, отримані при розв’язанні послідовно та паралельно однакові.

У результаті аналізу виконаної роботи було отримано таблицю залежності часу виконання від об’єму вхідних даних та кількості потоків, якими було розв’язано завдання. Також цю залежність було зображено графічно. Підсумовуючи проведений аналіз можу сказати, що використання декількох потоків при малих об’ємах даних недоцільно, оскільки прискорення виконання завдання спостерігається починаючи з масиву 1000×1200 елементів цілочисельного типу, а саме 4,5МБайтів даних.

Також, у зв’язку з моєю неуважністю до підрахунку об’єму даних мені довелося стикнутися з помилкою пам’яті, адже я намагалася використати масив 100000×120000, що еквівалентно 45ГБайтам пам’яті.