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

Паралельні обчислення

Лабораторна робота №1

Роботу виконав:
Дєрюгін Є. О.
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 7.

Створити вектор з N>=10000 елементами з випадкових чисел. Знайти норму вектора.

1. Лістинг програми мовою Java

```
import java.util.Arrays;

public class ThreadSample {
    public static int SIZE = 6000;
    public static int NUNMBER_THREADS = 8;
    public static int TABLE_HEADER = 2;
    public static int TABLE_FOOTER = 2;
    public static int TESTS = 10;
    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[]{"Norm","Time(us)","Norm","Time(us)"};
        long[] parallelTimes = new long[TESTS];
        long[] serialTimes = new long[TESTS];

        for(int test_i=0;test_i<TESTS;test_i++) {


            double vectA[] = new double[SIZE];
            int rand_min = 1;
            int rand_max = 42;
            for (int i = 0; i < SIZE; i++) { //початкове заповнення векторів
випадковими величинами з зазначеного проміжку
                vectA[i] = rand_min + (int) (Math.random() * rand_max);
            }

            long start = System.nanoTime();

            double serialResult = 0;
            for (int i = 0; i < SIZE; i++) {
                serialResult += Math.abs(vectA[i]);
            }

            long end = System.nanoTime();
            long serialTime = (end - start) / 1000;
            serialTimes[test_i] = serialTime;

            ThreadCacl TreadArray[] = new ThreadCacl[NUNMBER_THREADS];

            start = System.nanoTime();


            for (int i = 0; i < NUNMBER_THREADS; i++) { //розбиття на потоки
                TreadArray[i] = new ThreadCacl(vectA,
                        SIZE / NUNMBER_THREADS * i,
                        i == (NUNMBER_THREADS - 1) ? SIZE : SIZE /
NUNMBER_THREADS * (i + 1)); //тернарна умовна операція
                TreadArray[i].start();
            }
            for (int i = 0; i < NUNMBER_THREADS; i++) { //очікування
завершення усіх потоків
                TreadArray[i].join();
```

```
            }
            double parallelResult = 0;
            for (int i = 0; i < NUNMBER_THREADS; i++) { //збір результатів
паралельної роботи
                parallelResult += TreadArray[i].getResult();
            }

            end = System.nanoTime();

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

            resultsTable[TABLE_HEADER + test_i] = new
String[]{String.valueOf(serialResult),String.valueOf(serialTime),String.value
Of(parallelResult),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("Size =" + String.valueOf(SIZE));
        System.out.println("Number of threads = " +
String.valueOf(NUNMBER_THREADS));
        System.out.println("");
        for(final Object[] row : resultsTable){
            System.out.format("%10s%10s%10s%10s%n",row);
        }
    }
}
```

Лістинг 1. ThreadSample.java

```
class ThreadCacl extends Thread {

    double vectA[];
    int startIndex;
    int endIndex;
    double result;

    public ThreadCacl(double[] vectA, int startIndex, int endIndex) {
//конструктор класу, приймає дані для обчислень
        this.vectA = vectA;
        this.startIndex = startIndex;
        this.endIndex = endIndex;
    }

    public double getResult() {
        return result;
    }
```

```
    @Override
    public void run() { //обрахунки, що здійснюватимуться в зазначеному
потоці
        for (int i = startIndex; i < endIndex; i++) {
            result += Math.abs(vectA[i]);
        }
    }
}
```

Лістинг 2. ThreadCalc.java

2. Порівняння реалізацій

При значеннях:

SIZE = 10000, NUMBER_THREADS = 2

Size =10000
Number of threads = 2

Serial		Parallel	
Norm	Time(us)	Norm	Time(us)
215115.0	362	215115.0	3245
212371.0	611	212371.0	1325
216468.0	489	216468.0	2008
215091.0	406	215091.0	2958
216073.0	58	216073.0	4257
214886.0	113	214886.0	1545
215479.0	54	215479.0	674
214756.0	48	214756.0	870
215796.0	44	215796.0	827
216217.0	48	216217.0	599
Average:	223.3	Average:	1830.8

Рисунок 2.1 Таблиця з порівнянням для вектору 10000 та 2-ма потоками

При значеннях:

SIZE = 100000, NUMBER_THREADS = 4

```
Size =100000
Number of threads = 4

    Serial            Parallel
      Norm  Time(us)      Norm  Time(us)
 2150163.0       547 2150163.0      6731
 2146480.0       506 2146480.0      7597
 2153857.0       979 2153857.0      3627
 2150569.0       513 2150569.0      9845
 2153342.0      1004 2153342.0      3689
 2149296.0       453 2149296.0      6049
 2153552.0       457 2153552.0      5651
 2147003.0       511 2147003.0      2479
 2152518.0      5203 2152518.0      4665
 2147798.0       545 2147798.0      4295

 Average:     1071.8 Average:     5462.8
```

Рисунок 2.2 Таблиця з порівнянням для вектору 10000 та 4-ма потоками

3. Порівняння часу виконання

Таблиця 3.1 Порівняння часу виконання алгоритму

SIZE	Serial	2 Threads	4 Threads
10000	264.4	2070.0	1789.7
15000	233.2	7506.4	3672.8
20000	473.6	3118.1	3087.2
50000	1011.6	4777.2	4593.6
100000	1852.0	3814.0	3301.5

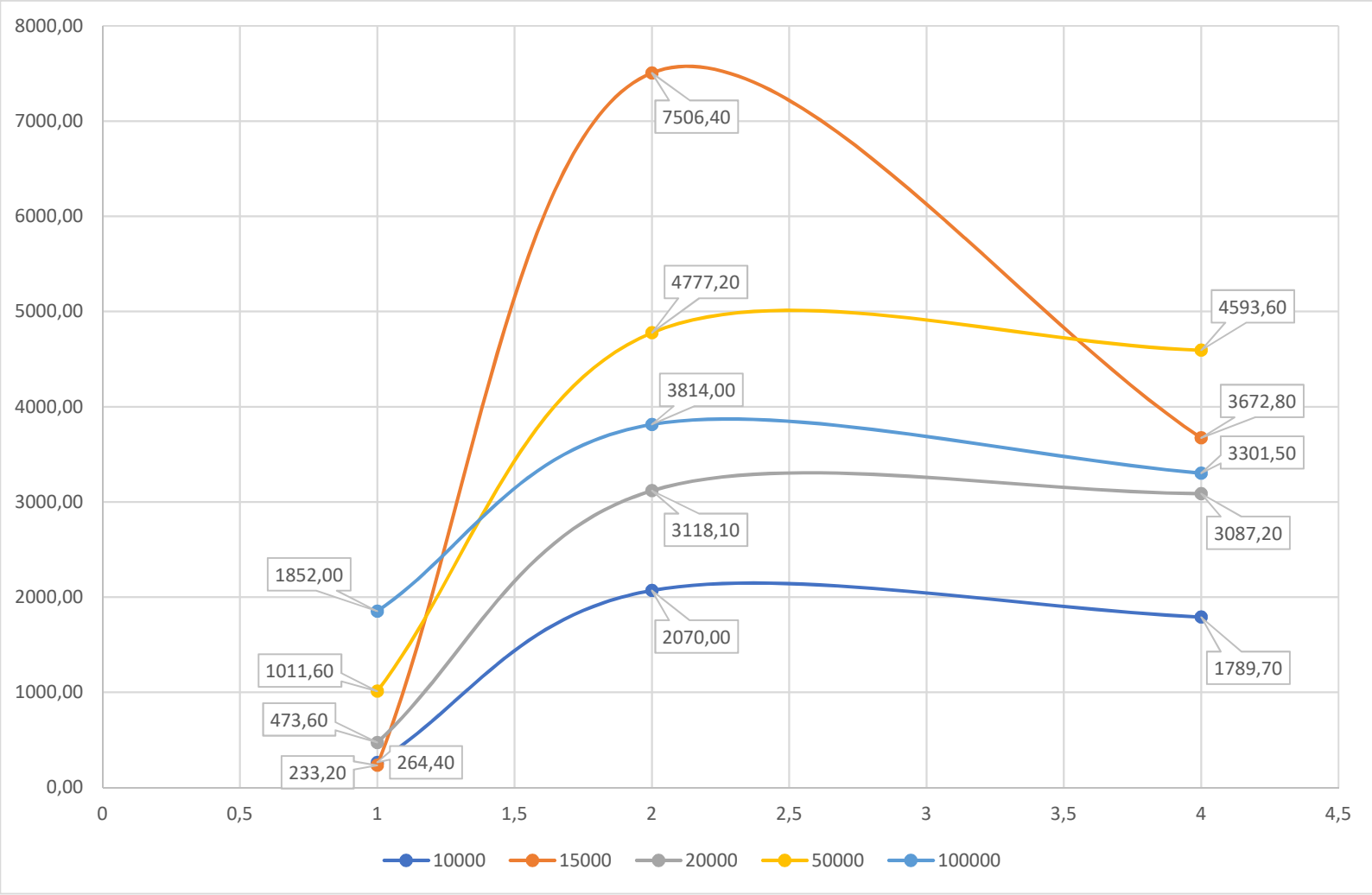

Рисунок 3.1 Графічне зображення порівняння часу виконання алгоритму

Висновки

У ході виконання даної лабораторної роботи я розробив програму, що реалізує варіант мого завдання як послідовно, так і паралельно, а саме розрахунок норми вектора розмірністю N>=10000. 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.