Maeva Lecavelier (마에바 르꺄블리에) – 50191580

Multicore Computing – proj1: problem1

1. Goal of the project _____ p.1
2. Environment under which the experimentations were made _____ p.1
3. Graphs and tables _____ p.2
4. Interpretation and explanation _____ p.3
5. Output screenshots _____ p.4
6. Source code _____ p.5

1. Goal:

Implement multithreaded version of pc_serial.java using static load balancing and dynamic load balancing and submit the modified JAVA codes ("pc_static.java" and "pc_dynamic.java"). Your codes also should print the (1) execution time of each thread and (2) execution time for the entire thread computation.

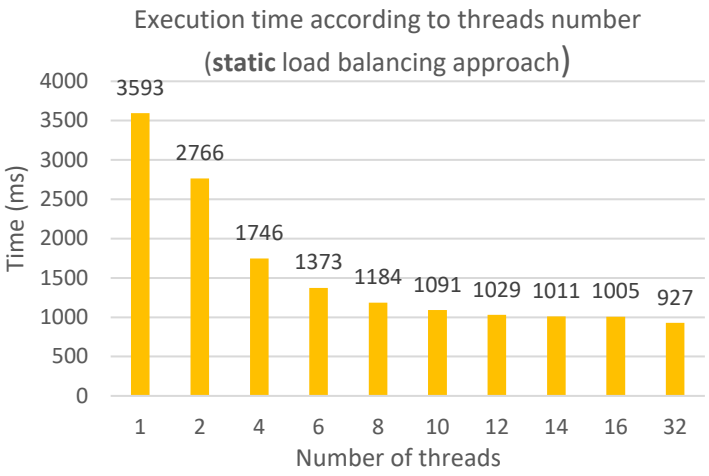
2. Environment:

- Windows 10 (64 bits)
- CPU i7 quad-core
- 8 Go RAM
- HyperThreading ON
- Base speed: 1.99 GHz (there was a peak at 3 GHz when computing)

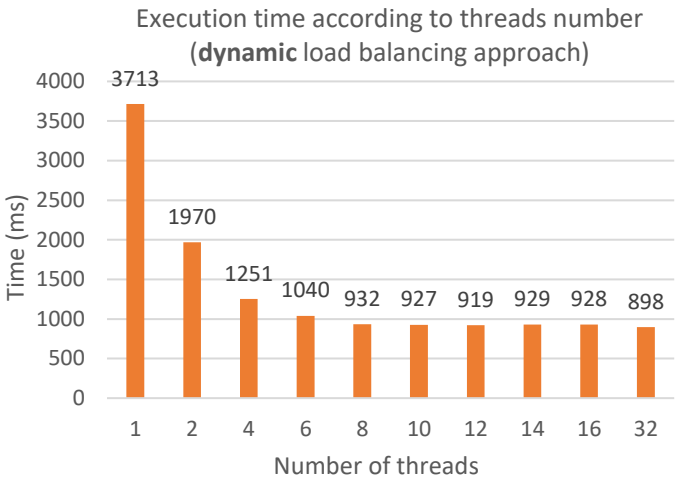
3. Execution time and performance graphs results

Table 1: Execution time according to thread number for a static and a dynamic load balancing approach.

Execution time (ms)	1	2	4	6	8	10	12	14	16	32
Static	3593	2766	1746	1373	1184	1091	1029	1011	1005	927
Dynamic	3713	1970	1251	1040	932	927	919	929	928	898



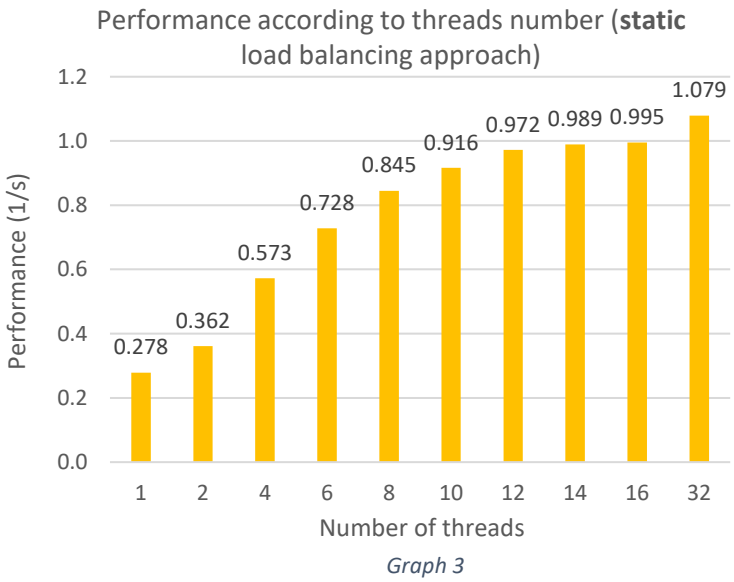
Graph 1



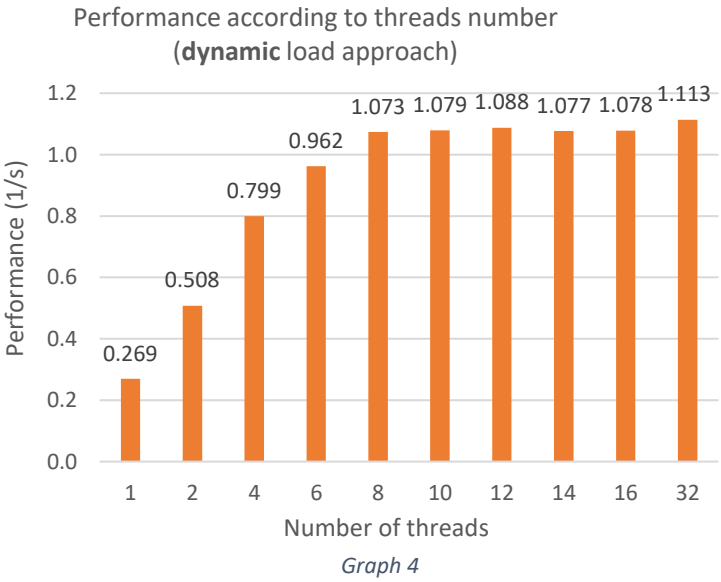
Graph 2

Table 2: performance according to thread number for a static and a dynamic load balancing approach.

Performance (1/s)	1	2	4	6	8	10	12	14	16	32
Static	0.278	0.362	0.573	0.728	0.845	0.916	0.972	0.989	0.995	1.079
Dynamic	0.269	0.508	0.799	0.962	1.073	1.079	1.088	1.077	1.078	1.113



Graph 3



Graph 4

4. Interpretation and explanation

The measures on the tables are an average over three values. The measures for both programs were done one after another, so the PC's performance was nearly the same all along the experimentations.

The difference between static and dynamic load balancing, is the attribution of work for each thread. In static, it's the programmer that chooses which thread works on which data. On balancing, the programmer lets the threads work by themselves and follow their "natural order". That's why we can observe a better performance for pc_dynamic.

As each thread works at its own pace, if a thread finishes its work before another, it can directly works on another task. Some threads can be faster than other, so attributing work dynamically assure that the tasks are done as soon as possible. In static, if a thread is faster than another, it will finish its task before and the main thread will wait until the slowest has finished.

So, performances are better in dynamic load balancing. But there are some limits. We can see that the static load balancing approach has better scalability. Indeed, time execution keeps decreasing as we increase thread number. That is not the case with dynamic approach. From 8 threads, times execution – so performance – are almost constant.

I'd say that static load balancing has a better scalability but not better performance, and dynamic load balancing doesn't have a nice scalability from eight threads, but it's more efficient. We can also see that with parallel programming (from 2 threads), the dynamic approach is really more effective.

But it's not the case in serial programming (with only one thread). I think this is the way I develop it that does this. Indeed, in dynamic, I create another class "Index" for making a static object over the method. It's not the most efficient way to make dynamic load balancing, but it was easier for me to understand the principle of attribute work dynamically.

5. Screenshot results

- pc_static.java (default number of threads and 10 threads)

```
maevl@LAPTOP-EPBLAAD0 MINGW64 ~/Documents/Cours/CAU/Sem2/Mult
$ java pc_static
Num thread: 4
Thread 0 execution time: 251 ms.
Thread 1 execution time: 667 ms.
Thread 2 execution time: 1061 ms.
Thread 3 execution time: 1417 ms.

Total exection time: 1418 ms.
Between 1 and 199999, there are : 17984 prime numbers.
```

```
maevl@LAPTOP-EPBLAAD0 MINGW64 ~/Documents/Cours/CAU/Sem2/Mult
$ java pc_static 10
Num thread: 10
Thread 0 execution time: 77 ms.
Thread 1 execution time: 238 ms.
Thread 2 execution time: 360 ms.
Thread 3 execution time: 462 ms.
Thread 4 execution time: 592 ms.
Thread 5 execution time: 609 ms.
Thread 6 execution time: 677 ms.
Thread 7 execution time: 751 ms.
Thread 8 execution time: 818 ms.
Thread 9 execution time: 857 ms.

Total exection time: 890 ms.
Between 1 and 199999, there are : 17984 prime numbers.
```

- pc_dynamic.java (default number of threads and 6 threads)

```
maevl@LAPTOP-EPBLAAD0 MINGW64 ~/Documents/Cours/CAU/Sem
$ java pc_dynamic
Num thread: 4

Thread 0 : 962 ms.
Thread 1 : 962 ms.
Thread 2 : 962 ms.
Thread 3 : 962 ms.

Total execution time: 963ms
Between 1 and 199999, there are : 17984 prime numbers.
```

```
maevl@LAPTOP-EPBLAAD0 MINGW64 ~/Documents/Cours/CAU/Sem
$ java pc_dynamic 6
Num thread: 6

Thread 0 : 895 ms.
Thread 1 : 895 ms.
Thread 2 : 894 ms.
Thread 3 : 895 ms.
Thread 4 : 895 ms.
Thread 5 : 895 ms.

Total execution time: 896ms
Between 1 and 199999, there are : 17984 prime numbers.
```

6. Java source code

pc_static.java

```
import java.time.LocalDate;

class pc_static {

    private static final int NUM_END = 200000;
    private static int NUM_THREAD = 4;


    public static void main(String [] args) {
        int i, counter, gap, leftover;
        /*
        *   gap and leftover are used for distributing data to threads
        *   in case of NUM_END/NUM_THREAD isn't an integer
        */
        long beginning, ending, diffTime;


        if (args.length==1) {
            NUM_THREAD = Integer.parseInt(args[0]);
        }

        gap = NUM_END/NUM_THREAD;
        leftover = NUM_END%NUM_THREAD;

        System.out.println("Num thread: "+NUM_THREAD);

        PrimeThreadStatic[] threads = new PrimeThreadStatic[NUM_THREAD];
        counter = 0;
        beginning = System.currentTimeMillis();

        for(i = 0; i < NUM_THREAD - 1; i++){
            threads[i] = new PrimeThreadStatic(i*gap, (i+1)*gap);
            threads[i].start();
        }
        // the last thread get the left numbers in case of NUM_END/NUM_THREAD isn't an int
        threads[NUM_THREAD-1] = new PrimeThreadStatic(i*gap, (i+1)*gap + leftover);
        threads[NUM_THREAD-1].start();

        try{
            for(i = 0; i < NUM_THREAD; i++){
                threads[i].join();
                counter += threads[i].counter;
            }
        }
        catch(InterruptedException e) {}

        ending = System.currentTimeMillis();
        diffTime = ending - beginning;
        for(i = 0; i < NUM_THREAD; i++)  System.out.println("Thread "+i+" execution time:
"+threads[i].exeTime+" ms.");
        System.out.println("\nTotal exection time: "+diffTime+" ms.");
        System.out.println("Between 1 and "+(NUM_END - 1)+", there are : "+counter+" prime numbers.");
    }
}

class PrimeThreadStatic extends Thread {
    int begin, end, counter;
    long exeTime;

    PrimeThreadStatic(int begin, int end){
        this.begin = begin;
        this.end = end;
        this.counter = 0;
    }

    public void run(){
        this.exeTime = 0;
        long startTime = System.currentTimeMillis();
        for(int x = this.begin; x < this.end; x++){
            if(isPrime(x)) this.counter++;
        }
        long endTime = System.currentTimeMillis();
        this.exeTime = endTime - startTime;
    }

    private static boolean isPrime(int x){
        int i;
        if(x <= 1) return false;
        for(i = 2; i < x; i++){
            if(x%i == 0) return false;
        }
        return true;
    }

}
//end pc_static.java
```

pc_dynamic.java

```
import java.time.LocalDate;

class pc_dynamic {

    private static final int NUM_END = 200000;
    private static int NUM_THREAD = 4;

    public static void main(String [] args) {
        int i, j, counter;
        long beginning, ending, diffTime;
        counter = 0;

        if (args.length==1) {
            NUM_THREAD = Integer.parseInt(args[0]);
        }

        Thread mainThread = Thread.currentThread();
        Index index = new Index(0);

        PrimeThreadDynamic[] threads = new PrimeThreadDynamic[NUM_THREAD];

        beginning = System.currentTimeMillis();
        for(i = 0; i < NUM_THREAD; i++){
            threads[i] = new PrimeThreadDynamic(NUM_END, index);
            threads[i].start();
        }

        try{
            for(i = 0; i < NUM_THREAD; i++){
                threads[i].join();
                counter += threads[i].counter;
            }
        }
        catch (InterruptedException e) {}
        ending = System.currentTimeMillis();
        diffTime = ending - beginning;
        System.out.println("Num thread: "+NUM_THREAD+"\n");

        for(i = 0; i < NUM_THREAD; i++){
            System.out.println("Thread "+i+" : "+threads[i].exeTime+" ms.");
        }

        System.out.println("\nTotal execution time: "+diffTime+"ms");
        System.out.println("Between 1 and "+(NUM_END - 1)+", there are : "+counter+" prime numbers.");

      }

}


class PrimeThreadDynamic extends Thread {
    int max, counter;
    Index index;
    long exeTime;

    PrimeThreadDynamic(int numend, Index index){
        this.index = index;
        this.max = numend;
        this.counter = 0;
        this.exeTime = 0;
    }

    public void run(){
        int indexInt = index.getNextValue();
        long startTime = System.currentTimeMillis();
        /*
        *   threads stay in the loop until the
        *   static parameter index is equal to max (=NUM_END)
        */
        while(indexInt < this.max){
            if(isPrime(indexInt)) this.counter++;
            indexInt = index.getNextValue();
        }

        long endTime = System.currentTimeMillis();
        long diffTime = endTime - startTime;
        this.exeTime += diffTime;
    }

    private boolean isPrime(int x){
        int i;
        if(x <= 1) return false;
        for(i = 2; i < x; i++){
            if(x%i == 0) return false;
        }
        return true;
    }

}
```

```
// for easier manipulation of the static parameter
class Index {
    public static int value = 0;

    public Index(int value){
        this.value = value;
    }

    public synchronized int getNextValue(){
        this.value += 1;
        return this.value;
    }
}
//end pc_dynamic.java
```