Maeva Lecavelier (마에바 르꺄블리에) – 50191580

Multicore Computing – proj1: problem2

1. Goal:

Given a JAVA source code for matrix multiplication (the source code MatmultD.java is available on our class webpage), modify the JAVA code to implement parallel matrix multiplication that uses multi-threads. You may use either a static or a dynamic load balancing approach. Your code also should print (1) the execution time of each thread, (2) execution time for the entire thread computation, and (3) sum of all elements in the resulting matrix.

I made it in dynamic load balancing.

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 dynamic load balancing approach.

Execution time (ms)	1	2	4	6	8	10	12	14	16	32
Dynamic	372	331	240	222	212	214	229	238	234	243

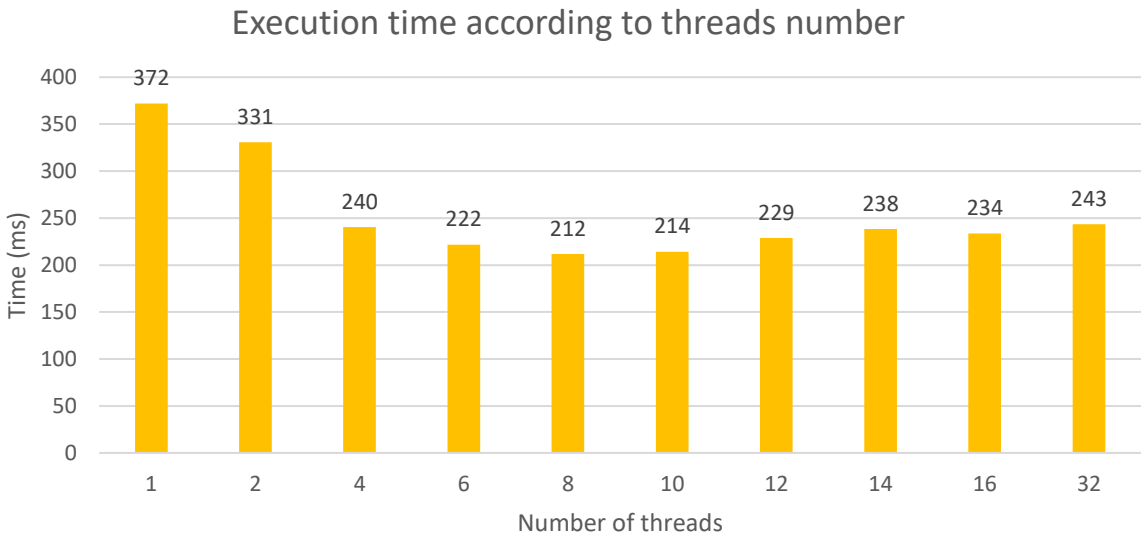
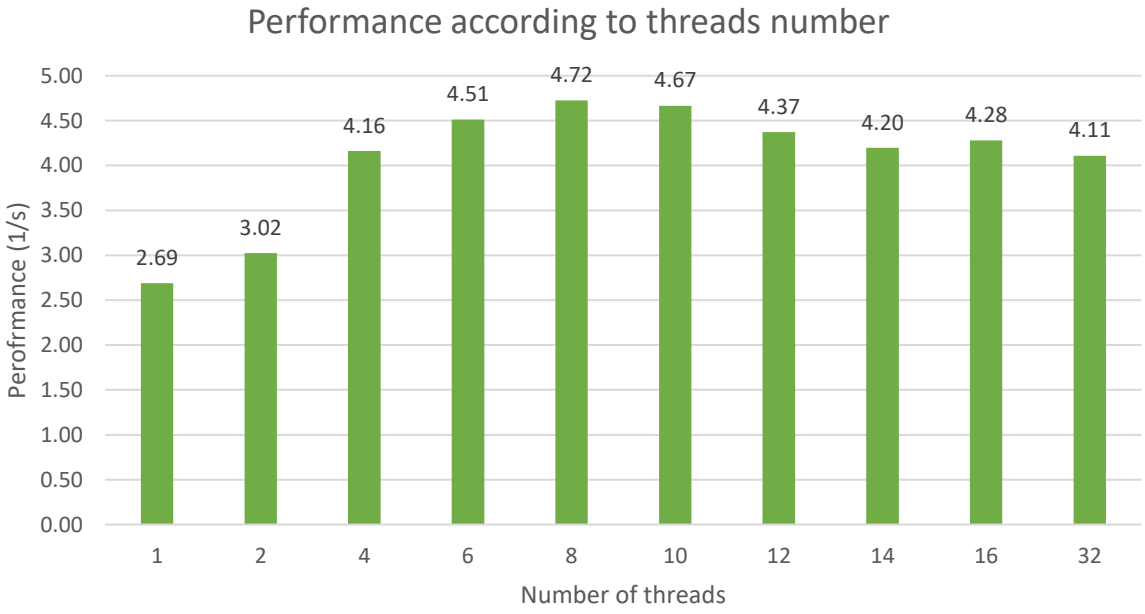


Table 2: performance according to thread number for a dynamic load balancing approach.

Performance (1/s)	1	2	4	6	8	10	12	14	16	32
Dynamic	2.69	3.02	4.16	4.51	4.72	4.67	4.37	4.20	4.28	4.11



4. Interpretation and explanation

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

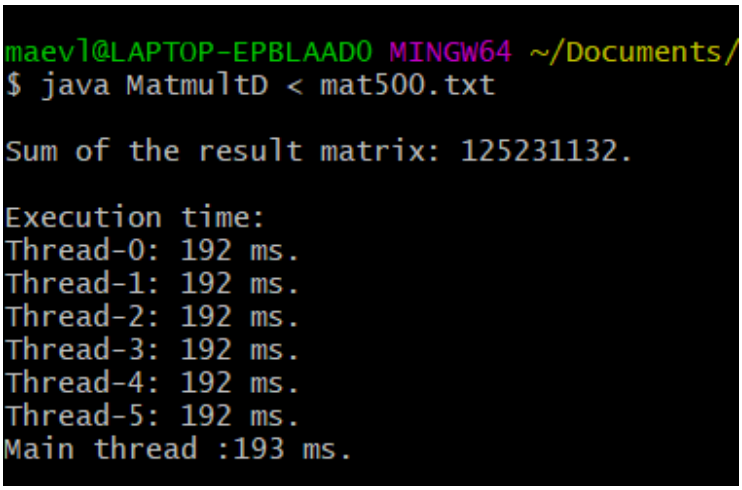
I've chosen dynamic approach because I think that it was more accurate and logical to do so. It'd take time to think about how exactly do it in static and, according to the previous problem, it seems more efficient to do it in dynamic.

We can see that, like the previous problem, dynamic approach is quite scalable for a few amounts of threads, but, this time, it's around 4 threads that the max performance is reached. And after that, the execution time – so performance – are almost constant. I think the little differences are due to my PC's performance that may vary a little.

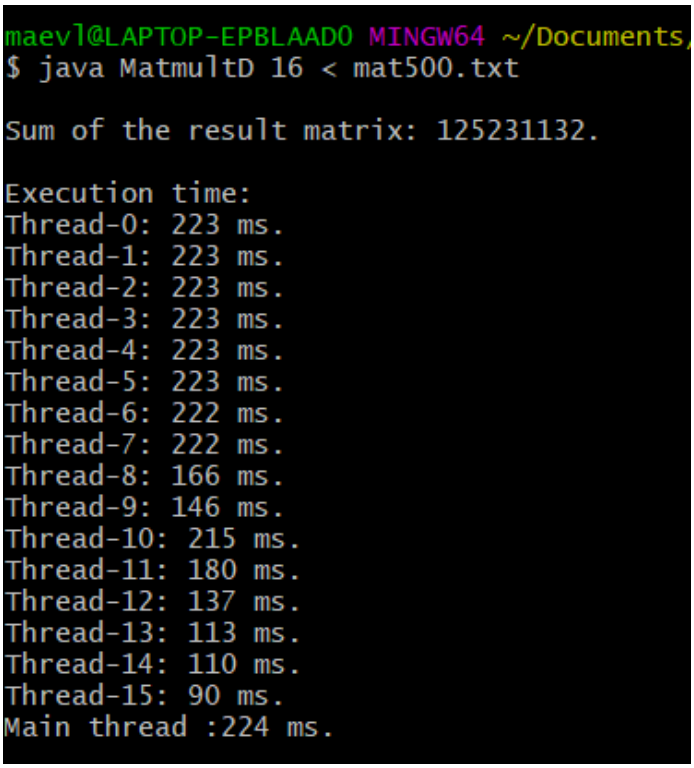
My code is maybe the cause of the stagnation. I use a static variable, which can be incremented only by a static and synchronized method. So, the more threads we use, the more queue there will be to access this method. And so, about 4 threads, the results are nearly constant, even if I increase number of threads. That is one limit of dynamic approach, one must use a shared variable and so make its access synchronized, and therefore, make one chunk of the program serialized.

5. Screenshot results

- For default number of threads (6)



- For 16 threads



6. Java source code

MatmultD.java

```
import java.util.*;
import java.lang.*;
import java.time.LocalDate;

public class MatmultD {
    private static Scanner sc = new Scanner(System.in);
    public static int[][] result;

    public static void main(String [] args) {
        int NUM_THREAD = 0;
        int i;
        long startTime, endTime, diffTime;
        if(args.length == 1) NUM_THREAD = Integer.valueOf(args[0]);
        else NUM_THREAD = 6;

        int a[][] = readMatrix();
        int b[][] = readMatrix();
        int[] dimensions = new int[2];
        dimensions[0] = getDimension(a)[0];
        dimensions[1] = getDimension(b)[1];
        result = new int[dimensions[0]][dimensions[1]];

        startTime = System.currentTimeMillis();
        MultThread[] threads = new MultThread[NUM_THREAD];

        for(i = 0; i < NUM_THREAD; i++){
            threads[i] = new MultThread(a, b, result);
            threads[i].start();
        }

        try{
            for(i = 0; i < NUM_THREAD; i++){
                threads[i].join();
            }
        }
        catch (InterruptedException e) {}
        endTime = System.currentTimeMillis();
        diffTime = endTime - startTime;
        System.out.println("\nSum of the result matrix: "+sumMatrix(result)+".\n");
        System.out.println("Execution time:");
        for(i = 0; i < NUM_THREAD; i++){
            System.out.println(threads[i].getName()+": "+threads[i].exeTime+" ms.");
        }
        System.out.println("Main thread :"+diffTime+" ms.");



    }

    public static void printMatrix(int[][] mat) {
      System.out.println("Matrix["+mat.length+"]["+mat[0].length+"]");
        int rows = mat.length;
        int columns = mat[0].length;
        int sum = 0;
        for (int i = 0; i < rows; i++) {
          for (int j = 0; j < columns; j++) {
            System.out.printf("%4d " , mat[i][j]);
            sum+=mat[i][j];
          }
          System.out.println();
        }
        System.out.println();
        System.out.println("Matrix Sum = " + sum + "\n");
      }

    public static int[][] readMatrix() {
        int rows = sc.nextInt();
        int cols = sc.nextInt();
        int[][] result = new int[rows][cols];
        for(int i = 0; i < rows; i++){
            for(int j = 0; j < cols; j++){
                result[i][j] = sc.nextInt();
            }
        }
        return result;
    }

    public static int[] getDimension(int[][] matrix){
        int rows = matrix.length;
        int columns = matrix[0].length;
        int[] res = new int[2];
        res[0] = rows;
        res[1] = columns;
        return res;
    }
```

```
    public static int sumMatrix(int[][] mat){
        int rows = mat.length;
        int cols = mat[0].length;
        int sum = 0;
        for(int i = 0; i < rows; i++){
            for(int j = 0; j < cols; j++){
                sum += mat[i][j];
            }
        }
        return sum;
    }
}


class MultThread extends Thread {
    //static object for making a commun point to all thread
    static int[] couple = new int[]{0,-1};
    long exeTime;
    static int[][] finalMatrix, a, b;
    int m, n, p;

    MultThread(int[][] a, int[][] b, int[][] res){
        // a: [m][n], b: [n][p], final: [m][p]
        this.a = a;
        this.b = b;
        this.finalMatrix = res;
        this.exeTime = 0;
        this.m = a.length;
        this.p = b[0].length;
        this.n = a[0].length;
    }

    public void run(){
        int[] coupleCopy = getNextCouple(this.p);
        long startTime = System.currentTimeMillis();

        while(coupleCopy[0] < this.m){
            finalMatrix[coupleCopy[0]][coupleCopy[1]] = 0;
            for(int k = 0; k < this.n; k++){
                finalMatrix[coupleCopy[0]][coupleCopy[1]] +=
this.a[coupleCopy[0]][k]*this.b[k][coupleCopy[1]];
            }
            coupleCopy = getNextCouple(this.p);
        }

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

    //static and synchronized so only one thread can access it
    public static synchronized int[] getNextCouple(int p){
        couple[1] += 1;
        if(couple[1] == p){
            couple[0] += 1;
            couple[1] = 0;
        }
        /*
        *  I use a copy because, if couple is updated while
        *  a thread is running in the while for loop, this may
        *  lead to (1) an out of range error and (2) some couples
        *  can be skiped.
        */
        int[] copy = new int[2];
        copy[0] = couple[0];
        copy[1] = couple[1];
        return copy;
    }

    public synchronized int reset(int x){
        return -1;
    }
}

//end MatmultD.java
```