Rapport between Static Load Balancing and Dynamic Load Balancing to calculate the product of 2 matrices.

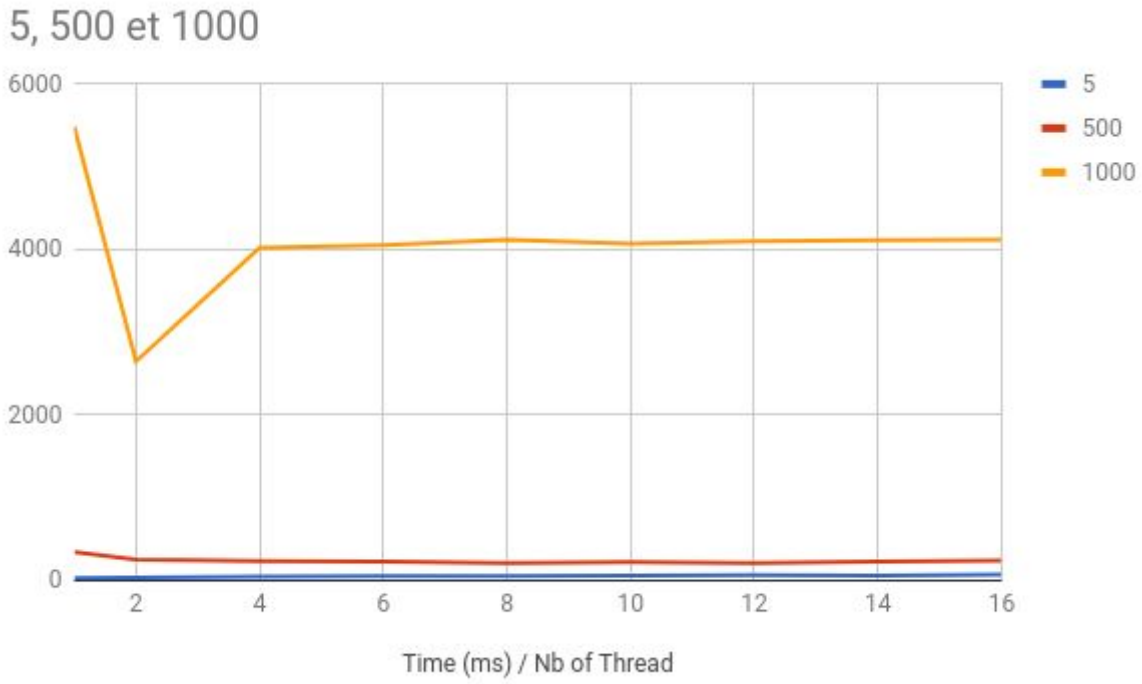
This experience was realized by QUIEF Hippolyte *(50171350) using a computer with a processor Intel i7 2.7GHz and 8Go of RAM.*

This experience have the goal to show the limit of thread for a program. Depending of the complexity of this program, use high number of thread can or cannot be useful.

To realize this experience, we launch *x* threads who will compute data from a file. Each thread will be given the same amount of work but not necessarily with the same complexity.

Time (ms) / Nb of Thread	1	2	4	6	8	10	12	14	16
5	32	37	45	52	54	60	66	60	70
500	343	253	237	229	211	220	213	230	240
1000	5486	2647	4021	4055	4120	4070	4102	4114	4121

1. *Result of the computation with matrice of 5, 500 and 1000*

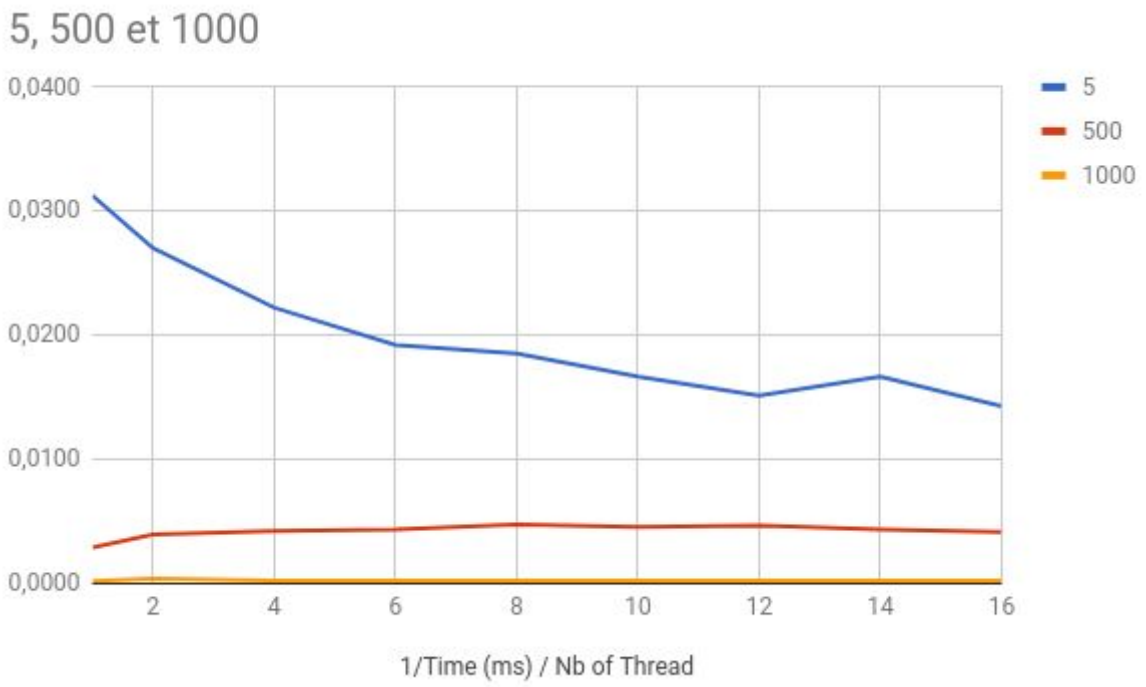


2. The graph created with the data of the array 1

By these 2 results, we can say that for big amount of data, it's interesting to use thread to reduce the time of computation.

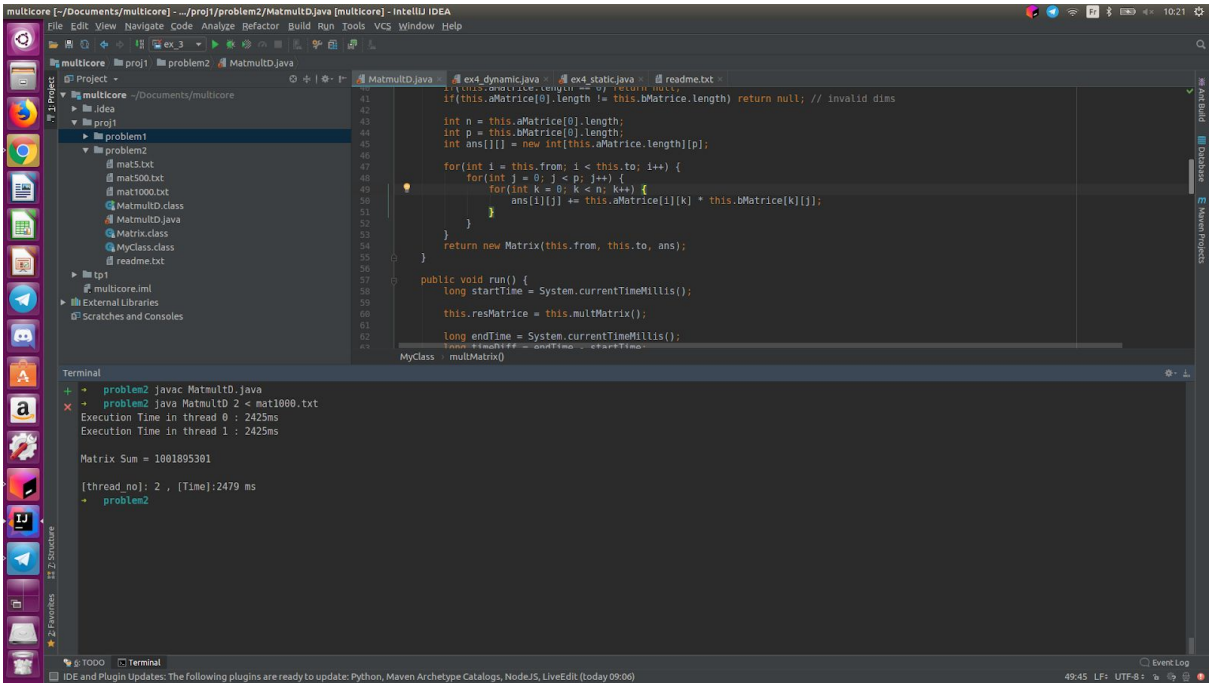
1/Time (ms) / Nb of Thread	1	2	4	6	8	10	12	14	16
5	0,0313	0,0270	0,0222	0,0192	0,0185	0,0167	0,0152	0,0167	0,0143
500	0,0029	0,0040	0,0042	0,0044	0,0047	0,0045	0,0047	0,0043	0,0042
1000	0,0002	0,0004	0,0002	0,0002	0,0002	0,0002	0,0002	0,0002	0,0002

3. Performance of the program depending of the number of thread



4. Graph of the performance

These graphs show that use more thread is not all the time interesting in term of performance. It can reduce it ! So depending of the computation needed, the number of useful thread can change. The number of thread is also determine by the capacity of computation of the computer use.



5. Execution of the program with 2 threads and matrice of 1000

Code of the program

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

class Matrix {
    public int from;
    public int to;
    public int[][] matrice = null;

    Matrix(int from, int to, int[][] matrice) {
        this.from = from;
        this.to = to;
        this.matrice = matrice;
    }
}

class MyClass extends Thread {
    private int[][] aMatrice;
    private int[][] bMatrice;
    private int from;
    private int to;
    private int id;
    public Matrix resMatrice = null;

    MyClass(int[][] aMatrice, int[][] bMatrice, int from, int to, int id) {
        this.aMatrice = aMatrice;
        this.bMatrice = bMatrice;
        this.from = from;
        this.to = to;
```

```
    this.id = id;
}

private Matrix multMatrix() {
    if(this.aMatrice.length == 0) return null;
    if(this.aMatrice[0].length != this.bMatrice.length) return null; // invalid dims

    int n = this.aMatrice[0].length;
    int p = this.bMatrice[0].length;
    int ans[][] = new int[this.aMatrice.length][p];

    for(int i = this.from; i < this.to; i++) {
        for(int j = 0; j < p; j++) {
            for(int k = 0; k < n; k++) {
                ans[i][j] += this.aMatrice[i][k] * this.bMatrice[k][j]; // calc the matrice
            }
        }
    }
    return new Matrix(this.from, this.to, ans);
}

public void run() {
    long startTime = System.currentTimeMillis();

    this.resMatrice = this.multMatrix();

    long endTime = System.currentTimeMillis();
    long timeDiff = endTime - startTime;

    System.out.println("Execution Time in thread " + this.id + " : " + timeDiff + "ms");
```

```
    }
}

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

    public static void main(String [] args)
    {
        int thread_no = 0;
        if (args.length == 1) thread_no = Integer.valueOf(args[0]);
        else thread_no = 1;

        int a[][] = readMatrix();
        int b[][] = readMatrix();
        int res[][] = new int[a.length][a[0].length];

        // start calc time
        long startTime = System.currentTimeMillis();

        MyClass[] myClasses = new MyClass[thread_no];
        for (int i = 0; i < thread_no; ++i) {
            if (i == thread_no - 1) {
                myClasses[i] = new MyClass(a, b, (a.length / thread_no) * i, (a.length / thread_no)
* (i + 1) + (a.length % thread_no), i);
            } else {
                myClasses[i] = new MyClass(a, b, (a.length / thread_no) * i, (a.length / thread_no)
* (i + 1), i);
            }
            myClasses[i].start();
```

```
    }
    for (int i = 0; i < thread_no; ++i) {
        try {
            myClasses[i].join();
            // Recreate matrice
            Matrix c = myClasses[i].resMatrice;
            for (int x = c.from; x < c.to; x++) {
                for (int y = 0; y < c.matrice[0].length; y++) {
                    res[x][y] = c.matrice[x][y];
                }
            }
        } catch (InterruptedException e) {
            System.err.println(e);
        }
    }

    long endTime = System.currentTimeMillis();

    printMatrix(res);

    System.out.printf("[thread_no]:%2d , [Time]:%4d ms\n", thread_no, endTime-startTime);
}

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 void printMatrix(int[][] mat) {
        if (mat == null) {
            System.out.println("The matrice cannot be print.");
            return;
        }
        // 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");
    }
}
```