PAR LABORATORY 3 REPORT

Q1 2025/2026

Arnau Hernández Navarro and Felipe Arturo Núñez Gómez

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

FIB

INDEX

1.Brief Introduction

The purpose in this laboratory assignment is to explore different parallel strategies based on iterative and recursive task decomposition. In this case, the Mandelbrot Set is the one to be computed. It is a particular set of points in a complex domain.

To do this, we have a tool called Tareador which has the ability to visualize Task Dependency Graphs (TDG) and measure performance metrics with a number of processors to be chosen by us (T1 and T∞).

The strategies that will be used are the following:

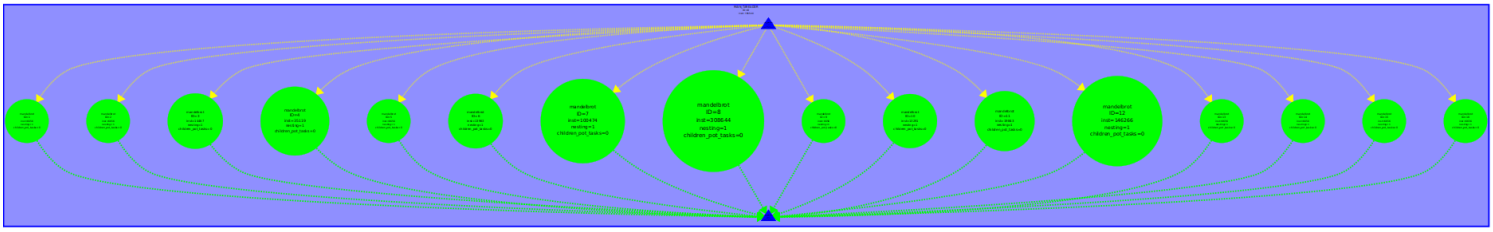
- Iterative
    - Original version without and with -d and -h arguments.
    - Finer grain parallel strategy.
    - Column of tiles parallel strategy.
- Recursive
    - Leaf recursive parallel strategy.
    - Tree recursive parallel strategy.

2.Iterative task decomposition analysis

Before continuing, we must mention that the original version with no arguments and the ones with the arguments -d and -h are purely emphasized to understand the data dependencies they carry and disable them due to data sharing to exploit parallelism among tasks.

2.1.Original Version with no arguments

This first strategy to be analyzed has no arguments. As we can see in the following image, Tareador has not detected any dependencies and every task seen can be executed in parallel.



Proceeding to the simulation with Tareador, we obtained the following results for the variables T1 and T∞:

- T1 = 705 ns.
- T∞ = 308 ns.

With these two values we can obtain the value of the Parallelism:

- Parallelism = T1 / T∞ = 2.28.

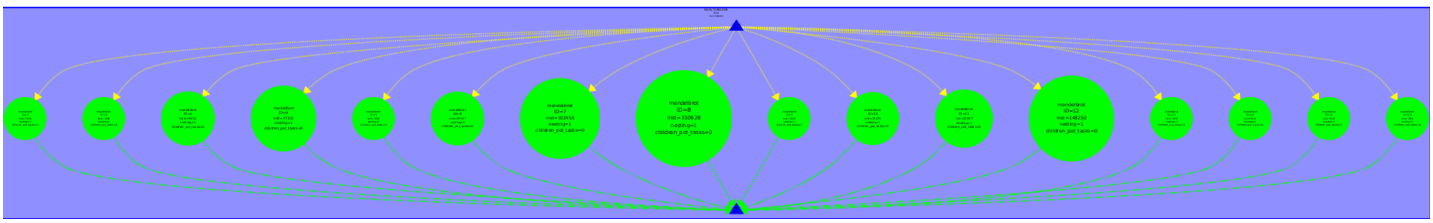
As we can see in the image, the graph indicates that load unbalance is present due to some tasks that have to perform a higher number of instructions than others (this is indicated by the size of the green balls).

2.2.Original Version with the -d argument

This time, the code is to be executed with the -d argument which allows to generate a computed image of The Mandelbrot Set.

This image corresponds with the execution of the code without disabling the variables that cause dependencies in the code. Observing can be seen that the tasks follow a queue making impossible to parallelize.

To get rid of the Tasks Dependencies, we executed Tareador to discover what was causing the problem. It was found out that the variable responsible for the dependencies was “X11_COLOR_fake”. We removed the dependency using the following code “tareador_disable_object(&X11_COLOR_fake)” and executing the code with Tareador again this was the result:



Now can be seen that no dependencies are left in the code.

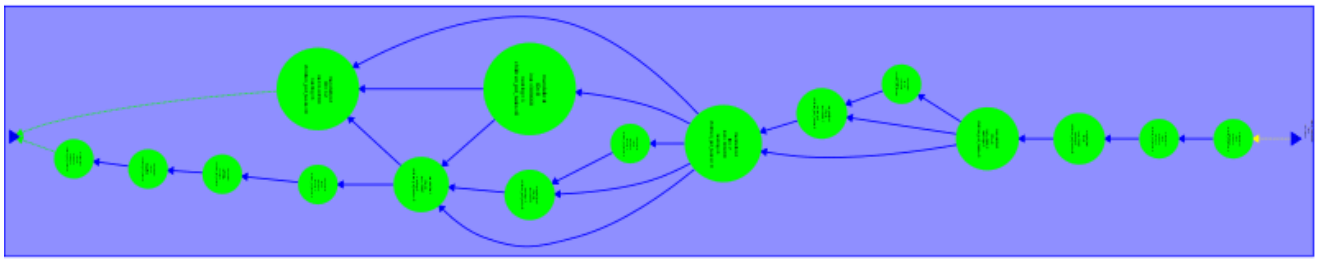
Proceeding to the simulation with Tareador, we obtained the following results for the variables T1, T∞ and Parallelism:

- T1 = 729 ns.
- T∞ = 310 ns.
- Parallelism = T1 / T∞ = 2.35.

As before, we can also see load unbalance due to some tasks that have to perform a higher number of instructions than others.

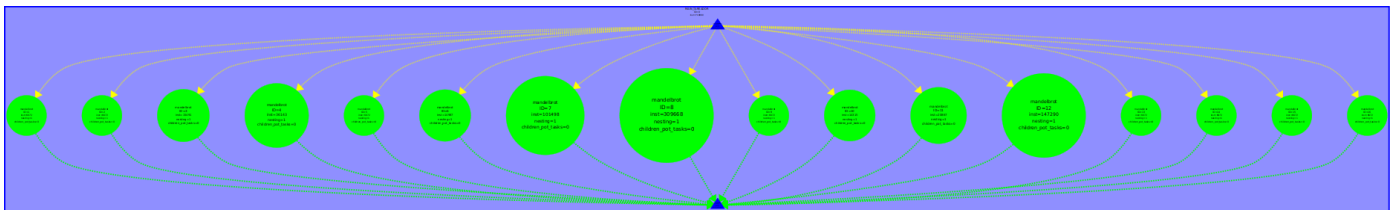
2.3.Original Version with the -h argument

The -h argument produces a histogram of values of the computed image. The resulting TDG with dependencies is as follows:



Now, even more dependencies can be seen in the graph in contrast with the previous two versions. Again, in order to get rid of the Dependencies, we execute Tareador being the variable named “Histogram” the problematic one causing dependencies.

To solve it, we added the following line of code tareador_disable_object(histogram). Executing the code with tareador again has given the next result:



No dependencies are left in the code as shown in the graph.

Proceeding to the simulation with Tareador, we obtained the following results for the variables T1, T∞ and Parallelism:

- T1 = 713 ns.
- T∞ = 309 ns.
- Parallelism = T1 / T∞ = 2.30.

As before, we can also see load unbalance due to some tasks that have to perform a higher number of instructions than others.

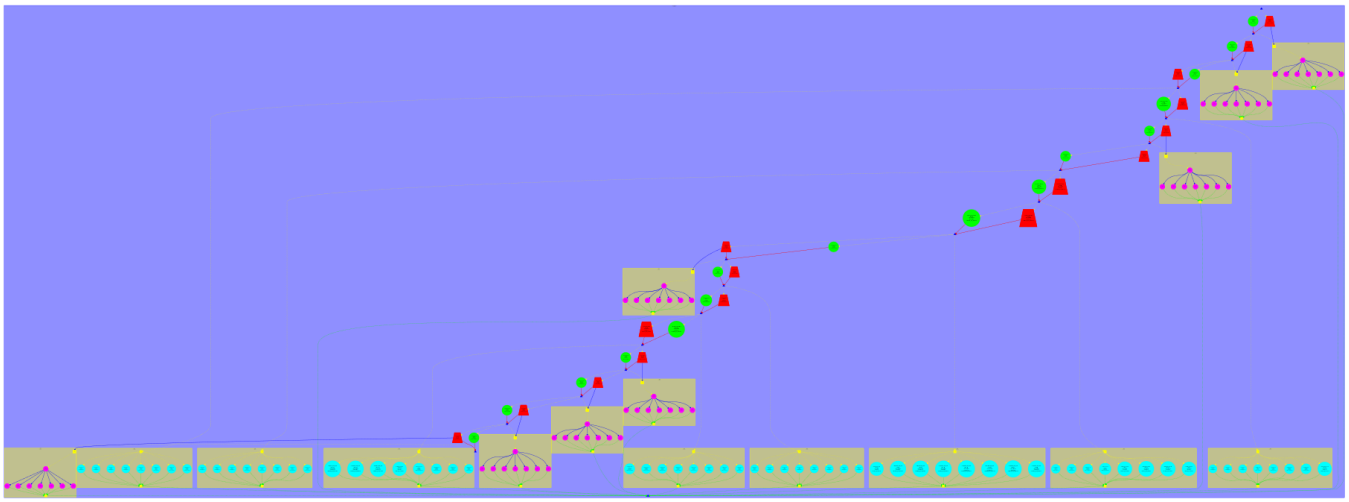
2.4.Finer grain parallel strategy

This approach consists of dividing the code into smaller tasks to be executed in parallel. We were asked to modify the original version of the code with the following conditions:

- Check vertical borders is a task.
- Check horizontal borders is a task.
- Entirety of the if-else statement is a task and this included two other sub-tasks:
    - Each py iteration of fill all with the same value is a task.
    - Each py iteration of computation of a tile is a task.

We also detected before making the changes that there were two dependencies due to the variables “Hmatrix” and “equal”. Also found a data sharing dependence due to M[y][x]. All these dependencies were deactivated with tareador_disable_object().

After implementing the changes we obtained the following graph as Tareador was executed:



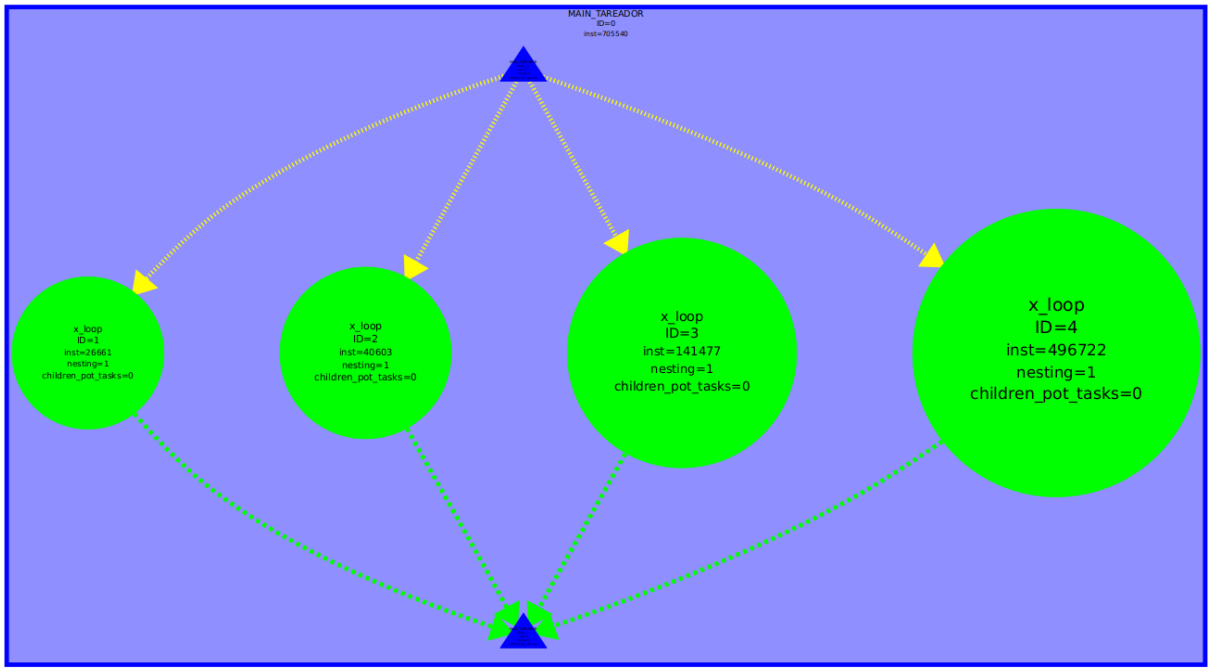
Proceeding to the simulation with Tareador, we obtained the following results for the variables T1, T∞ and Parallelism:

- T1 = 705 ns.
- T∞ = 159 ns.
- Parallelism = T1 / T∞ = 4.43.

This time, no load unbalance seems to be happening as all tasks have the same size and even execute a similar number of instructions.

2.5.Column of tiles parallel strategy

This strategy consists in assigning a task for each column. In order to achieve this, we interchanged the x and y loops and created a task for each x iteration.
Owing to the fact that each task worked with his own column of tile we have not found any dependencies between them, as we can see in the TDG. Therefore, there is not any data sharing or synchronization problem. On the contrary, we do have load unbalance as there are tasks executing way more instructions than the others.



Proceeding with the Tareador simulation, we obtained the following results for the variables T1, T∞ and Parallelism:

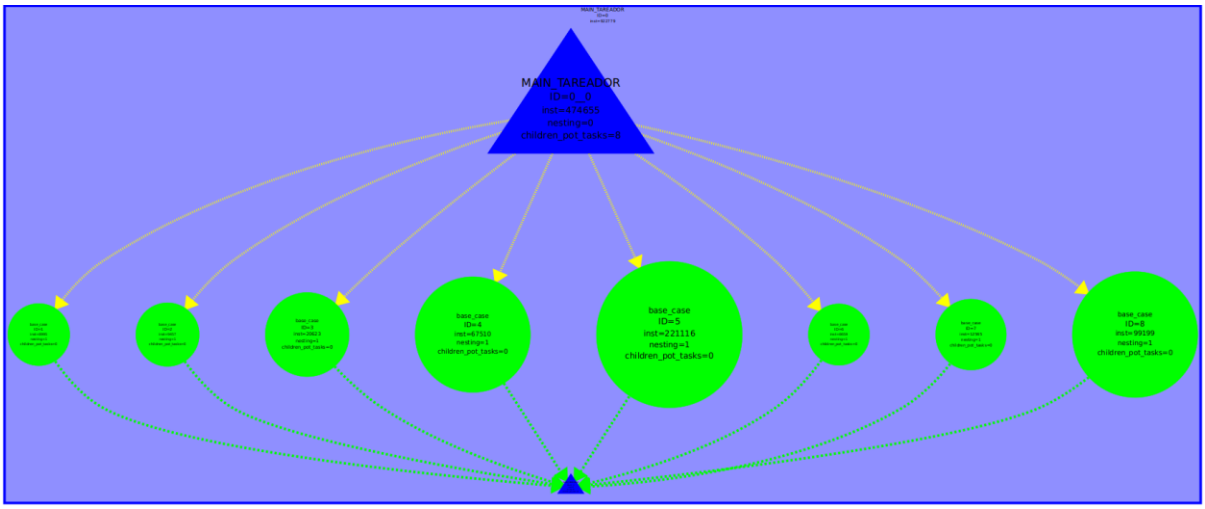
- T1 = 705 ns.
- T∞ = 496ns.
- Parallelism = T1 / T∞ = 1.42.

3.Recursive task decomposition analysis

3.1.Leaf recursive parallel strategy

Leaf strategy is based on distributing the tasks only among the leaf nodes. To accomplish this we created tasks only for the base cases.

Since only leaf nodes are defined as tasks, there cannot be any dependencies, as no subsequent task depends on a leaf node. For this reason, there is not any data sharing or synchronization problem. Nevertheless, we do have load unbalance as there are tasks executing way more instructions than the others.

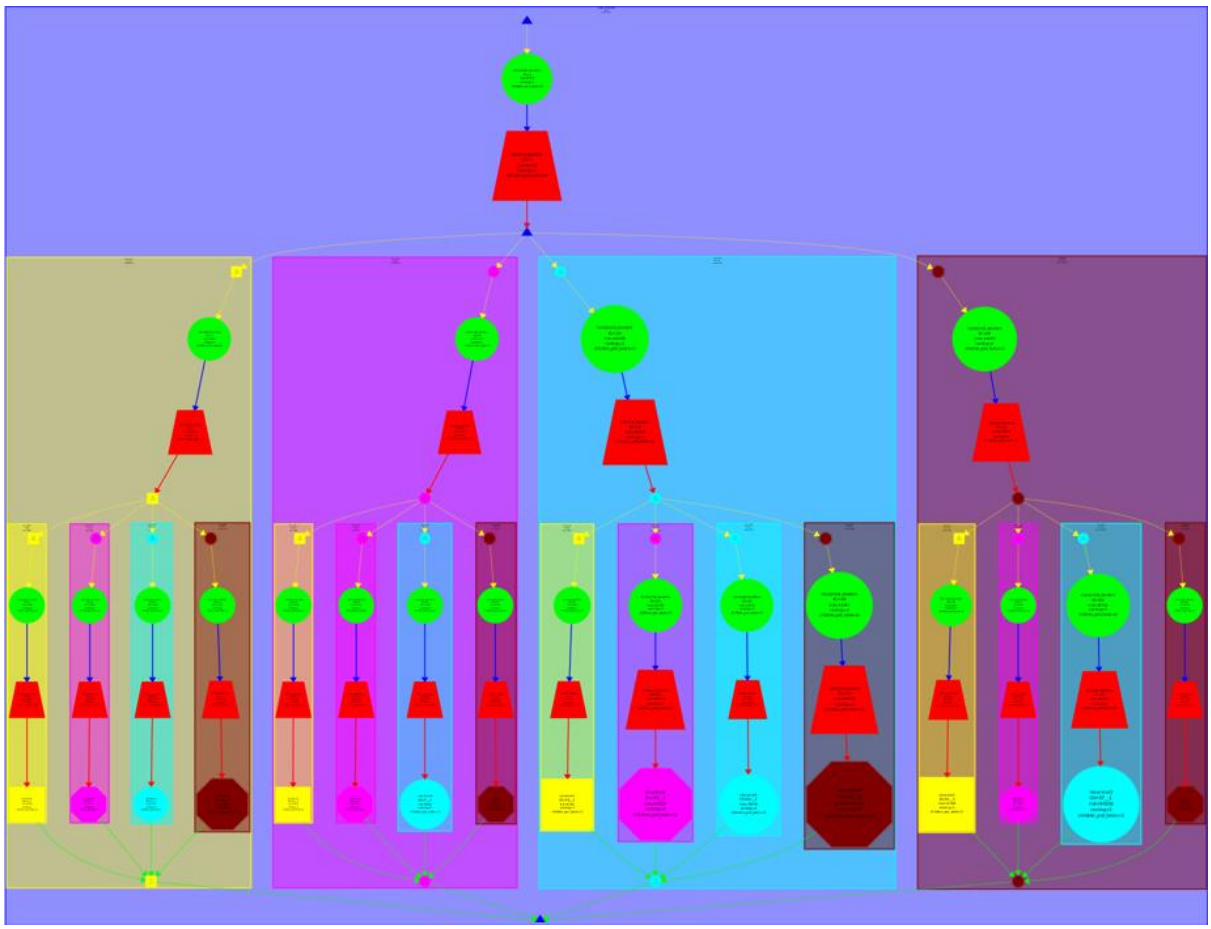


Proceeding with the Tareador simulation, we obtained the following results for the variables T1, T∞ and Parallelism:

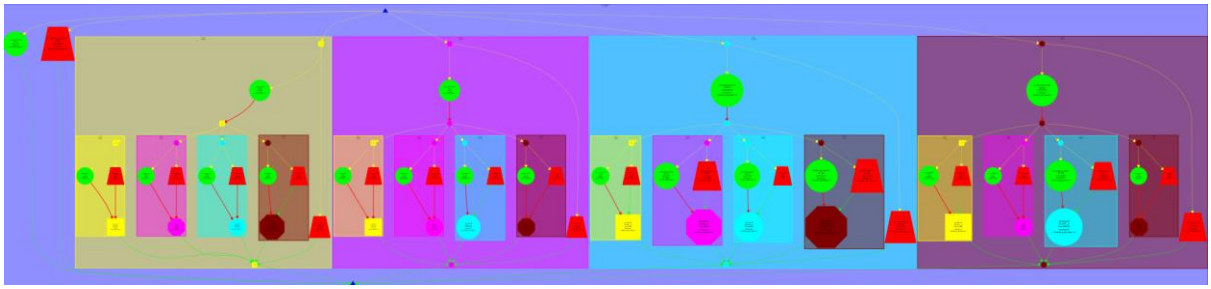
- T1 = 922ns.
- T∞ = 559ns.
- Parallelism = T1 / T∞ = 1.64.

3.2.Tree recursive parallel strategy

Tree strategy involves dividing the tasks among all the nodes of the recursive tree. In addition to that, we were asked to exploit the parallelism between both the vertical and horizontal border check loops. In order to do so, we created tasks for vertical and horizontal border check loops, base cases and each recursive call.
We had dependencies and synchronization problems due to data sharing with the variables equals and Hmatrix.
Moreover, we do have load unbalance as there are tasks executing way more instructions than the others.



We disabled the dependencies by disabling the variables equals and Hmatrix and we obtained the following TDG:



Proceeding with the Tareador simulation, we obtained the following results for the variables T1, T∞ and Parallelism:

- T1 = 922ns.
- T∞ = 307.
- Parallelism = T1 / T∞ = 4.17.

4.Conclusions

Task decomposition	Strategy	Run arguments	T1	T∞	Parallelism	Load unbalance: yes/no
Iterative	Original		705	308	2.28	yes
	Original	-d	729	310	2.35	yes
	Original	-h	713	309	2.30	yes
	Fine grain		705	159	4.43	no
	column		705	496	1.42	yes
Recursive	Leaf		922	559	1.64	yes
	Tree		922	307	3.003	yes
Best task decomposition	Strategy	Reason why				
	Fine Grain	As shown, it has the best approach for parallelism (4.43) with only an execution time of 159 nanoseconds.				