UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

LAB 3: Deliverable
The Mandelbrot set

Arnau Garcia Gonzalez
Jan Santos Bastida

Table of contents

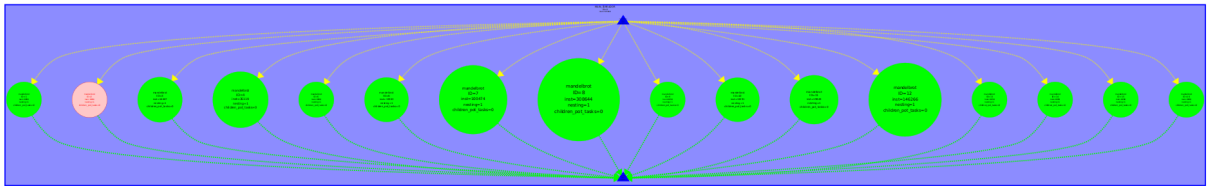
Iterative task decomposition

Original with no Arguments

Code

Original code from the statement. The file name is mandel-seq-iter-tar.c.

Tareador TDG

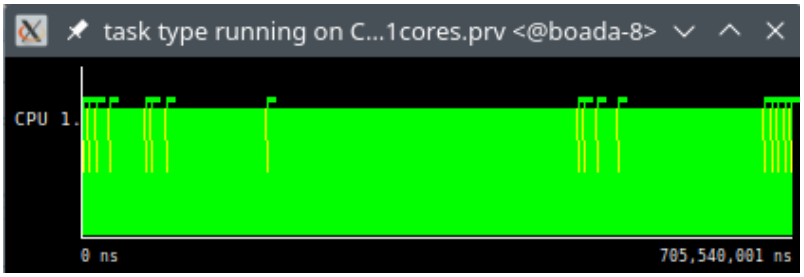


TDG Original without args

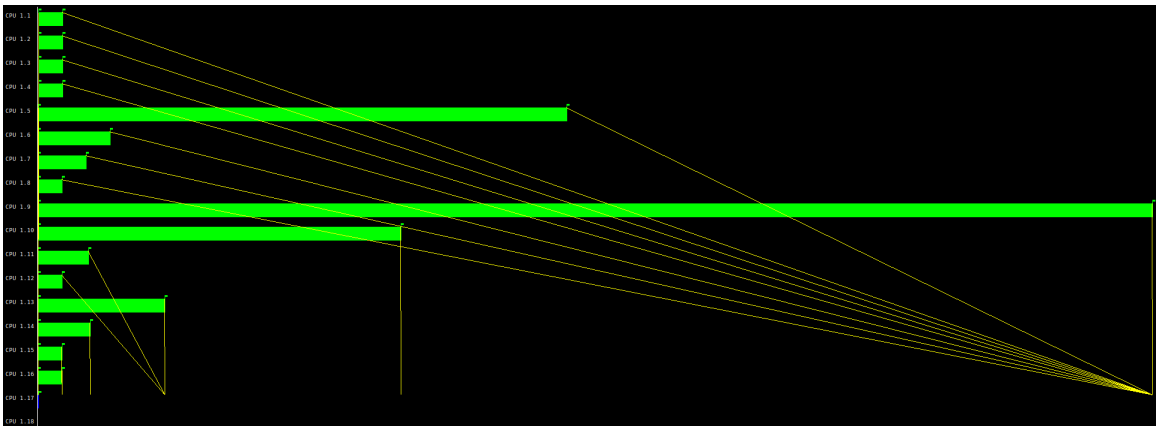
Dependence Analysis

We can see there are no dependencies on the graph and all the created tasks are executed in parallel. We can also see load unbalance in the number of instructions of the different tasks, some of them having much more than others.

T ∞ Analysis



T1 = 705.540.001 ns



Tinf = 308.750.001 ns

Parallelism = 705.540.001/308.750.001 = 2.285

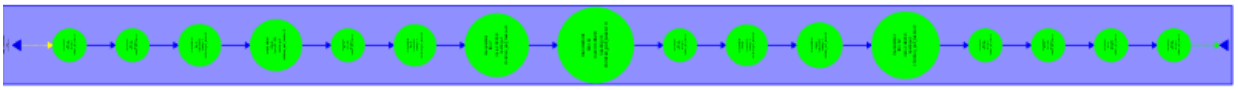
We can see that we can get the Tinf with just 16 processors, since that's the maximum number of tasks executing in parallel at the same time.

We can also see again the load unbalance, some tasks take more time than others and therefore some processors are not used most of the time.

Original with -d

Code
With dependencies: same code as Original.
No dependencies: the file name is mandel-seq-iter-tarv2.c.

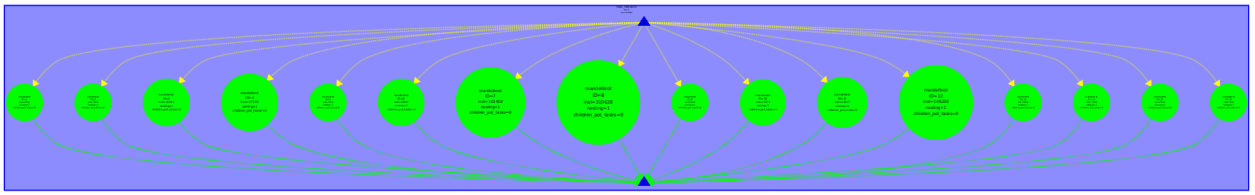
Tareador TDG with dependencies



TDG Original -d with dependencies

Dependence Analysis
As we can see on the TDG we have dependencies on the original version with argument -d. If we check which variable causes the dependency we can see that the cause is the variable X11-COL. We can disable the variable to execute the code without dependencies.
We can see as well in this version load unbalance. Some of the tasks have more instructions than others.

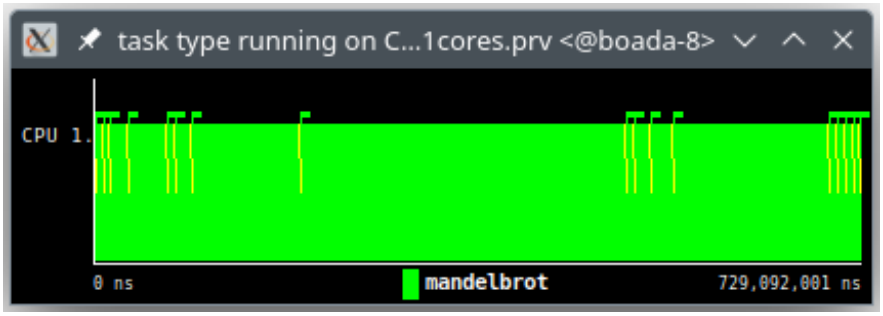
Tareador TDG with no dependencies



TDG original -d without dependencies

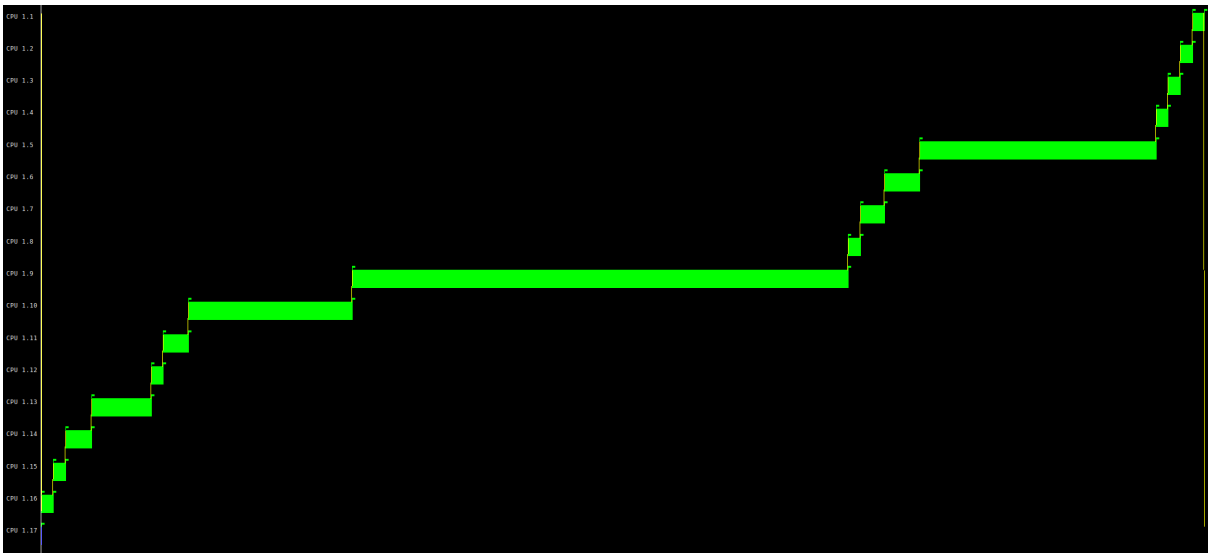
T ∞ Analysis

Same time T1 in both versions, with or without dependencies:



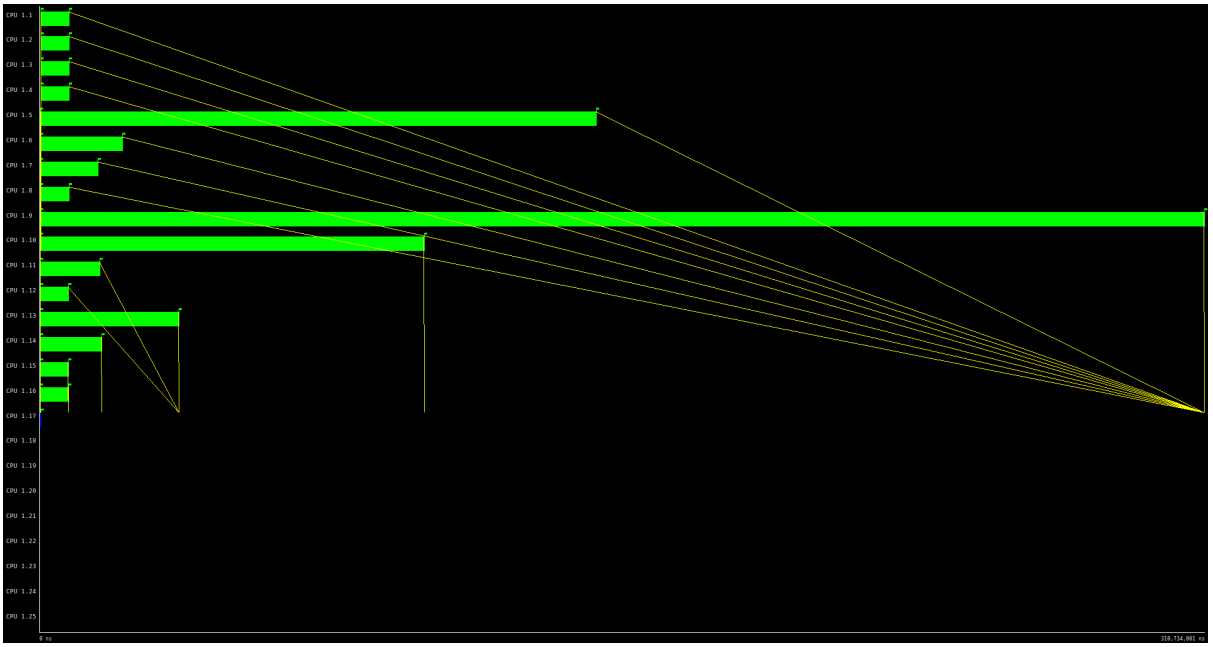
T1 = 729.092.001 ns

Version with dependencies:



Tinf = 728.904.001 ns
Parallelism = 729.092.001 / 728.904.001 = 1

Version without dependencies:



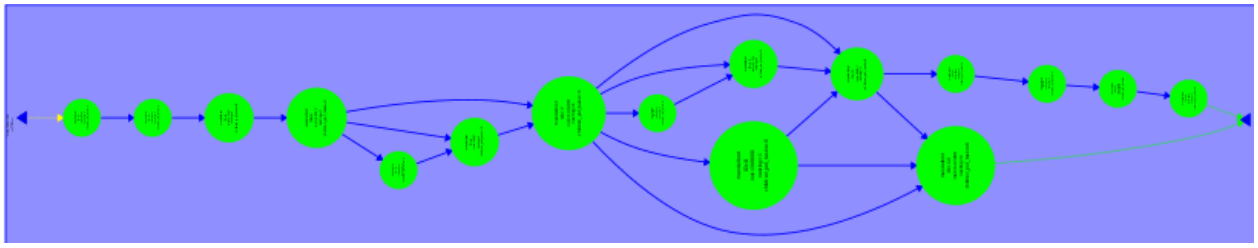
Tinf = 310.734.001 ns
Parallelism = 729.092.001 / 310.734.001 = 2.346

As we can see we can exploit the maximum parallelism with 16 processors. In the version with dependencies we have almost no parallelism because the dependence makes the program run sequentially.
We can appreciate the load unbalance, we can see that some processors are most of the time without working.

Original with -h

Code
With dependencies: Same code as original
Without dependencies: The file name is mandel-seq-iter-tarv3.c

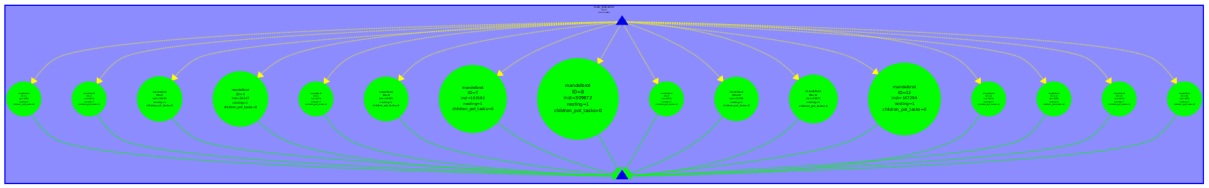
Tareador TDG with dependencies



TDG Original -h with dependencies

Dependence Analysis
As it can be seen on the TDG the program has dependencies. After checking which variable causes the dependence we identify that it is the vector histogram. If we disable the variable we can execute the code with its maximum parallelism.

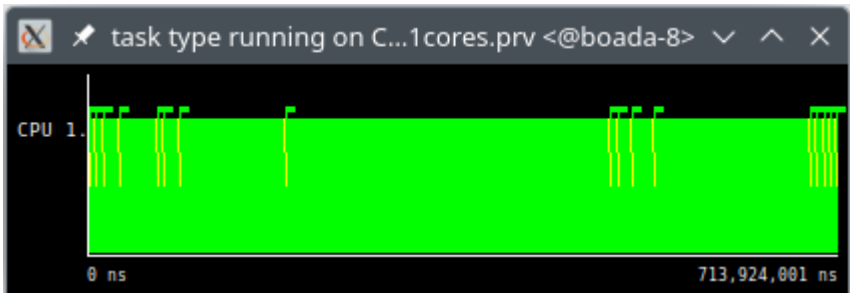
Tareador TDG with no dependencies



TDG original -h without dependences

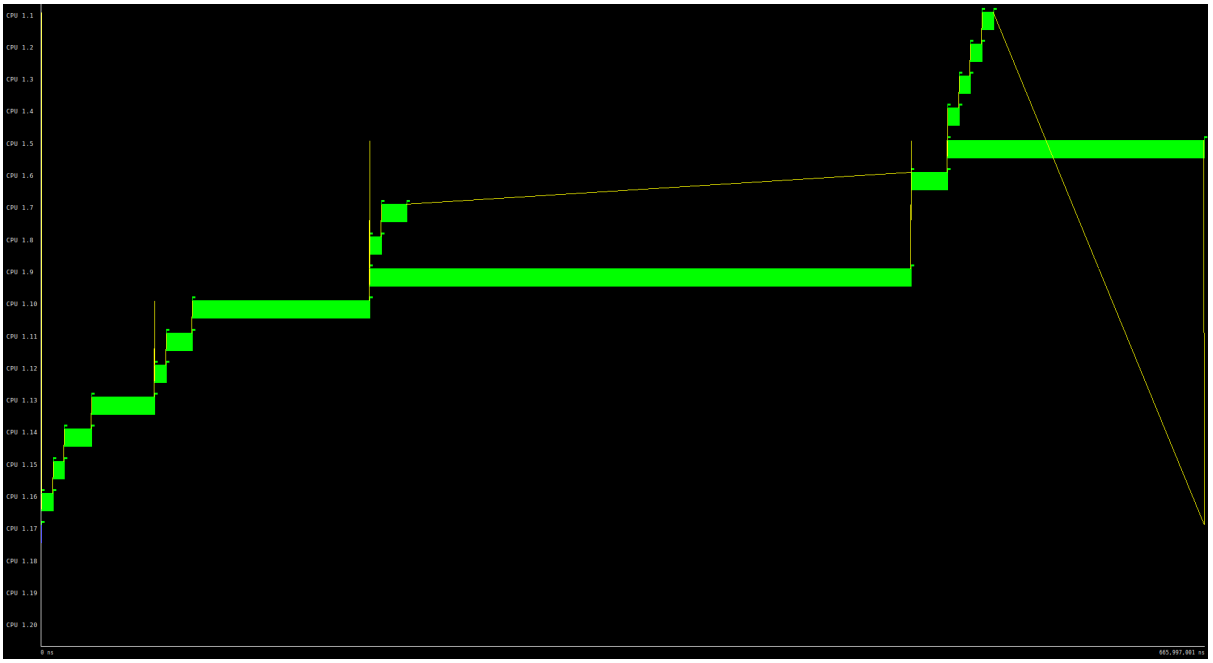
T ∞ Analysis

Same time T1 (very similar) in both versions, with or without dependencies:



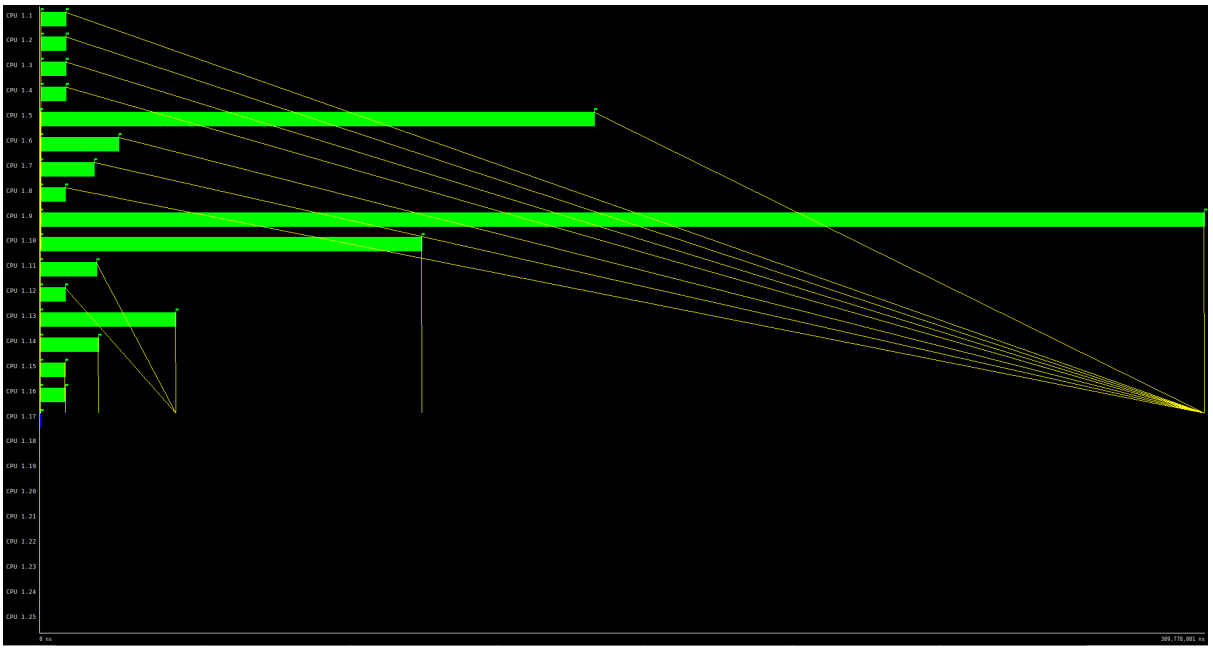
T1 = 713.924.001 ns

Version with dependencies:



Tinf = 665.997.001 ns
Parallelism = 713.924.001 / 665.997.001 = 1.072

Version without dependencies:



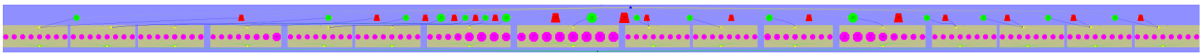
Tinf = 309.778.001 ns
Parallelism = 713.924.001 / 309.778.001 = 2.305

We can see that we can get the Tinf with just 16 processors. And here again in the version with a dependence the parallelism is so small.
We can also appreciate again the load imbalance, some tasks take more time than others and that causes some processors not to be used most of the time.

Finer grain

Code
Without dependencies: The file name is mandel-seq-iter-tarv4.c.

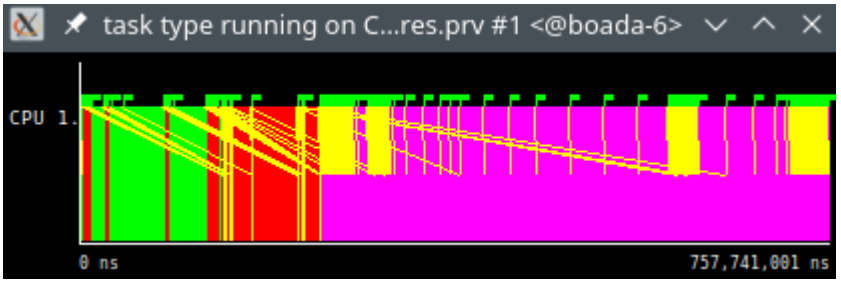
Tareador TDG with/without dependencies



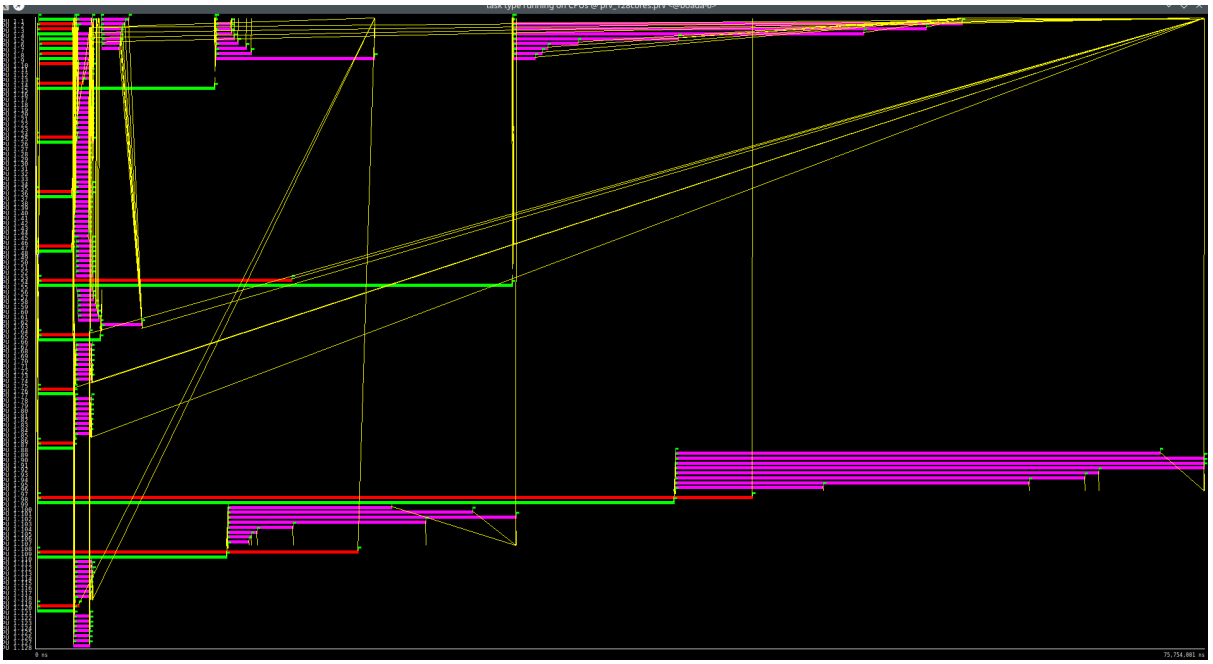
TDG Finer grain without dependencies

Dependence Analysis
As we can see in the TDG, there are no dependencies between tasks because from the beginning we have changed the way equal works (we have divided the variable in two, equalX and equalY), so that the variable does not create any dependencies between tasks.

T ∞ Analysis



T1 = 757.741.001 ns



Tinf = 75.754.001 ns
Parallelism = 757.741.001 / 75.754.001 = 10.003

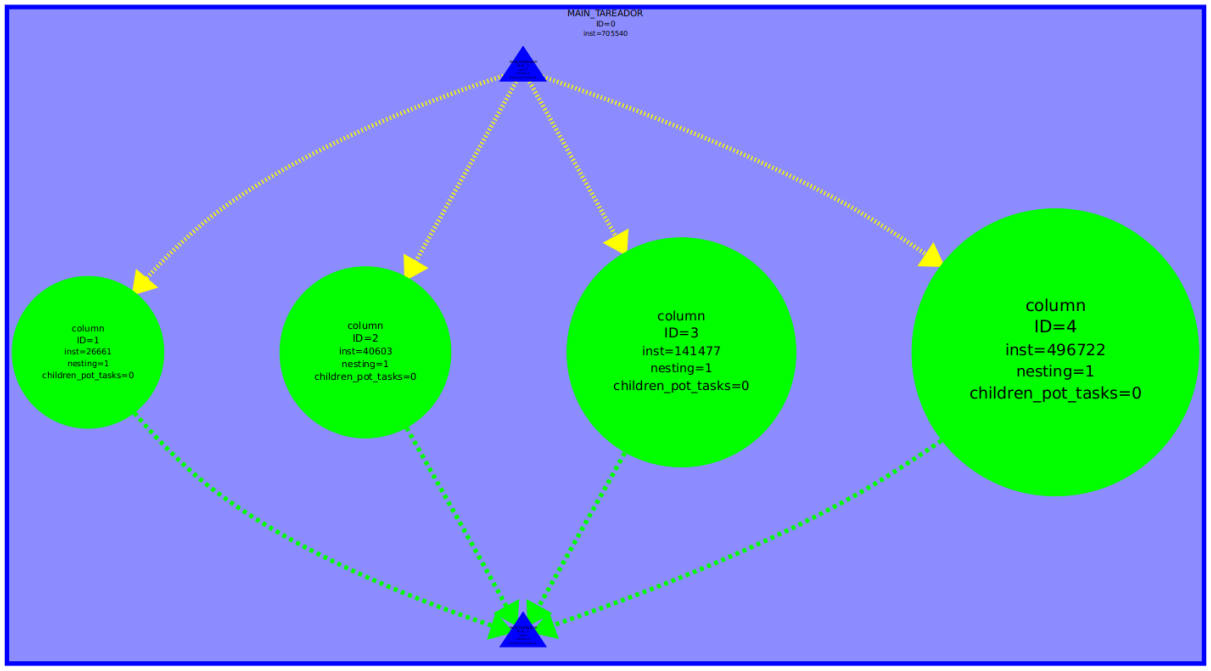
We can appreciate that there is a load unbalance as some tasks work a lot more than others. That's caused because some points of the mandelbrot set require more calculation than others.

Column

Code

mandel-seq-iter-tarv5.c (modified using the original code as a base, as the Statement said)

Tareador TDG without dependencies

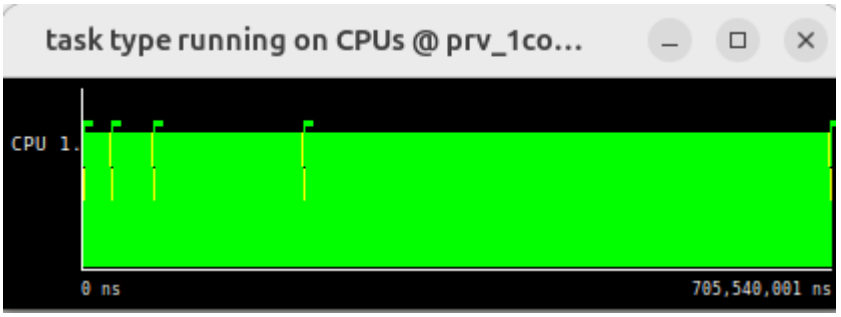


TDG column without dependencies

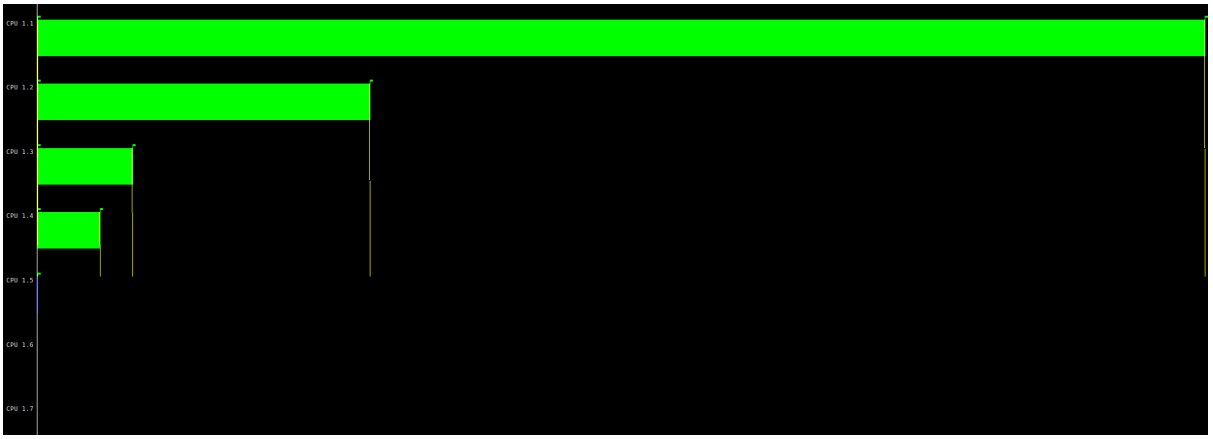
Dependence Analysis

We can see in the photograph that there are no dependencies between the horizontal tasks, so we can exploit the program's full parallelism without disabling any variable. We can also appreciate a clearly unbalance between the tasks, as the ones in the right execute many more instructions than the ones in the left.

T ∞ Analysis



T1 = 785.540.001 ns



Tinf = 496.778.001 ns
Parallelism = 785.540.001 / 496.778.001 = 1.581

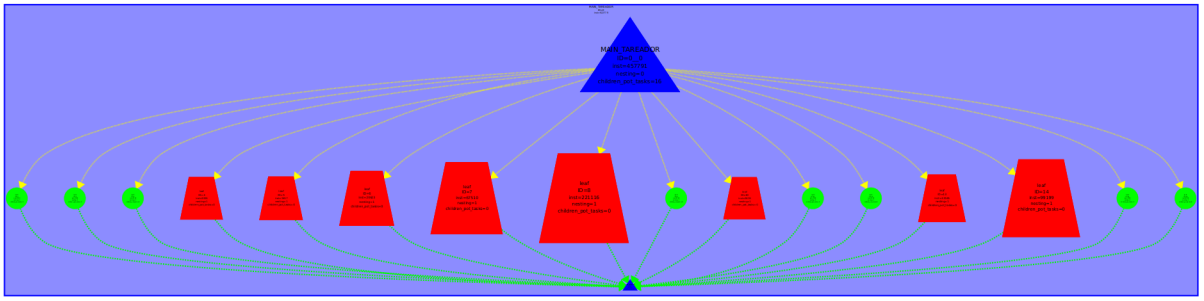
As we don't have dependencies, we can exploit the full parallelism of the program, giving us an upgrade of 1.581. We can appreciate the load unbalance as the first task will be running all the program and the lower one will only be executing for a brief moment.

Recursive task decomposition

Leaf Strategy

Code
mandel-seq–rec-tar-leaf.c

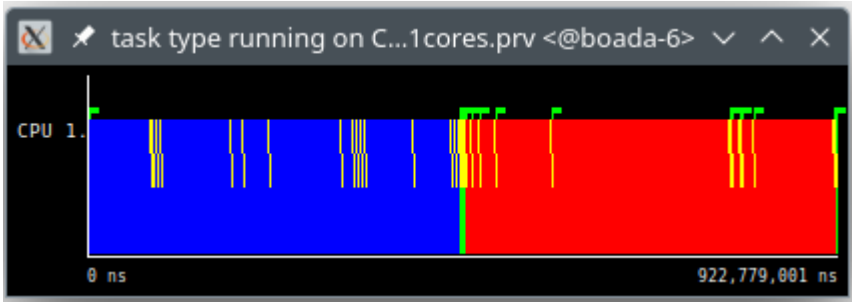
Tareador TDG with no dependencies



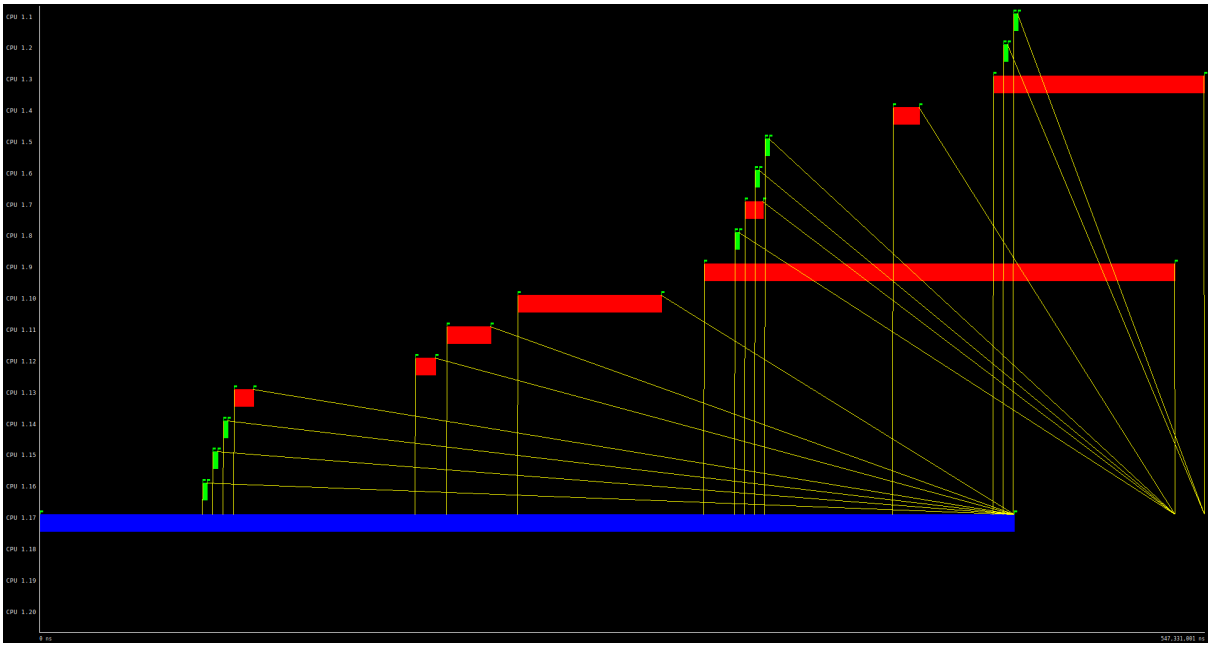
TDG leaf without dependencies

Dependence Analysis
There are no dependencies between the tasks. The main tasks is executed first and then all the recursive ones.

T ∞ Analysis



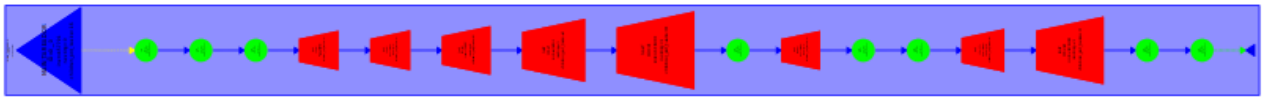
T1 = 922.779.001 ns



Tinf = 547.331.001 ns
Parallelism = 922.779.001 / 547.331.001 = 1.686

As we said, there are no dependencies so the program is fully parallelizable. As we can appreciate, there is load unbalance as the main task is longer than the red ones, which are longer than the green ones. That's because the main tasks can`t be divided into other tasks as we do with the recursive ones, and that’s also why the parallelism obtained is low.

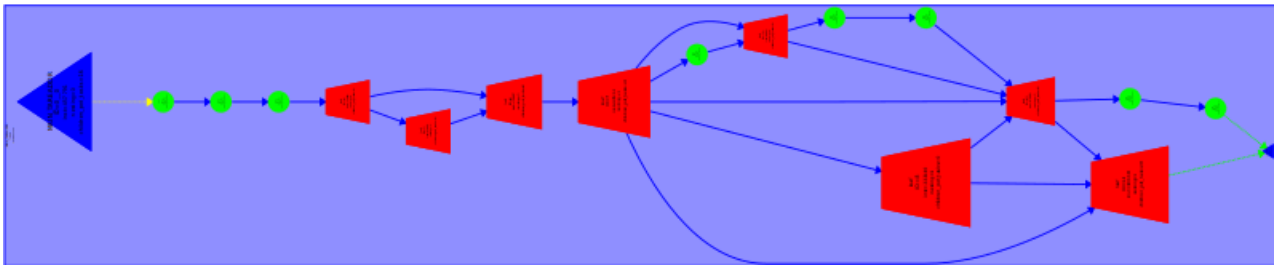
With -d:



TDG leaf with -d with dependencies

This case is using the argument -d to obtain the display of the mandelbrot set. As happened in the iterative case, we obtain a graph which has dependencies between its tasks, caused by the variable X11-COL. If we disable it (like we did in the iterative case), the program will be fully parallelizable.

With -h:



TDG leaf with -h with dependencies

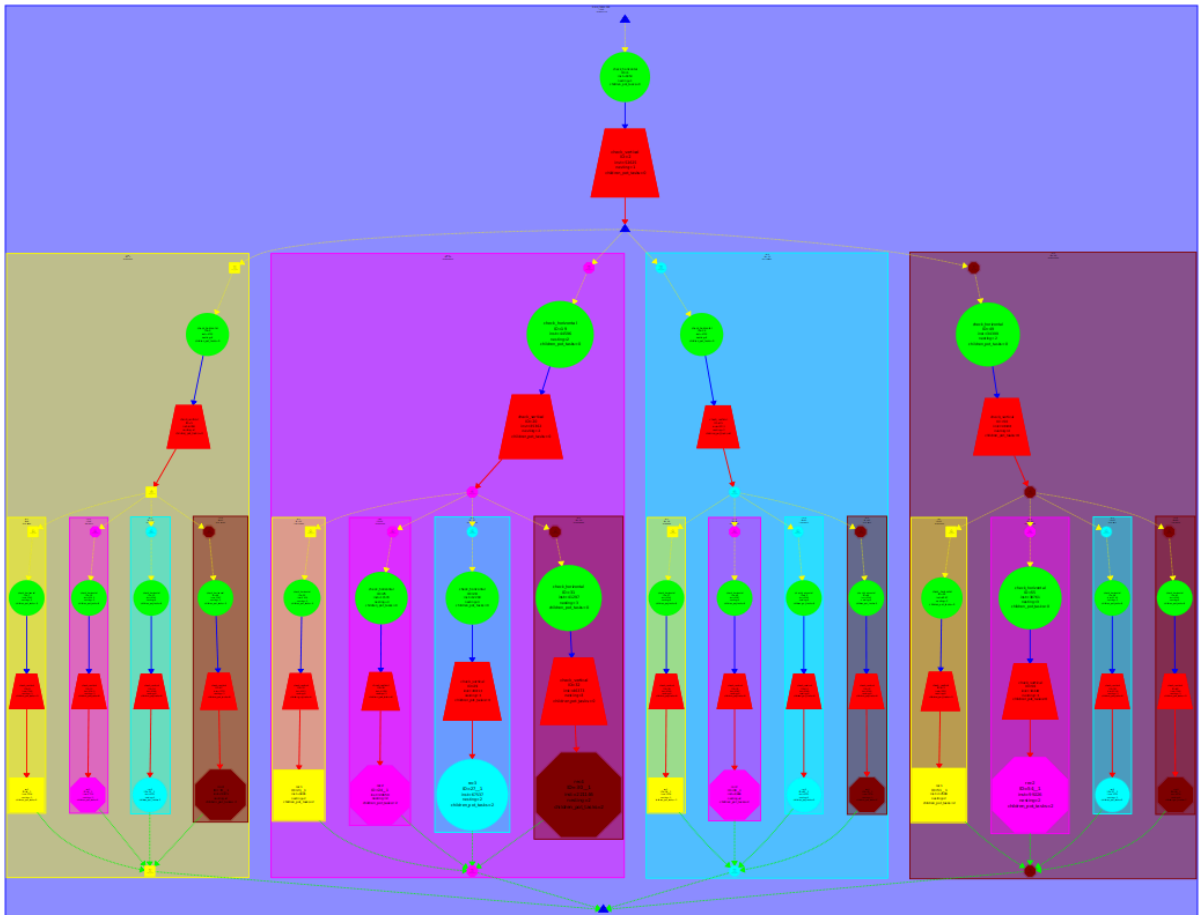
The same happens with argument -h. It causes the dependencies between tasks (because of the variable histogram), as happened in the iterative case. If we disable the variable histogram, the dependencies will disappear and the program will be fully parallelizable.

In all cases we can appreciate the load unbalance between the red tasks and the green ones, caused by the number of calculations which have to be done in the red ones.

Tree Strategy

Code
With dependencies: mandel-seq-rec-tar-tree-dep.c
Without dependencies: mandel-seq-rec-tar-tree.nodep.c

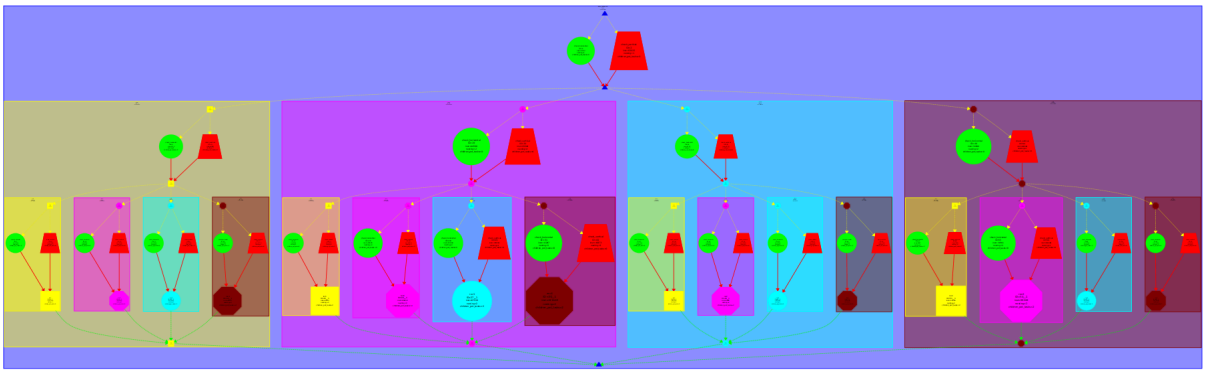
Tareador TDG with dependencies



TDG tree with dependencies

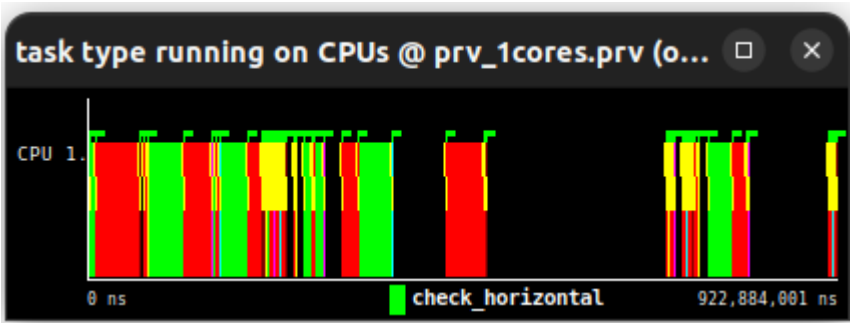
Dependence Analysis
There's a dependency caused by the variable equal, which can be seen between the green and red circles, on top of each rectangle.

Tareador TDG with no dependencies

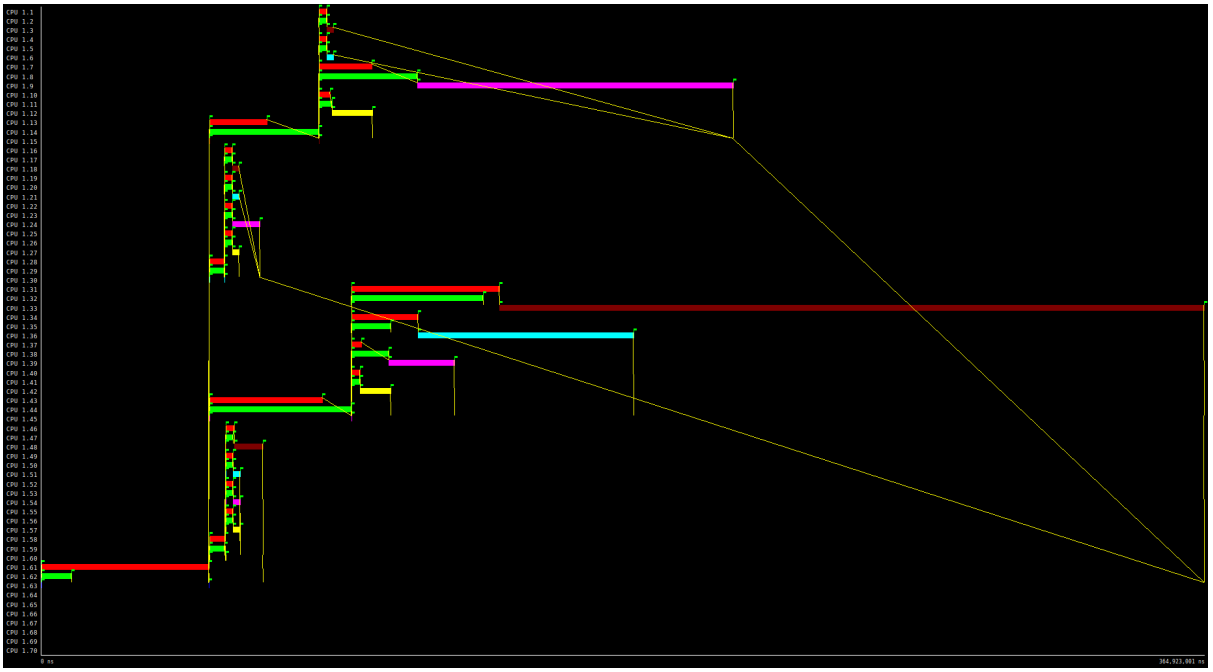


TDG tree without dependencies

T ∞ Analysis



T1 = 922.884.001 ns

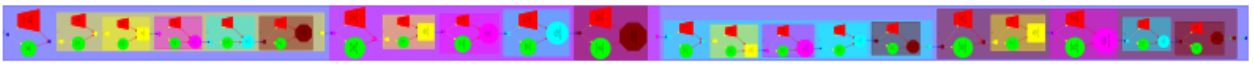


Tinf = 364.923.001 ns

Parallelism = 922.884.001 / 364.923.001 = 2.529

As we have deleted the dependency between the variable equal (dividing it into two other variables, equalH and equalV) we can now exploit all the parallelism of the code. We can see there's load unbalance, as the maroon task is very large compared with the other ones.

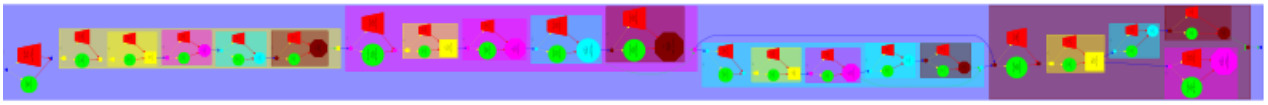
With -d:



TDG tree with -d with dependencies

The same case as before, the code is executed sequentially because the variable X11_COL creates the dependency between tasks.

With -h:



TDG tree with -h with dependencies

Also, the same as before. There's a dependency created because of the variable histogram. It can be deleted if we disable the variable histogram.

Final results

Task Decomposition	Strategy	Run Arguments	T1 (ns)	Tinf (ns)	Parallelism	Load Unbalance
Iterative	original		705.540.001	308.750.001	2.285	Yes
	Original	-d (CDEP)	729.092.00	728.904.001	1	Yes
	Original	-d (SDEP)	729.092.001	310.734.001	2.305	Yes
	Original	-h (CDEP)	713.860.001	665.997.001	1.072	Yes
	original	-h (SDEP)	713.924.001	309.778.001	2.305	Yes
	Finer grain		757.741.001	75.754.001	10.003	Yes
	Column		785.540.001	496.778.001	1.581	Yes
Recursive	Leaf		922.779.001	547.331.001	1.686	Yes
	tree		922.884.001	364.923.001	2.529	Yes