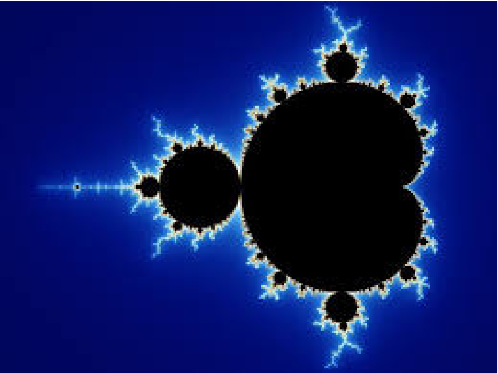


**Universitat Politècnica de Catalunya**

Facultat d’Informàtica de Barcelona

**LAB 3**



Nadia Khier

Èric Díez Apolo

**PAR- Josep Ramon Herrero Zaragoza**

2 d’Abril del 2025

**Índex**

[**1. ITERATIVE TASK DECOMPOSITION ANALYSIS 3**](#_b6uj4s2ap783)

[**1.1. Original parallel strategy - No arguments 3**](#_q0w1p356sb32)

[1.1.1. Arguments 3](#_pbtsai7ol0mu)

[1.1.2. Code 3](#_33lsok8cxs9)

[1.1.3. Tareador TDG (with dependencies) 3](#_2h5l5qtfebl5)

[1.1.4. Dependence Analysis 3](#_usv8oryjy9gy)

[1.1.5. Tareador TDG (without ”datasharing” dependences) 3](#_izq2brd4gxjq)

[1.1.6. T ∞ Analysis 4](#_b0br1s8uuxhk)

[**1.2. Original parallel strategy (-d) 4**](#_wpfi8v6yi5jp)

[1.2.1. Arguments 4](#_yg7lorxgxy2)

[1.2.2. Code 4](#_8ofsolj4lge8)

[1.2.3. Tareador TDG (with dependences) : 5](#_nl2bj9s9u1ke)

[1.2.4. Dependence Analysis 5](#_ktzsyfybil0w)

[1.2.5. Tareador TDG (without ”datasharing” dependences): 5](#_ujgwatfixzl2)

[1.2.6. T ∞ Analysis 5](#_57q8lcv2mx6n)

[**1.3. Original parallel strategy (with -h) 6**](#_atga77kpsn13)

[1.3.1. Arguments 6](#_11k1a6kwb8bi)

[1.3.2. Code 6](#_a884evmixmhv)

[1.3.3. Tareador TDG (with dependencies) 6](#_ylmdwyozb51b)

[1.3.4. Dependence Analysis 7](#_edbr69okoc01)

[1.3.5. Tareador TDG (without ”datasharing” dependences): 7](#_px8q969twrbc)

[1.3.6. T ∞ Analysis 7](#_dfg2l0hbq6sf)

[**1.4. Finer grain parallel strategy 8**](#_d8na24tfn7re)

[1.4.1. Arguments 8](#_nta8cdo1nqsx)

[1.4.2. Code 8](#_8kp28rn4vo3d)

[1.4.3. Tareador TDG (with dependencies) 8](#_j7dgohn8r1fr)

[1.4.4. Dependence Analysis 8](#_bgl7tron891l)

[1.4.5. Tareador TDG (without ”datasharing” dependences) 8](#_yr5intzd9i67)

[1.4.6. T ∞ Analysis 9](#_mwwd8hjqfq00)

[**1.5. Column of tiles parallel strategy. 10**](#_rbo5dwulm4fo)

[1.5.1. Arguments 10](#_hzcmd57cn5ar)

[1.5.2. Code 10](#_2bc45sxpfzs)

[1.5.3. Tareador TDG (with dependences) : 10](#_d2apm6p0nn55)

[1.5.4. Dependence Analysis: 10](#_84dqi85163jw)

[1.5.5. Tareador TDG (without ”datasharing” dependences): 10](#_lbzi1z9l9dt5)

[1.5.6. T ∞ Analysis 11](#_phmqx7903hq6)

[**2. RECURSIVE TASK DECOMPOSITION ANALYSIS 12**](#_97wzjkdyc2bs)

[**2.1. Leaf recursive strategy 12**](#_wcy0d7pnqrb5)

[2.1.1. Arguments 12](#_wm09p2jpev7y)

[2.1.2. Code 12](#_pha9b6v3vihv)

[2.1.3. Tareador TDG (with dependences) 12](#_drtyqtceo51s)

[2.1.4. Dependence Analysis 12](#_iplazkt648v2)

[2.1.5. Tareador TDG (without “datasharing” dependences”) 12](#_49a9axn3o578)

[2.1.6. T ∞ Analysis 13](#_c45fbr7pkfqc)

[**2.2. Tree recursive strategy 13**](#_pb6chyaswced)

[2.2.1. Arguments 13](#_m9myvyobe5g)

[2.2.2. Code 13](#_bhxh3yg2s31f)

[2.2.3. Tareador TDG (with dependences) 14](#_po6s7eg40pe2)

[2.2.4. Dependence Analysis 14](#_2hp59yng1ywx)

[2.2.5. Tareador TDG (without “datasharing” dependences”) 14](#_rjd194il2vuk)

[2.2.6. T ∞ Analysis 15](#_hgovgxifdoh3)

[**3. SUMMARY OF THE STRATEGIES 16**](#_nhno9pp90lzn)

[3.1. Summary 16](#_aqkvmqhu2nxu)

### 

### 

### 

### 

### 

### 

### 

# **ITERATIVE TASK DECOMPOSITION ANALYSIS**

## Original parallel strategy - No arguments

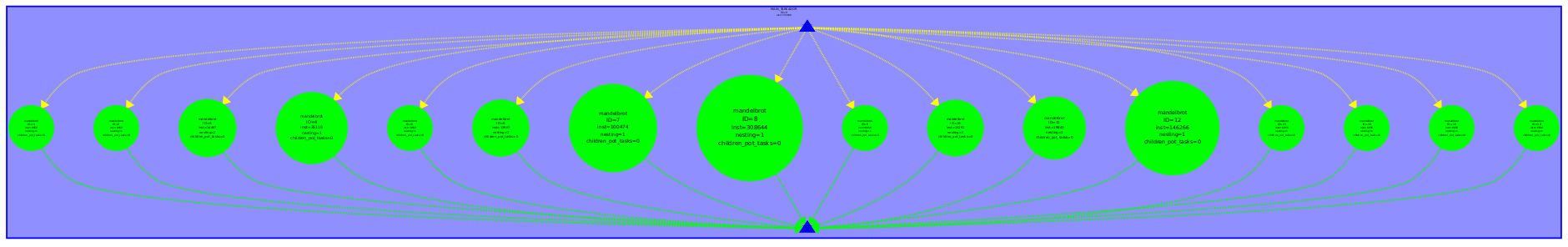
### Arguments

In this case, as requested, the code will be executed without any parameters.

### Code

mandel-seq-iter-tar.c

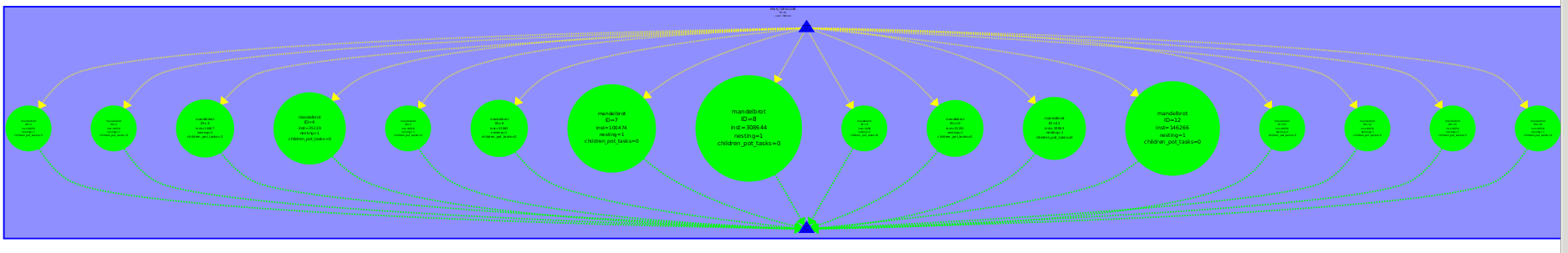
### Tareador TDG (with dependencies)



### Dependence Analysis

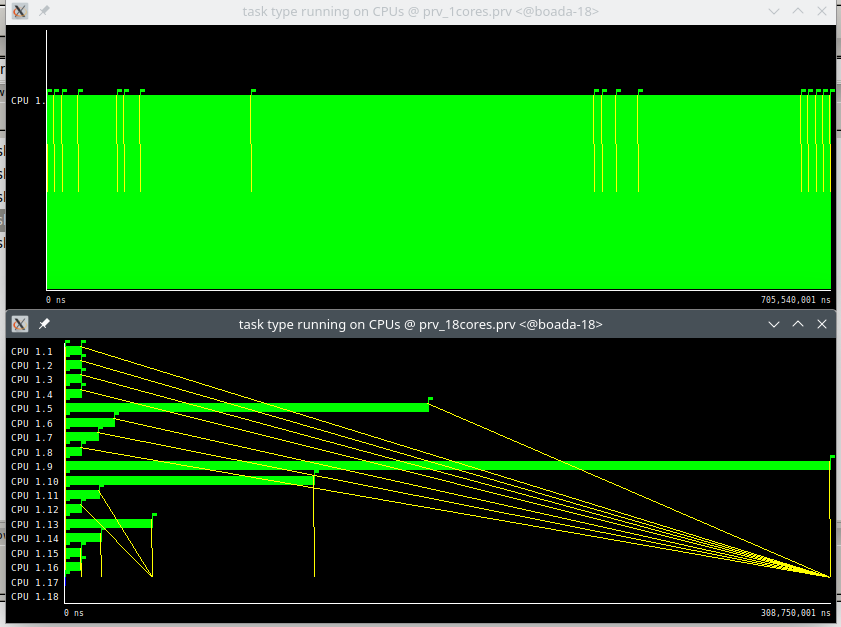
As illustrated in the preceding TDG figure, all tasks run simultaneously without any dependencies between them. Additionally, there is a noticeable imbalance in workload, particularly from 4 tasks, where the most demanding task processes 308,644 instructions, while the least intensive ones handle only 6,656.

### Tareador TDG (without ”datasharing” dependences)



In this case, the TDG is the same as before, as all tasks run simultaneously without any dependencies between them, so no changes were made to the code.

### T ∞ Analysis



T1 = 705.540.001 ns

T∞ = 308.750.001 ns

Each thread executes one task, and it is noticed that the critical path is the largest task. The load unbalancing can be observed as well with this, because of the tiles that are closer to the border subregion. As each tile is a task, the tiles that are harder to compute are still individual tasks, which causes one thread (CPU 9) having much more work than the other ones. Also, comment that from 128 threads only 17 threads are needed for full parallelism, because only this number of threads execute a task.

## Original parallel strategy (-d)

### Arguments

In this case, as requested, the code will be executed with the “**-d**“argument.

### Code

**With dependencies**: mandel-seq-iter-tar.c

**Without dependencies**: mandel-seq-iter-tar(-d).c

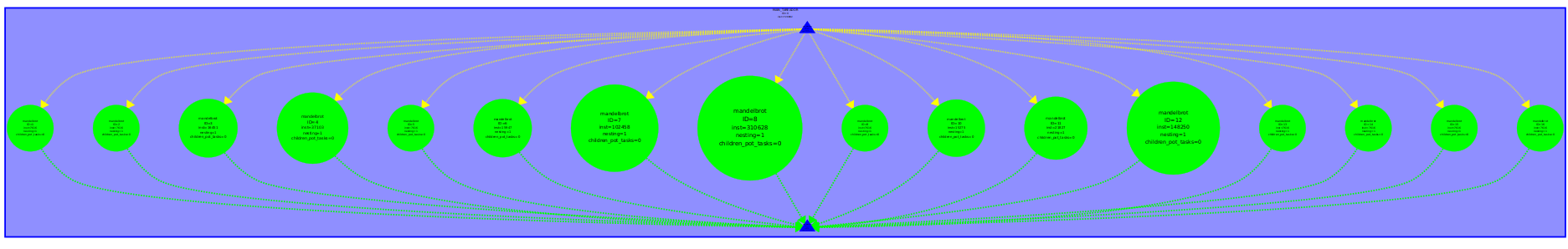
### Tareador TDG (with dependences) :

\*\*The TDG is rotated 90º

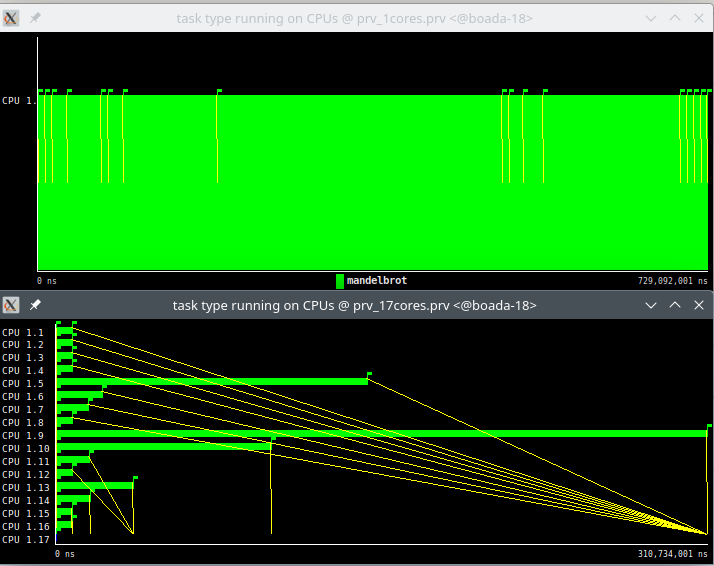
### Dependence Analysis

The tasks before the changes were executed sequentially, so to avoid that, we added **tareador\_disable\_object(&X11\_COLOR\_fake)** right before the task creation, since the variable X11\_COLOR\_fake was utilized in every task, so we removed this dependency to enable parallel execution, as illustrated in the images below.

### Tareador TDG (without ”datasharing” dependences):



### T ∞ Analysis



T1 = 729.092.001 ns

T∞ = 310.734.001 ns

The load unbalancing can be observed because of the tiles that are closer to the border subregion. As each tile is a task, the tiles that are harder to compute are still individual tasks, which causes one thread (CPU 9) having much more work than the other ones. Also, comment that from 128 threads only 17 threads are needed for full parallelism, because only this number of threads execute a task.

## Original parallel strategy (with -h)

### Arguments

In this case, as requested, the code will be executed with the “**-h**“argument.

### Code

**With dependencies**: mandel-seq-iter-tar.c

**Without dependencies**: mandel-seq-iter-tar(-h).c

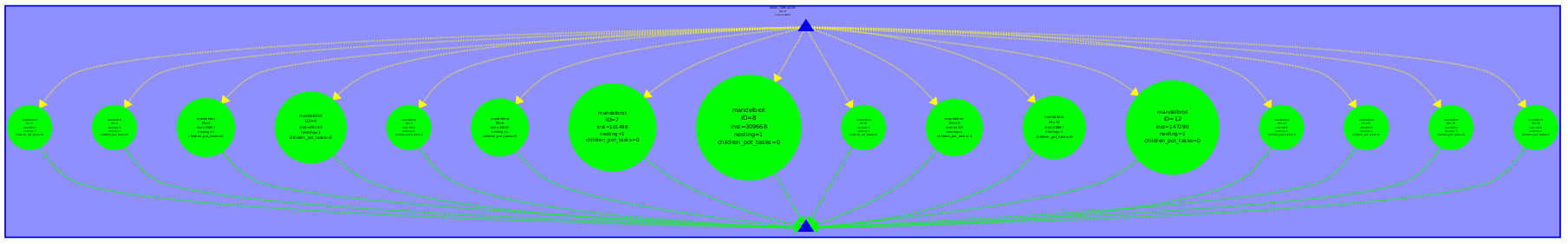
### Tareador TDG (with dependencies)



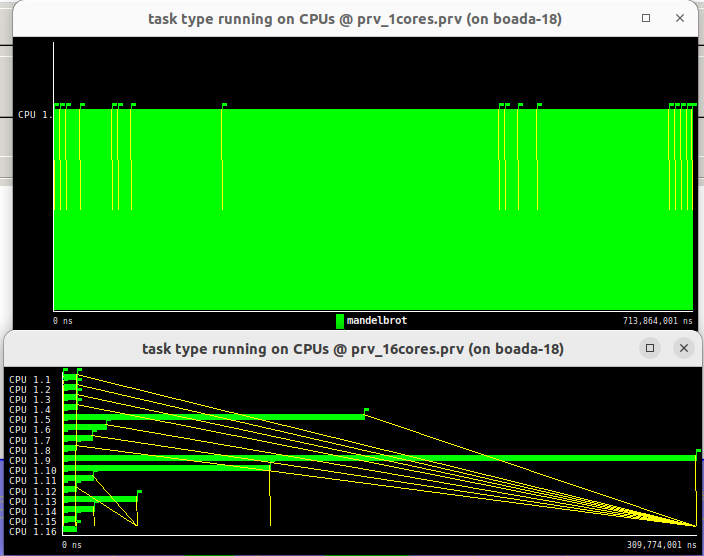
### Dependence Analysis

In Tareador, the Task Dependence Graph (TDG) shows that all nodes share the same dependency: the histogram. To mitigate these dependencies, we inserted **tareador\_disable\_object(histogram)** just before initiating the task, and we added **tareador\_enable\_object(histogram)** immediately after completing the task. This allows us to take advantage of parallelism effectively.

### Tareador TDG (without ”datasharing” dependences):



### T ∞ Analysis



T1 = 713.864.001 ns

T∞ = 389.737.001 ns

The load unbalancing can be observed because of the tiles that are closer to the border subregion. As each tile is a task, the tiles that are harder to compute are still individual tasks, which causes one thread (CPU 9) having much more work than the other ones. Also, comment that from 128 threads only 16 threads are needed for full parallelism, because only this number of threads execute a task.

## Finer grain parallel strategy

### Arguments

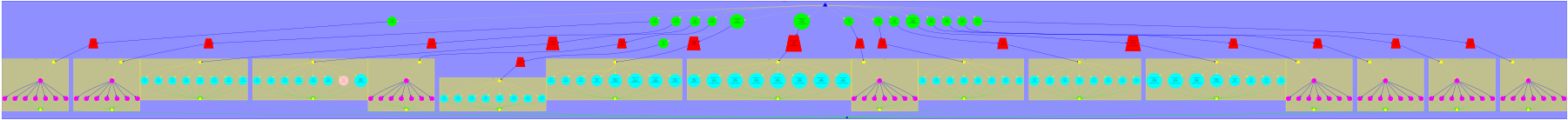
In this case, as requested, the code will be executed without any parameters.

### Code

**With dependencies**: mandel-seq-iter-tar\_finergrain.c

**Without dependencies**: mandel-seq-iter-tar\_finergrain\_nodep.c

### Tareador TDG (with dependencies)



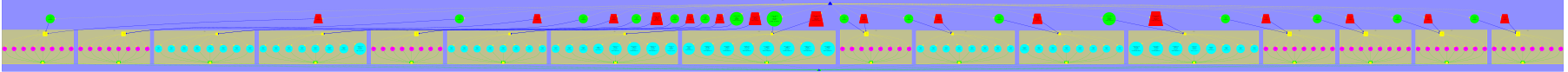
### Dependence Analysis

The first dependency observed in the TDG is the shared equal variable between the vertical and horizontal tasks. To resolve this, we perform a manual reduction.

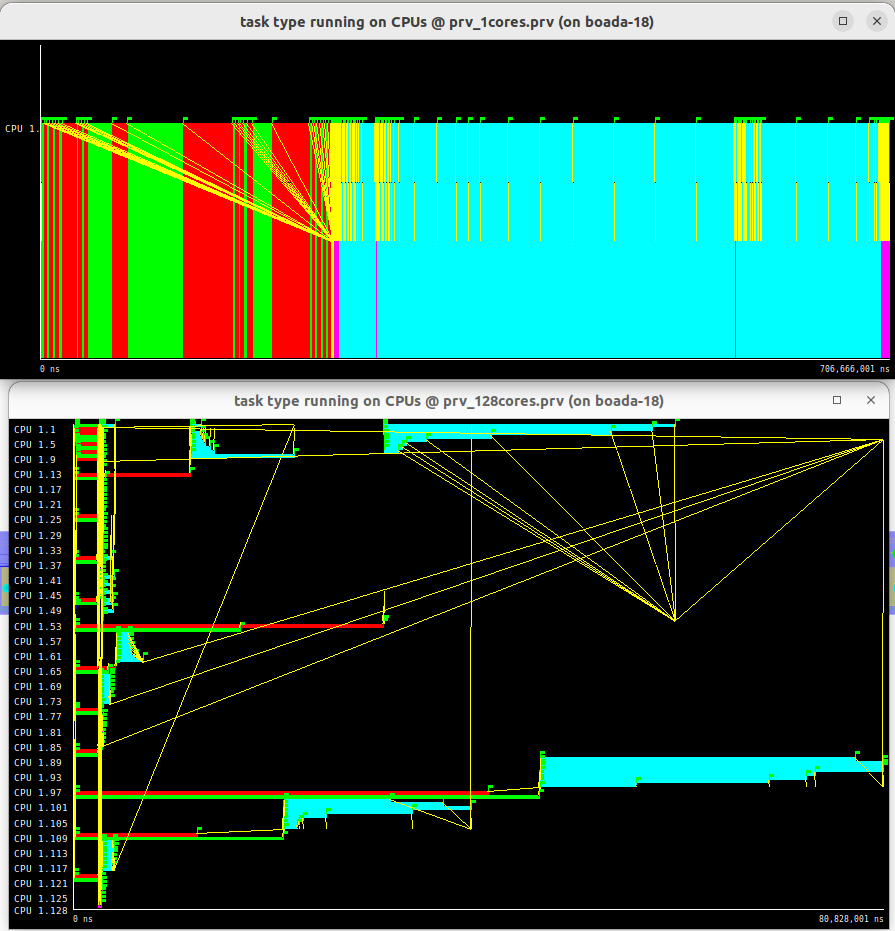
To achieve this, we define two new variables: **vertical\_equal** and **horizontal\_equal**. During execution, we use these variables instead of the global equal. Then, just before running the if-else task, we update equal as **equal = vertical\_equal & horizontal\_equal**.

The second dependency appears in the equal-py task. This occurs because the task must first access M[y][x] to begin looping. Since this value remains constant throughout the px loop, we can disable the dependency on M[y][x]. By doing so, all executions can run in parallel without requiring a prior iteration.

### Tareador TDG (without ”datasharing” dependences)



### T ∞ Analysis



T1 = 706.666.001 ns

T∞ = 80.828.001 ns

The load unbalancing can be observed because finer grain task decomposition divides better the work between tasks. The reason for unbalance here is the same as the original strategy, where CPU 93 is the last one in finishing. Also all 128 threads are used to simulate the infinite processors, however, in practice this can not be checked, and the real T∞ is approximated with T128.

## Column of tiles parallel strategy.

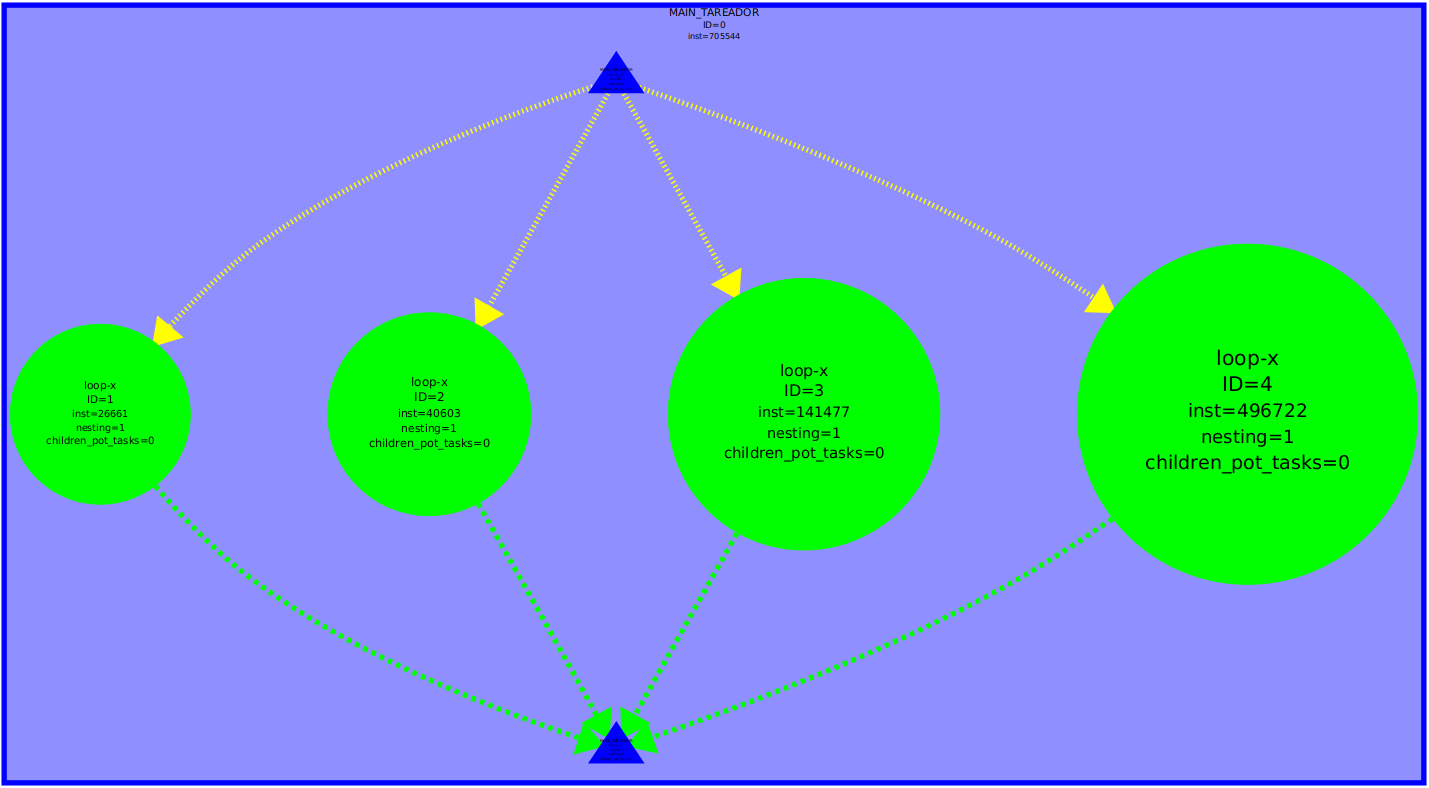
### Arguments

In this case, as requested, the code will be executed without any parameters.

### Code

mandel-seq-iter-tar\_column.c

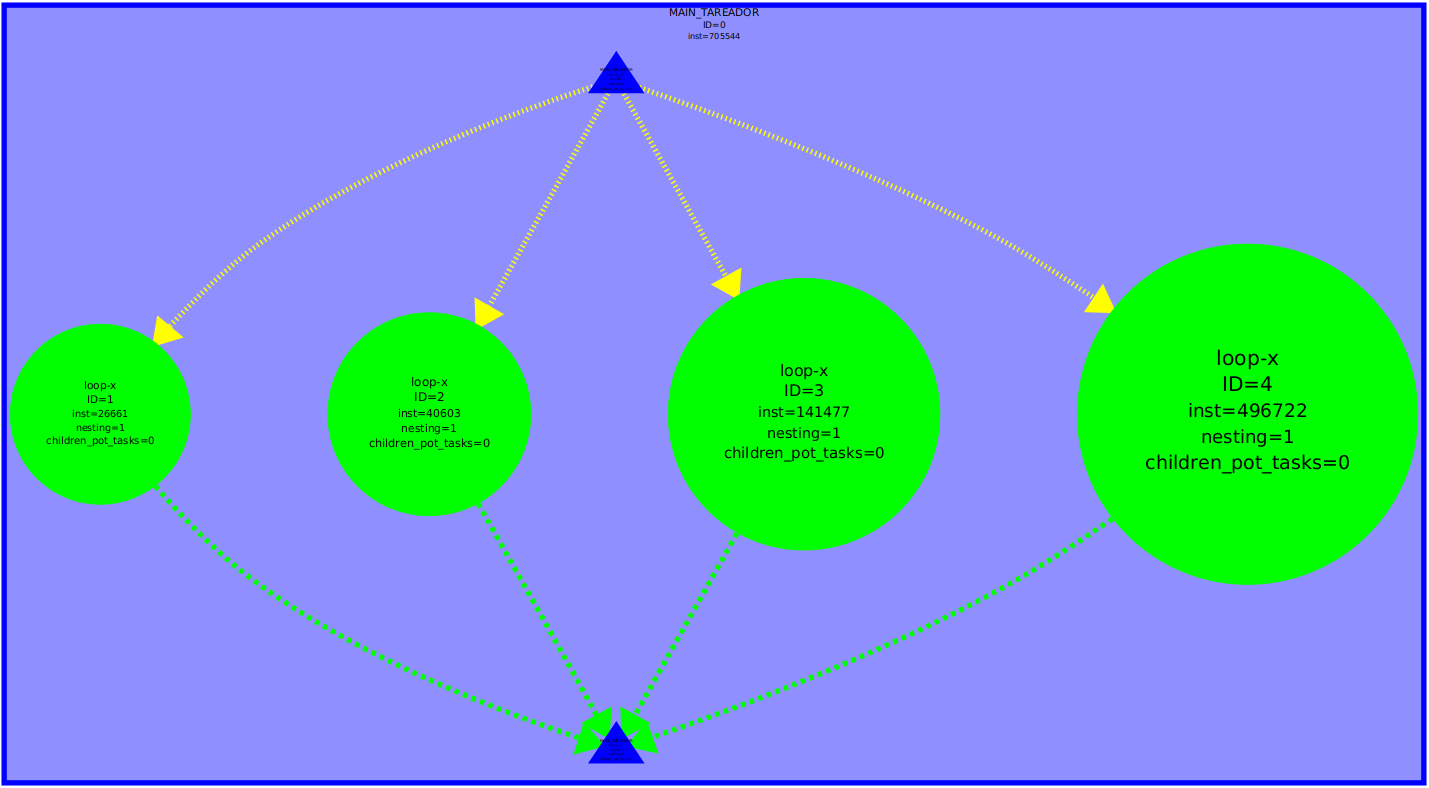
### Tareador TDG (with dependences) :



### Dependence Analysis:

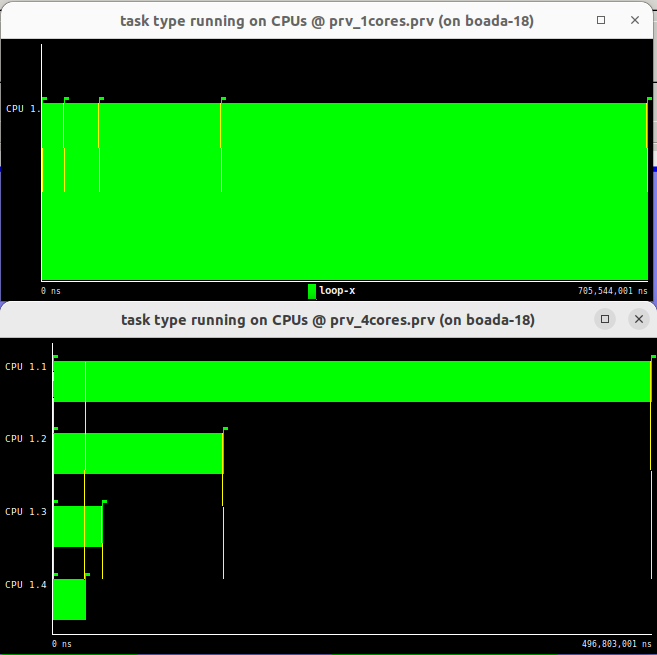
When we swap the loops, and as observed in the TDG, there are no data dependencies. However, there is a significant load imbalance. Thread 1.1 handles most of the workload, while threads 1.2, 1.3, and 1.4 have considerably less work assigned.

### Tareador TDG (without ”datasharing” dependences):



In this case, the TDG is the same as before, as all tasks run simultaneously without any dependencies between them, so no changes were made to the code.

### T ∞ Analysis



T1 = 705.544.001 ns

T∞ = 496.803.001 ns

The load unbalancing can be observed because this is the worst task decomposition, as there is almost no task decomposition and CPU 1 executes the majority of the time. Each tasks computes a whole block of 8x8 block, computing all the points in that region. The density of iteration increases depending on what subregion you compute. Also, comment that from 128 threads only 4 threads are needed for full parallelism.

# **RECURSIVE TASK DECOMPOSITION ANALYSIS**

## Leaf recursive strategy

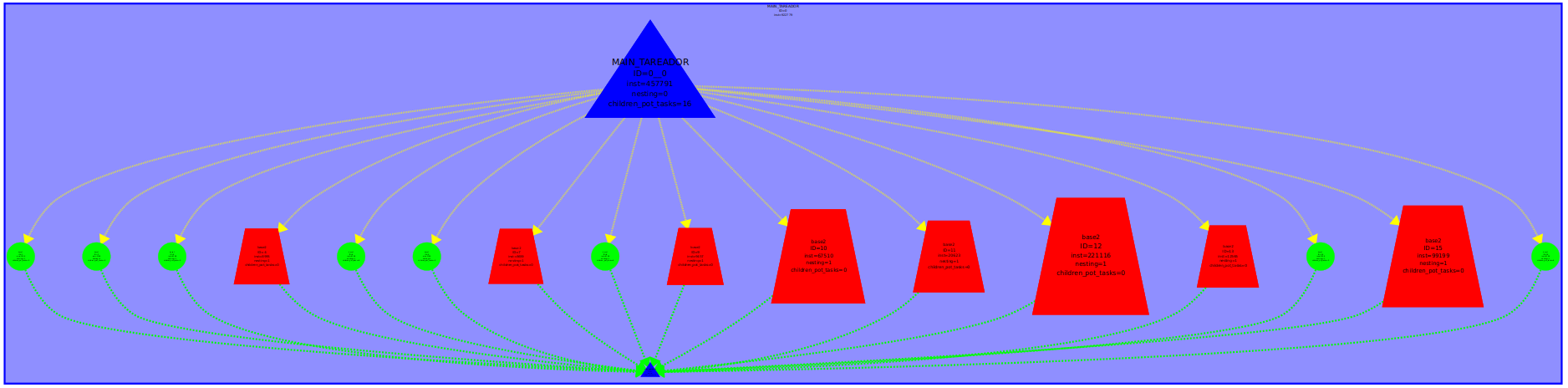
### Arguments

In this case, as requested, the code will be executed without any parameters.

### Code

mandel-seq-rec-tar\_leaf.c

### Tareador TDG (with dependences)

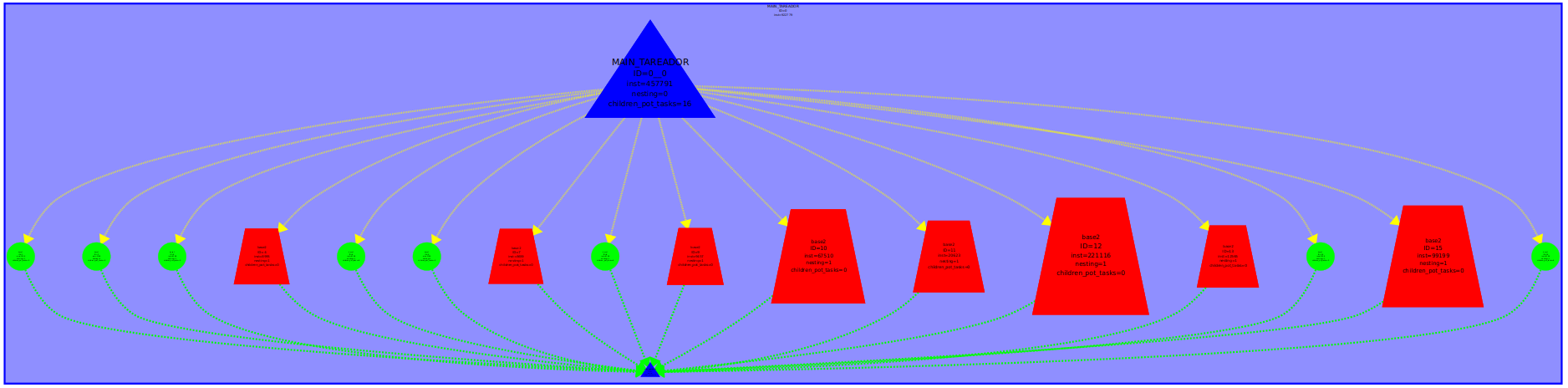


### Dependence Analysis

There are no dependences, as all tasks created at the base cases.

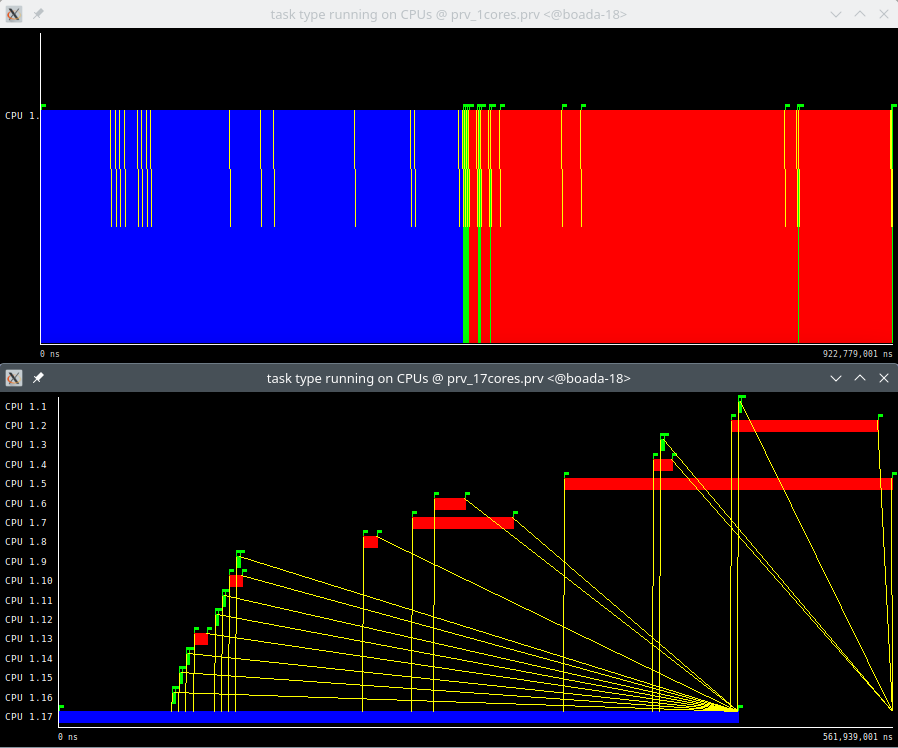
base1 for filling a tile with the same value case and base2 for computation case) are executed parallelly.

### Tareador TDG (without “datasharing” dependences”)



In this case, the TDG is the same as before, as all tasks run simultaneously without any dependencies between them, so no changes were made to the code.

### T ∞ Analysis



T1 = 922.779.001 ns

T∞ = 561.939.001 ns

It can be seen that the base case is more work loaded than the other cases (the pixel\_dwell function and creation of the other tasks). This can be seen for CPU 17. Also, comment that from 128 threads only 17 threads are needed for full parallelism, because only this number of threads execute a task.

## Tree recursive strategy

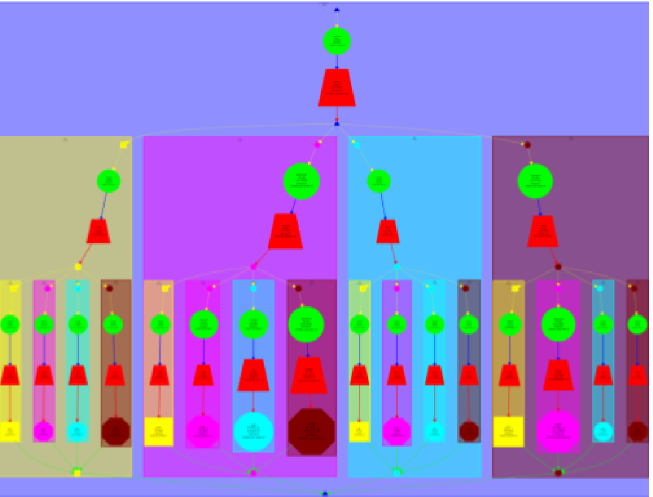
### Arguments

In this case, as requested, the code will be executed without any parameters.

### Code

mandel-seq-rec-tar\_tree.c

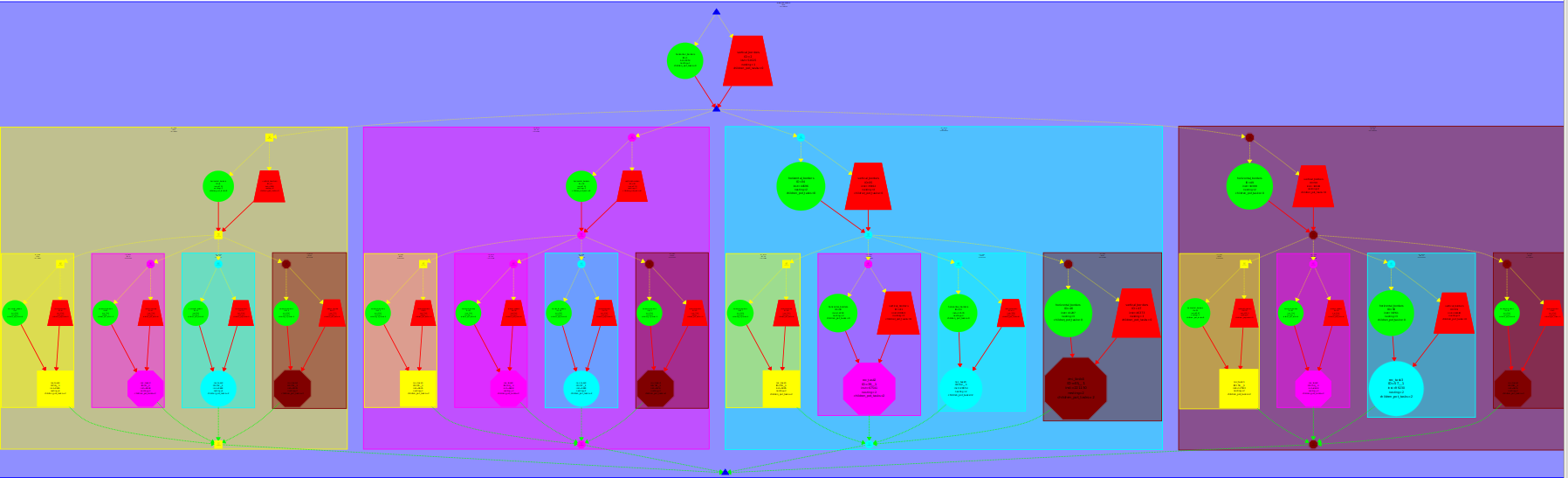
### Tareador TDG (with dependences)



### Dependence Analysis

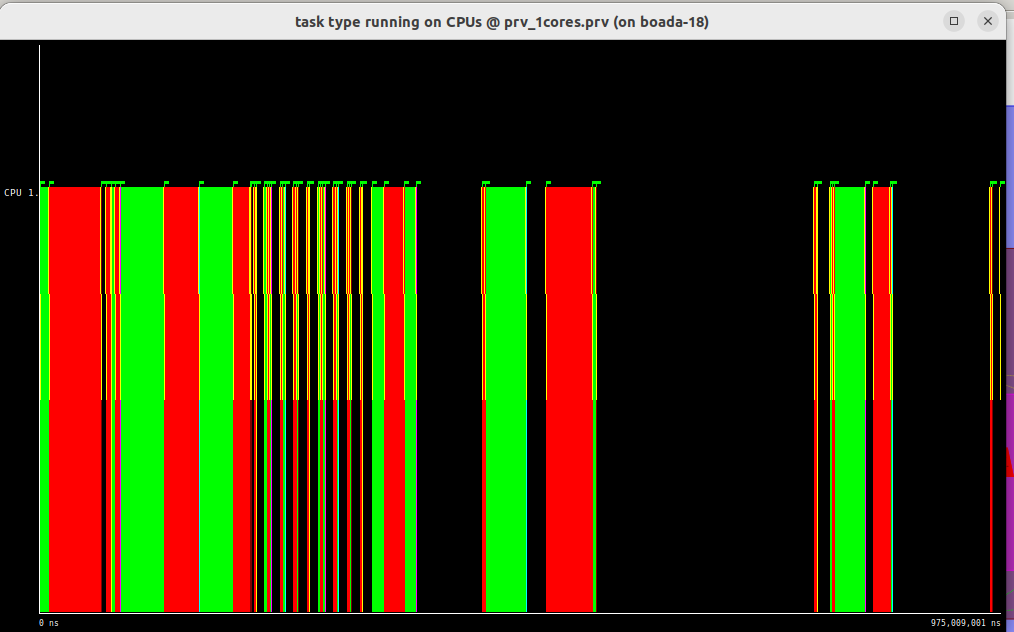
In this case there are also no data dependencies. The creation of each task is the only dependence, so thye code runs completely in parallel.

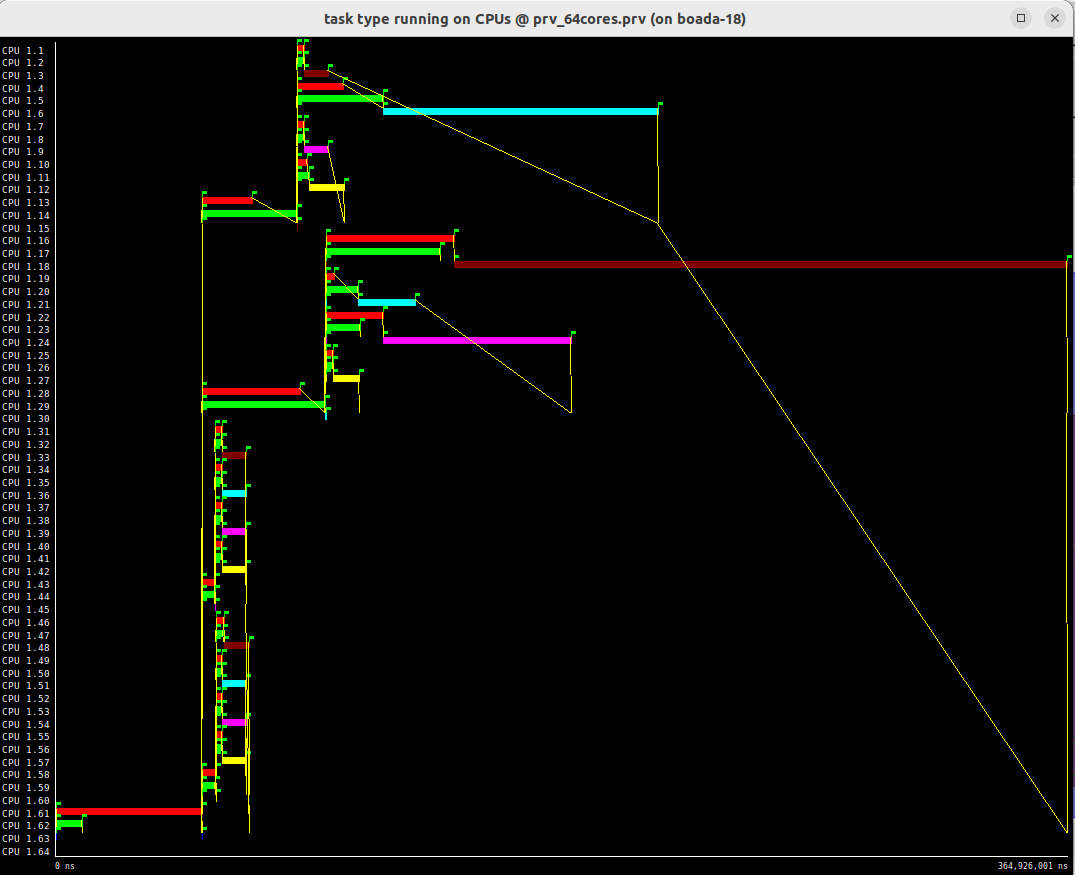
### Tareador TDG (without “datasharing” dependences”)



In this case, the TDG is the same as before, as all tasks run simultaneously without any dependencies between them, so no changes were made to the code.

### T ∞ Analysis





T1 = 975.009.001 ns

T inf = 364.926.001 ns

Small work unbalance due to the difference in regions of computations of the mandelbrot set is observed. The biggest task is executed in CPU 6, and it supposes almost ⅔ of the whole execution time. Also, comment that from 128 threads only 63 threads are needed for achieving full parallelism, because only this number of threads execute a task.

# **SUMMARY OF THE STRATEGIES**

## Summary

| **Task Decomposition** | **Strategy** | **Run Arguments** | **T₁ (ns)** | **T∞ (ns)** | **Parallelism** | **Load Unbalance (Yes/No) Why?** |
| --- | --- | --- | --- | --- | --- | --- |
| **Iterative** | Original | — | 705.540.001 | 308.750.001 | 2,29 | Yes, because of the tiles that are closer to the border subregion. As each tile is a task, the tiles that are harder to compute are still individual tasks, which causes one thread having much more work than the other ones. |
| Original | -d | 729.092.001 | 310.734.001 | 2,34 | Yes, because of the tiles that are closer to the border subregion. As each tile is a task, the tiles that are harder to compute are still individual tasks, which causes one thread having much more work than the other ones. |
| Original | -h | 713.864.001 | 389.737.001 | 1,83 | Yes, because of the tiles that are closer to the border subregion. As each tile is a task, the tiles that are harder to compute are still individual tasks, which causes one thread having much more work than the other ones. |
| Finer grain | — | 706.666.001 | 80.828.001 | 8,74 | Yes. However, finer grain task decomposition divides better the work between tasks. The reason for unbalance here is the same as the original strategy. |
| Column | — | 705.544.001 | 496.803.001 | 1,42 | Yes. This is the worst task decomposition, as there is almost no task decomposition. Each tasks computes a whole block of 8x8 block, computing all the points in that region.  The density of iteration increases depending on what subregion you compute. |
| **Recursive** | Leaf | — | 922.779.001 | 561.939.001 | 1,64 | Yes. The base case is more  work loaded than the other cases (the pixel\_dwell function). |
| Tree | — | 975.009.001 | 364.926.001 | 2,67 | Yes, small work unbalance deu to the difference in regions of computations of the mandelbrot set. |
| **Best Task Decomposition** | **Strategy** | **Reason why** | | | | |
| Iterative fine grain | Finer grain strategy is the best one, as the work is splitted reducing the size of each task creating more of them in the inner code. One task for the horizontal checking and another for the vertical checking helps defining this finer granularity. | | | | |

## 