.out -> no obrir, es per comparar després amb el diff

2.1 (fixar-nos lab1) a nivell de row(fila) -> ficar start i end

després fer el mateix a nivell de punt (+ baix)

**1.2**

“In the previous laboratory assignment you should have learned about the different options in OpenMP, and when to use them, to express tasks out of loops (either with implicit tasks or explicit tasks with task and taskloop), with the appropriate thread creation (parallel and single) and how to enforce task order with task barriers (taskwait and taskgroup), and data sharing constraints (critical, atomic and reduction operations). In addition you should remember from the first assignment how to use the Tareador API and GUI to understand the potential parallelism available in a sequential code, as well as the causes that limit this parallelism. And also the use of modelfactors and Paraver to visualise the execution of your parallel OpenMP program and understand its performance.”

single: només ho fa un

atomic: no pot protegir un tros de codi (protegir una zona de mem. esc/lec)

critical: només ho fa un a la vegada

Per protegir la part del -d amb el critical: dins el if -> 3 opcions

*//execució normal*

*output[row][col]=k;*

*//execució amb el -h*

*if (output2histogram) histogram[k-1]++;*

*//execució amb el -d*

*if (output2display) {*

*/\* Scale color and display point \*/*

*//ficar critical?*

*long color = (long) ((k-1) \* scale\_color) + min\_color;*

*//ficar critical?*

*if (setup\_return == EXIT\_SUCCESS) {*

*//ficar critical?*

*XSetForeground (display, gc, color);*

*XDrawPoint (display, win, gc, col, row);*

*}*

*}*

Els 3 grafs tenen una característica en comú.

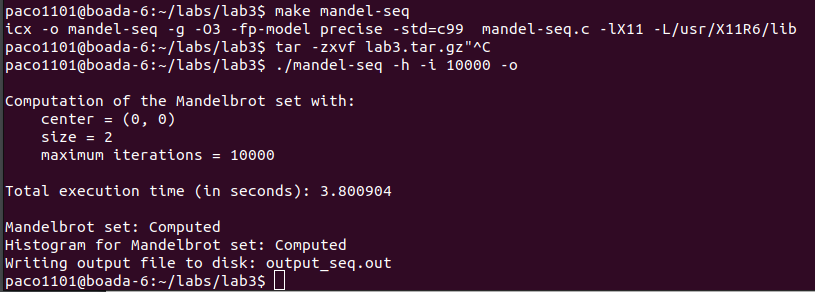
Si executem el codi tal com ens el donen amb OpenMP no ens donarà bé

Objectiu:

Fer sense opcions, ‘-d’ i ‘-h’ a nivell de row i a nivell de point

**2.1. The Mandelbrot set**

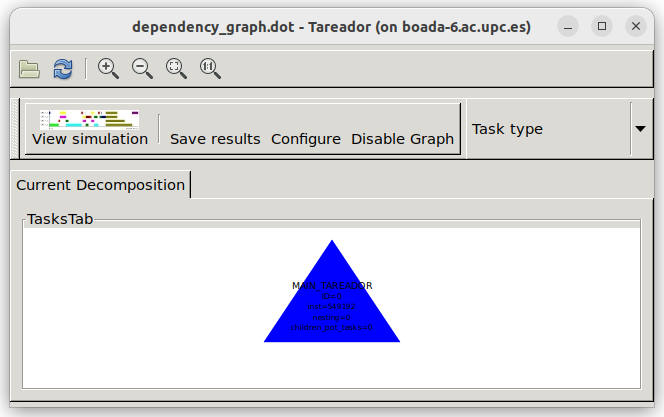
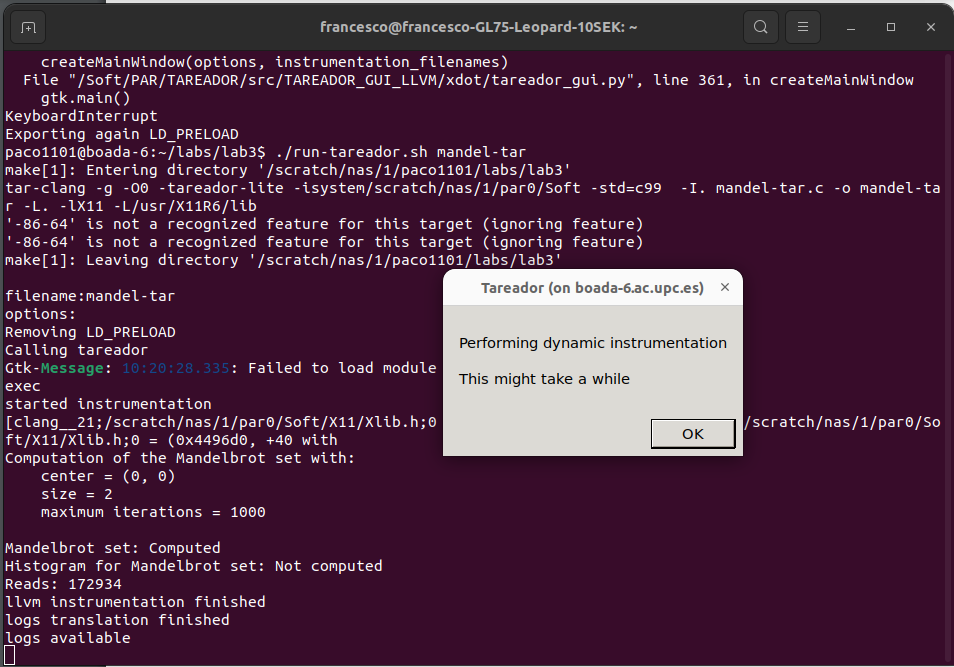
4. In order to have a reference of the sequential execution, execute the sequential version with "./mandel-seq -h -i 10000 -o" to get its execution time and the output file, to be used later to check the correctness of the different parallel versions you will write.



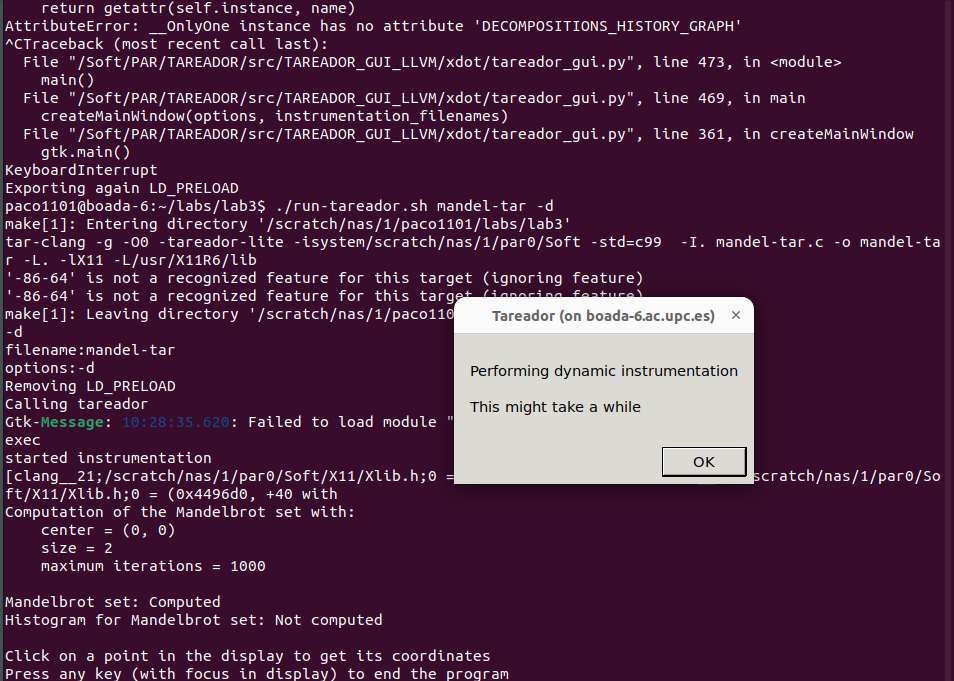
**2.2. Task decomposition analysis with Tareador**

1. Complete the sequential mandel-tar.c code partially instrumented in order to analyse the potential parallelism for the Row strategy, with granularity of one iteration of the row loop per task. Then compile the instrumented code using the appropriate target in the Makefile that generate the binary for analysis with Tareador. Note: tareador ON() and tareador OFF() calls are already done in the main program.

2. Interactively execute mandel-tar binary using the ./run-tareador.sh script, only indicating the name of the instrumented binary (with no additional options). The size of the image to compute is defined inside the script (we are using -w 8 as the size for the Mandelbrot image in order to generate a reasonable task graph in a reasonable execution time). For the deliverable: Which are the two most important characteristics for the task graph that is generated? Save the TDG generated for later inclusion in the deliverable.



3. Next interactively execute mandel-tar binary using the ./run-tareador.sh, but now indicating the name of the instrumented binary and the -d option. Important: look for a small window that will be opened to display the tiny Mandelbrot set that is computed and close it in order to finish its execution, or to be more efficient, press the Enter/Return key in your keyboard immediately after launching the execution. For the deliverable: Which are the two most important characteristics for the task graph that is generated? Which part of the code is making the big difference with the previous case? How will you protect this section of code in the parallel OpenMP code that you will program in the next sessions? Save the new TDG generated for later inclusion in the deliverable.



ficar task(paral·lelisme) després del loop

crear un .out per cada codi i despres compararlo