3r Deliverable

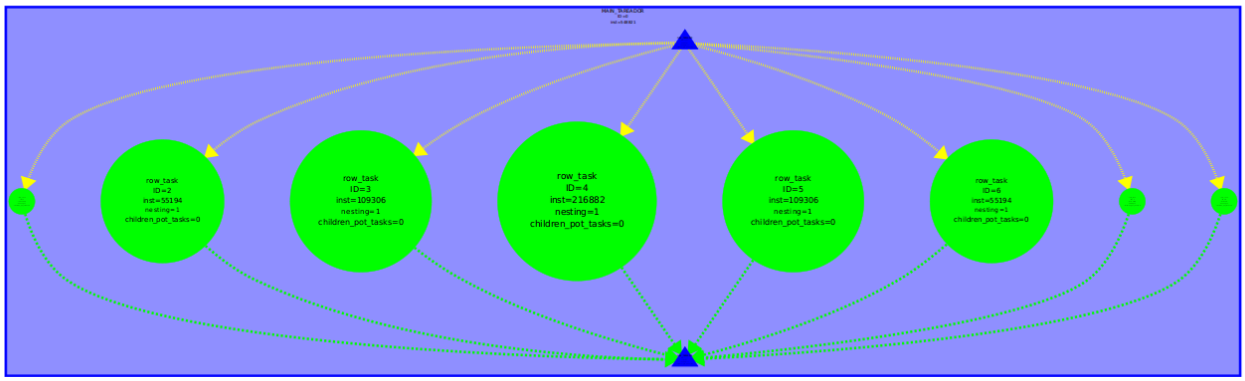
PAR2110

Eric Casanovas

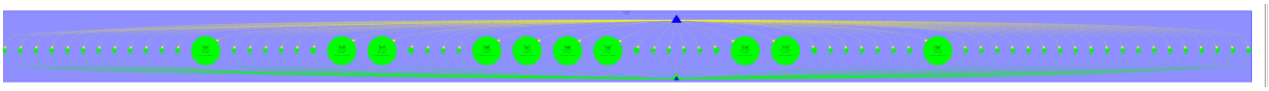
24/04/2018

6.1 Task granularity analysis

1. Which are the two most important common characteristics of the task graphs generated for the two task granularities (Row and Point) for the non-graphical version of mandel-tareador? Obtain the task graphs that are generated in both cases for -w 8.



Tareador-row



Tareador-point

Les dues característiques principals que observem són:

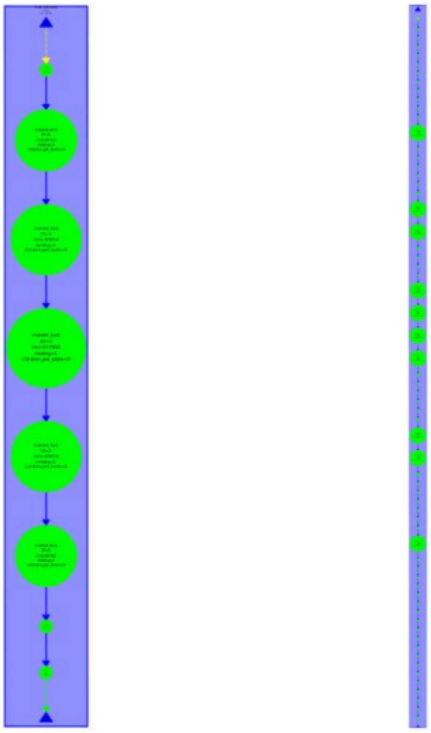
- No hi ha dependencies entre les tasques ni en el point ni en el row.
- Les tasques no tenen la mateixa granularitat ni en el row ni en el point.

2. Which section of the code is causing the serialization of all tasks in mandeld-tareador? How do you plan to protect this section of code in the parallel OpenMP code?

```
#if _DISPLAY_
            /* Scale color and display point  */
            long color = (long) ((k-1) * scale_color) + min_color;
            if (setup_return == EXIT_SUCCESS) {
                XSetForeground (display, gc, color);
                XDrawPoint (display, win, gc, col, row);
            }
#else
```

Aquest fragment de codi es el que esta provocant que es facin en serie i tinguin dependencies entre elles les tasques.

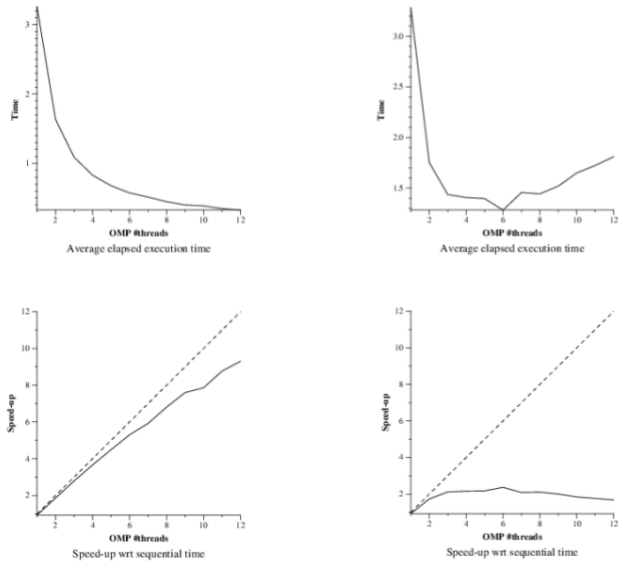
El podem protegir amb un #pragma omp critical per a fer que així únicament pugui entrar un thread i així evitar la mala representació del «Mendelbrot set» i per tant el mal funcionament del programa



Dreta: row, esquerra: point

6.2 OpenMP task–based parallelization

1. For the Row and Point decompositions of the non-graphical version, include the execution time and speed–up plots obtained in the strong scalability analysis (with -i 10000). Reason about the causes of good or bad performance in each case.

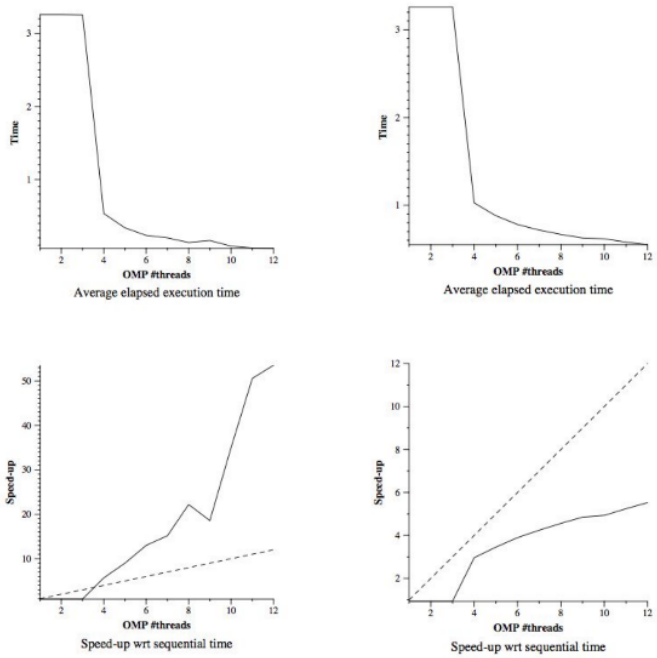


Dreta: row, esquerra: point

Observem que l'speedup del row es bastant superior al de point degut a que el point té el problema de que crearem un munt de tasques (de l'ordre de files*columnes de la matriu), cosa que farà que tinguem un overhead enorme i això repercuteixi moltissim en el rendiment del programa. Mentre que la de row al crear menys tasques observem que l'overhead no es tan gran i seria més rentable fer-ho així.

6.3 OpenMP taskloop–based parallelization

1. For the Row and Point decompositions of the non-graphical version, include the execution time and speed–up plots obtained in the strong scalability analysis (with -i 10000). Reason about the causes of good or bad performance in each case.

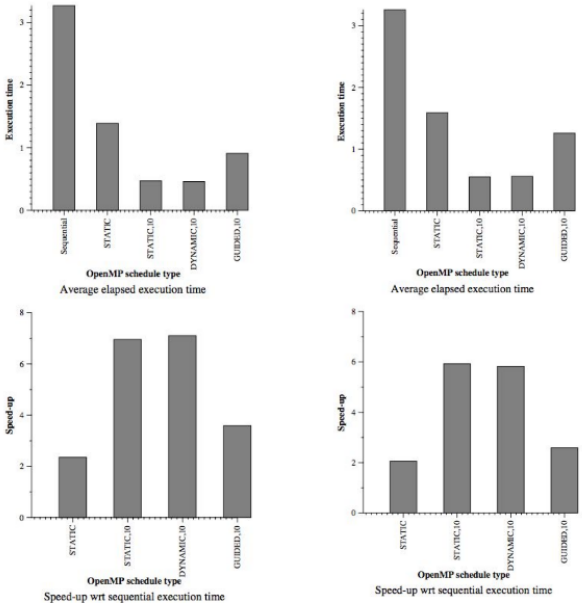


Esquerra: row, dreta: point

Com podem observar amb el taskloop tenim els mateixos problemes que amb el task. Observem que per rows te un speedup considerable, però per point no degut al enorme overhead que es produeix per fer tantes tasques.

6.4 OpenMP for–based parallelization

1. For the the Row and Point decompositions of the non-graphical version, include the execution time and speed–up plots that have been obtained for the 4 different loop schedules when using 8 threads (with -i 10000). Reason about the performance that is observed.



Esquerra: row, dreta: point

Observem que no hi ha gaire diferencia entre l'estrategia row i la point degut a que no cal sincronització entre els diferents threads, que és el que ens molestava al point degut a la creació de moltissimes tasks.

2. For the Row parallelization strategy, complete the following table with the information extracted from the Extrae instrumented executions (with 8 threads and -i 10000) and analysis with Paraver, reasoning about the results that are obtained.

	Static	Static, 10	Dynamic, 10	Guided, 10
Running average time per thread	446.231,13 µs	465.644,61 µs	473.839,19 µs	467.655,9 µs
Execution unbalance (average time divided by maximum time)	0,32	0,79	0,64	0,39
SchedForkJoin (average time per thread or time if only one does	990.306,76 µs	118.575,121 µs	54.449,35 µs	484.006,71 µs

Podem observar que el dynamic es el més ràpid degut a la seva planificació i el poc overhead que té. També caldria destacar que el guided és més lent degut al overhead tan gran que té al igual que el static. També podem veure com en blocs de 10 iteracions es més ràpid amb la planificació static degut a que al haver tantes iteracions si anem assignant a un thread en blocs de 10 en 10 serà més eficient que de 1 en 1. Per ultim cal remarcar que el static,10 és més ràpid que el guided degut al overhead causat pel guided.