PARAL·LELISME

Entrega lab1

Autors:
Sergi Soriano
Mingjian Chen

Grup: 21

5.1

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? Include the task graphs that are generated in both cases for -w 8.

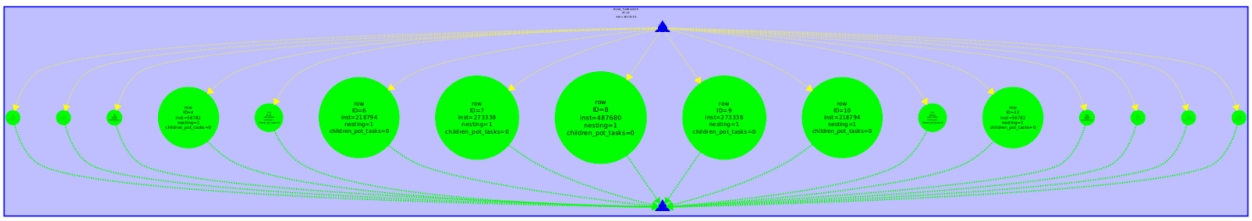


Figure 1: Traça del mandel-tareador-row

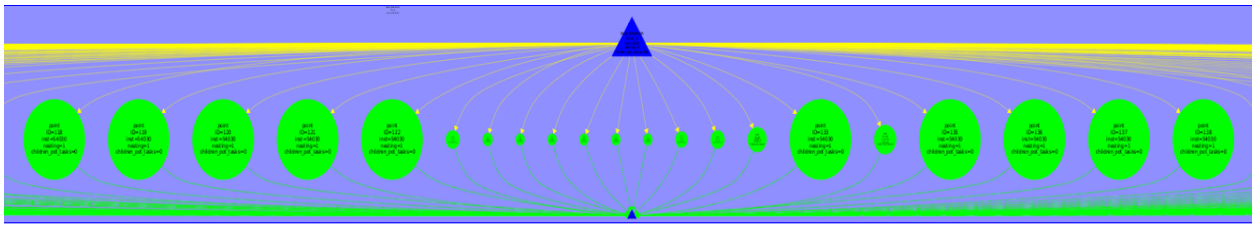


Figure 2: Traça ampliada del mandel-tareador-point

Característiques:
1) No hi ha dependència de dades entre tasques, estàn totalment paralitzades.
2) Hi ha tasques que gasten més temps que altres, les tasques no estàn homogenitzades.

La diferencia més notable de les dues traces és que en el cas del mandel-omp-point el nombre de tasques creades és molt superior al nombre de tasques que es creen en el mandel-omp-row

2. Which section of the code is causing the serialization of all tasks in mandeld-tareador? How have you protected this section of code in the parallel OpenMP code?

```
if (setup_return == EXIT_SUCCESS) {
      XSetForeground (display, gc, color);
      XDrawPoint (display, win, gc, col, row);
}
```

Per protegir la secció del codi quan executem en paralel posarem un:

#pragma omp critical

fent que el codi quedi de la següent forma:

```
#pragma omp critical
{
if (setup_return == EXIT_SUCCESS) {
      XSetForeground (display, gc, color);
      XDrawPoint (display, win, gc, col, row);
}
}
```

3. Using the results obtained from the simulated parallel execution for mandel-tareador and for a size of -w 16, complete the following table with the execution time and speed-up (with respect to the execution with 1 processor) obtained for the non-graphical version, for each task granularity. Comment the results highlighting the reason for the poor scalability.

	Row		**Point**	
#Procesadors	**Exec. Time**	**Speed-up**	**Exec. Time**	**Speed-up**
1	1.612.310.001	1	1.612.310.001	1
2	824.616.001	1.955	829.142.001	1.94
4	495.593.001	3.25	441.421.001	3.65
8	487.860.001	3.92	222.634.001	7.24
16	487.778.001	3.30	113.672.001	14.18

El temps està calculat en ns.

La versió point és notablement més ràpida que la versió row. Imaginem que es dóna aquest cas ja que en la versió point hi ha moltes més tasques a executar i per tant els threads estan sempre treballant mentre el master thread s'està executant.

Contrariament, a la versió row els threads encarregats d'executar les tasques estan molta estona esperant a poder executar-ne una, ja que es tarda molt més en crear-se les tasques.
5.2

1. Include the relevant portion of the codes that implement the task-based parallelization for the Row and Point decompositions (for the non–graphical and graphical options), commenting whatever necessary.

Versió paral·lelització en row:

El main just abans de fer la crida a la funció mandelbro escrivim:

```
#pragma omp parallel
#pragma omp single
```

Aquestes dues comandes ens serveixen per crear un pool de tasques i indiquem que nomes un thread crearà les tasques, els altres threads resoldràn les tasques que hi hagi en el pool de task per fer.

En el codi de mandelbro farem el següent:

```
for (row = 0; row < height; ++row) {
    #pragma omp task firstprivate(row) private(col)
      for (col = 0; col < width; ++col) {
```

Posarem la comanda de omp per indicar on crearem les tasques i indiquem quines variables les volem posar com a firstprivate i quina com a private. En aquest cas com les variables row i col estan definides globalment les hem de posar row com a firstprivate que vol dir que reserva un espai per la variable row ja inicialitzada i com a private col ja que encara no esta inicialitzada. Si no fessim això les variables estarian shared i provocaria problemes.

Versió paral·lelització en point:

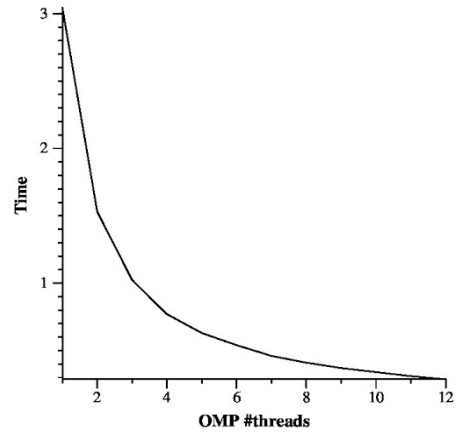
En el main mantindrem les dues linies de codi explicades abans.
En el codi de mandelbro farem el següent:

```
for (row = 0; row < height; ++row) {
     for (col = 0; col < width; ++col) {
      #pragma omp task firstprivate(row, col){
```

En aquest cas indiquem que la variables row i col com a firstprivate ja que estan inicialitzades i volem mantenir el valor que tenen. El motiu d'haver de declararles, en aquest cas firstprivate, és el mateix que el cas anterior. Les variables en row i col en shared donarien problemes.
En els dos casos,el codi que hi ha a la regió on el programa crea el gràfic s'hi produeix un data sharing i l'hem de protegir aplicant una comanda de OMP:

```
#pragma omp critcal
```

2. For the 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



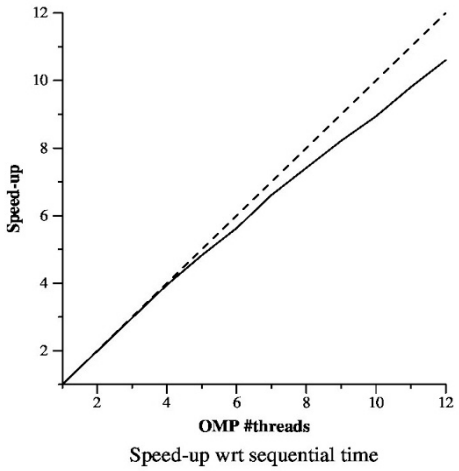
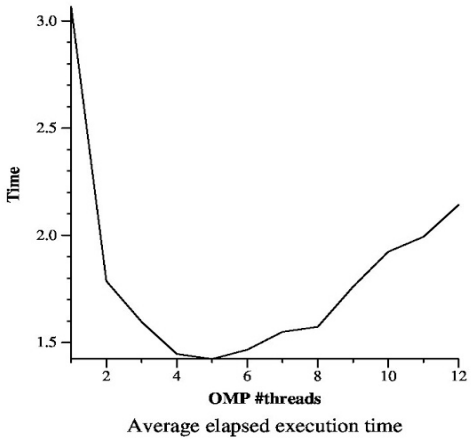


Figure 3: Gràfiques de temps i speed-up de la versió mandel-omp-row



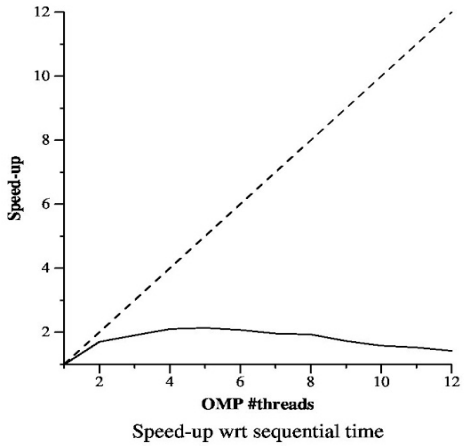


Figure 4: Gràfiques de temps i speed-up de la versió mandel-omp-point

En les gràfiques podem veure que la versió row el speed-up és molt superior al speed-up de la versió point.
Creiem que és perque la versió point el master thread va més lent creant les tasques que els threads en executar les tasques que hi ha al pool de tasques.
En canvi la versió row el master thread va més ràpid creant les tasques que els altres threads en executar-les.

5.3

1. Include the relevant portion of the codes that implement the for-based parallelization for the Row and Point decompositions (for the non–graphical and graphical options), commenting whatever necessary.

Versió row:

```
#pragma omp for schedule(runtime)
for (row = 0; row < height; ++row) {
    for (col = 0; col < width; ++col) {
```

Versió point:

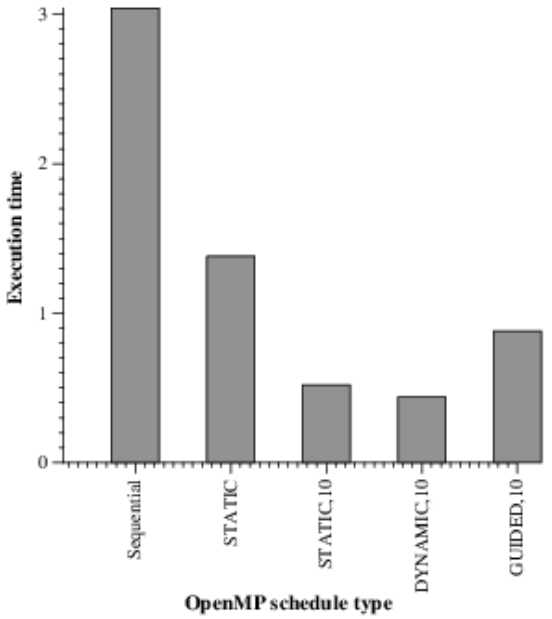
```
for (row = 0; row < height; ++row) {
    #pragma omp for schedule(runtime)
      for (col = 0; col < width; ++col) {
```

En ambdos casos el main tindrà:

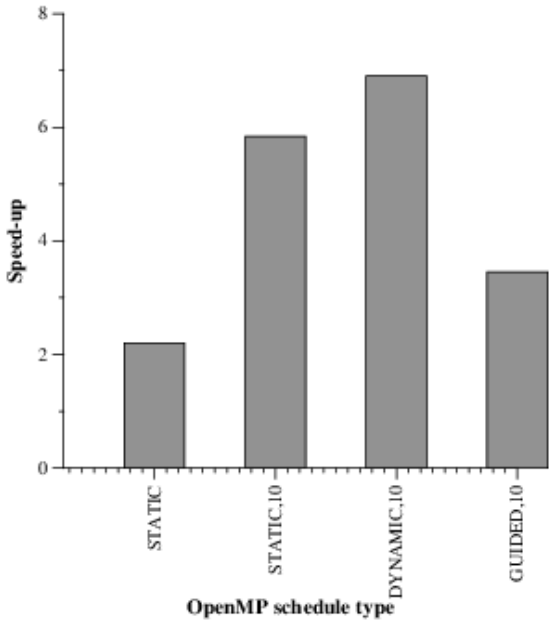
```
      #pragma omp parallel
```

La comanda de omp *#pragma omp for* fa que els threads executin el codi que hi hagi per sota d'ell però de forma que es dividiràn la feina. La forma de dividir-se la feina vindrà donat segons el tipus de *schedule* que li indiquem.
Nosaltres tenim *runtime* ja que indiquem la forma de dividir la feina a traves de la consola posant valor a la variable *OMP_SCHEDULE*.

2. 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.

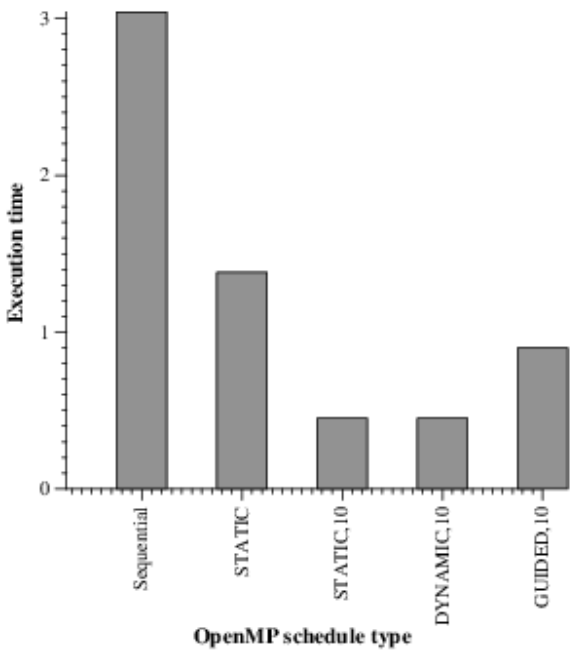


Average elapsed execution time

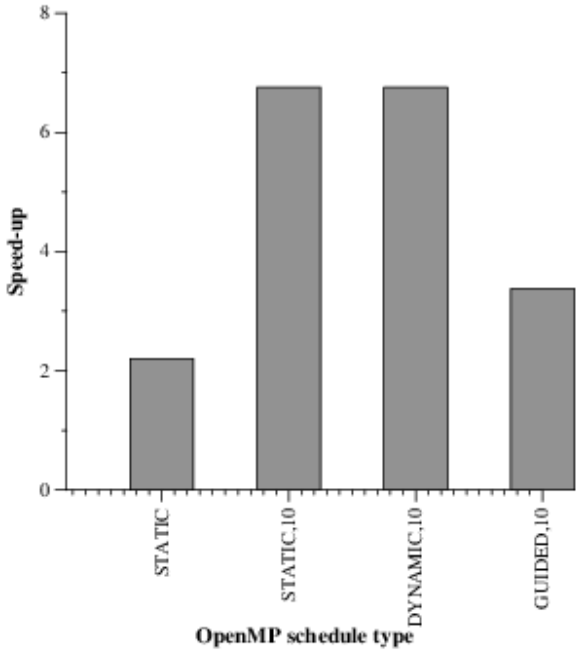


Speed-up wrt sequential execution time

Figure 5: Gràfiques de temps i speed-up de la versió mandel-omp-point amb diferents schedule



Average elapsed execution time



Speed-up wrt sequential execution time

Figure 6: Gràfiques de temps i speed-up de la versió mandel-omp-row amb diferents schedule

En les gràfiques podem veure com static,10 i dynamic,10 són les execucions més eficients tant en el cas de row com en el cas point.

3. 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	255,812,356.88	154,045,429	143,724,140.75	136,204,297
Execution unbalance (average time divided by maximum time)	0.20	0.63	0.90	0.43
SchedForkJoin (average time per thread or time if only one does)	151,619.25	153,704	165,687.75	152,695.75

Els valors estan calculats en ns.

Static: Divideix les iteracions entre els threads que tenim i va donant d'un en una iteració a cada thread per a que ho calculi.
Static,10: Divideix les iteracions entre els threads que tenim i va donant a cada thread 10 iteracions cada vegada per a calcular
Dynamic,10: El número de tasques que fa cada threads no es igual. La cpu va donant paquets de 10 tasques als threads que ja hagin acabat de fer les seves tasques.
Guided, 10: El numero de tasques que s'asigna als threads es com a minim 10. Aquest valor varia, al principi dona un conjunt de tasques més gran i va reduint-se.

Com podem veure en el gràfic el temps mitg que gasta cada thread es va reduint a mida que anem variant el tipus d'execució del parallel for.
El tipus d'execució més ràpid es el guided, 10 mentre que el static és el més lent.