ENTREGABLE PARAL·LELISME

PRIMER ENTREGABLE

Username: par4105
02 d'Octubre de 2018
Tardor del 2018

Àlex Valls
Roger Guasch

Taula de continguts

Node architecture and memory

Describe the architecture of the boada server. To accompany your description, you should refer to the following table summarising the relevant architectural characteristics of the different node types available:

Per obtenir una descripció acurada de l'arquitectura del servidor hem fet servir les comandes descrites en la Figura 1.1 . Per saber l'arquitectura dels servidors boada-5 i dels servidors boada-6 a boada-8 vam utilitzar les comandes de la Figura 1.2.

```
par4105@boada-1:~$ lstopo
par4105@boada-1:~$ lscpu
```

Figura 1.1 Comandes executades al boada-1 per determinar la seva arquitectura

```
par4105@boada-1:~$ ssh boada-6 "lstopo"
par4105@boada-1:~$ ssh boada-6 "lscpu"
```

Figura 1.2 Comandes executades al boada-6 per determinar la seva arquitectura. També serien vàlides pel boada-5 sempre i quan canviessim el secure shell de boada-6 a boada-5

	boada-1 to boada-4	boada-5	boada-6 to boada-8
Number of sockets per node	1	1	1
Number of cores per socket	6	6	8
Number of threads per core	2	2	1
Maximum core frequency	2395.0000 MHz	2600.0000 MHz	1700.0000 MHz
L1-I cache size (per-core)	32 KB	32 KB	32 KB
L1-D cache size (per-core)	32 KB	32 KB	32 KB
L2 cache size (per-core)	256 KB	256 KB	256 KB
Last-level cache size (per socket)	12288 KB	15360 KB	20480 KB
Main memory size (per socket)	12GB	31 GB	16 GB
Main memory size (per node)	12GB	31 GB	16 GB

Figura 1.3 Taula amb les dades obtingudes dels diferents servidors de boada

Com podem veure en la Figura 1.3, tots els servidors boada tenen 1 socket per node. Sobre el número de cores per socket tots tenen 6 a excepció de boada-6 a boada-8 que tenen 8 cores per socket.

En quant a la freqüència podem veure com clarament boada-5 té la més alta, seguida del boada-1 a boada-4 i per últim vindrien els servidors de boada-6 a boada-8.

Tots els servidors tenen 3 nivells de caché. Tant el primer com pel segon nivell de cache tenen el mateix tamany de 32KB i 256KB respectivament. La variació ve en l'últim nivell de

cache on es noten diferències substancials. Les dades obtingudes es poden consultar en la Figura 1.3.

La informació obtinguda amb les comandes anteriors és poc gràfica i pot resultar complicat comparar els diferents servidors. Aquest petit problema es pot solucionar executant les comandes de la Figura 1.4 per obtenir un esquema gràfic, expressat de l'arquitectura de cada servidor diferent.

```
par4105@boada-1:~$ lstopo --of fig map.fig
par4105@boada-1:~$ ssh boada-5 "lstopo --of fig map.fig"
par4105@boada-1:~$ ssh boada-6 "lstopo --of fig map.fig"
```

Figura 1.4 Comandes executades per obtenir la descripció gràfica de la l'arquitectura dels diferents servidors disponibles en el boada

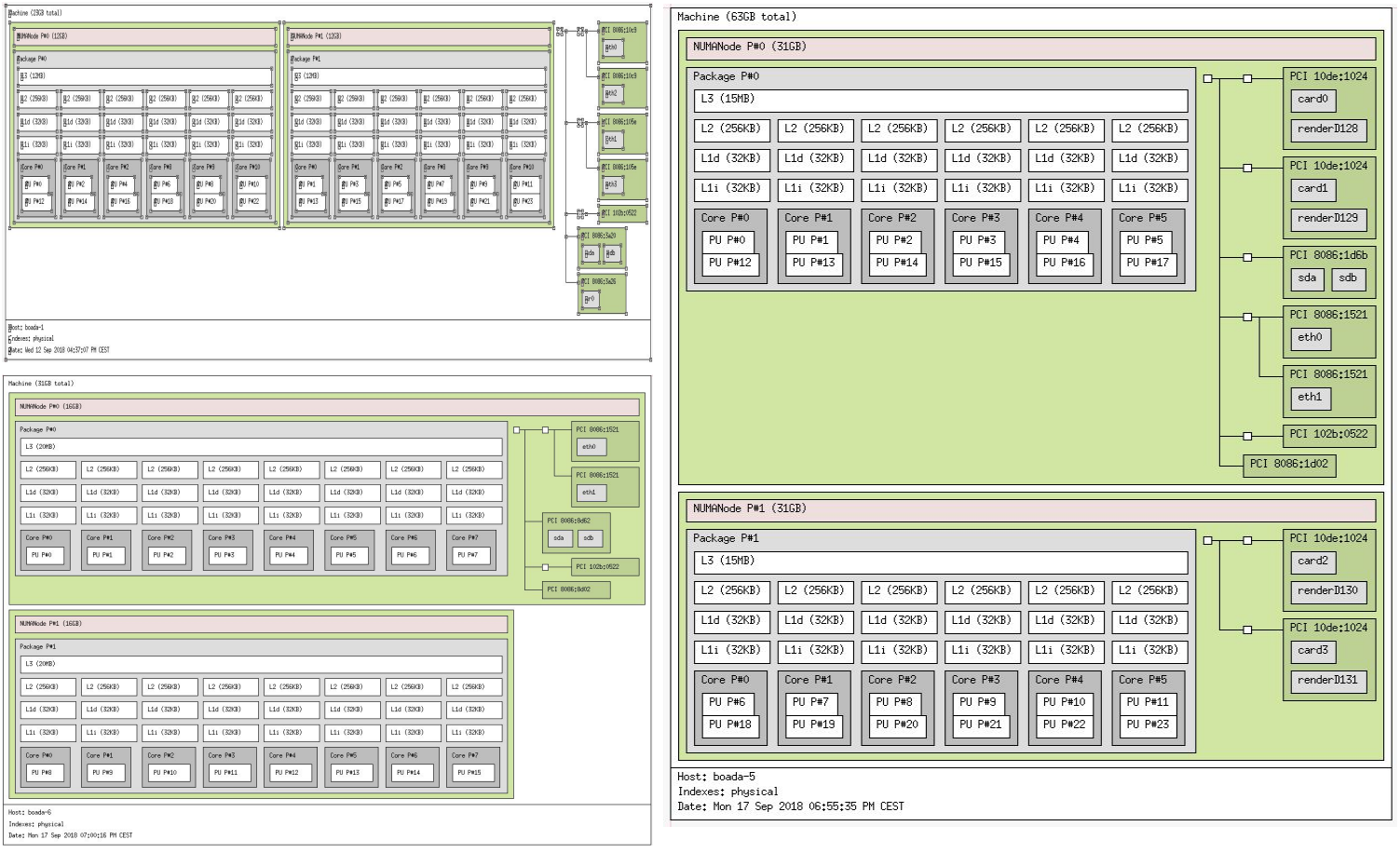


Figura 1.5 D'esquerra a dreta l'arquitectura dels tres tipus de boada: el boada-1, el boada-5 i el boada-8.

Strong vs. weak scalability

Briefly explain what strong and weak scalability refer to. Exemplify your explanation using the execution time and speed–up plots that you obtained for pi omp.c on the different node types available in boada. Reason about the results that are obtained.

Podem definir el terme *strong scalability* com la variació de temps per obtenir una solució a un problema fixe amb un nombre de CPUs variable. En canvi per a *weak scalability* entenem que el tamany del problema incrementa però volem aconseguir el mateix temps d'execució que un problema de mida menor.

En la Figura 1.6 podem observar clarament que la definició de *strong scalability* es compleix perfectament. Donat un problema d'un tamany invariable podem observar com el *speed-up* incrementa proporcionalment al nombre de threads utilitzats per executar el programa.

Strong Scalability

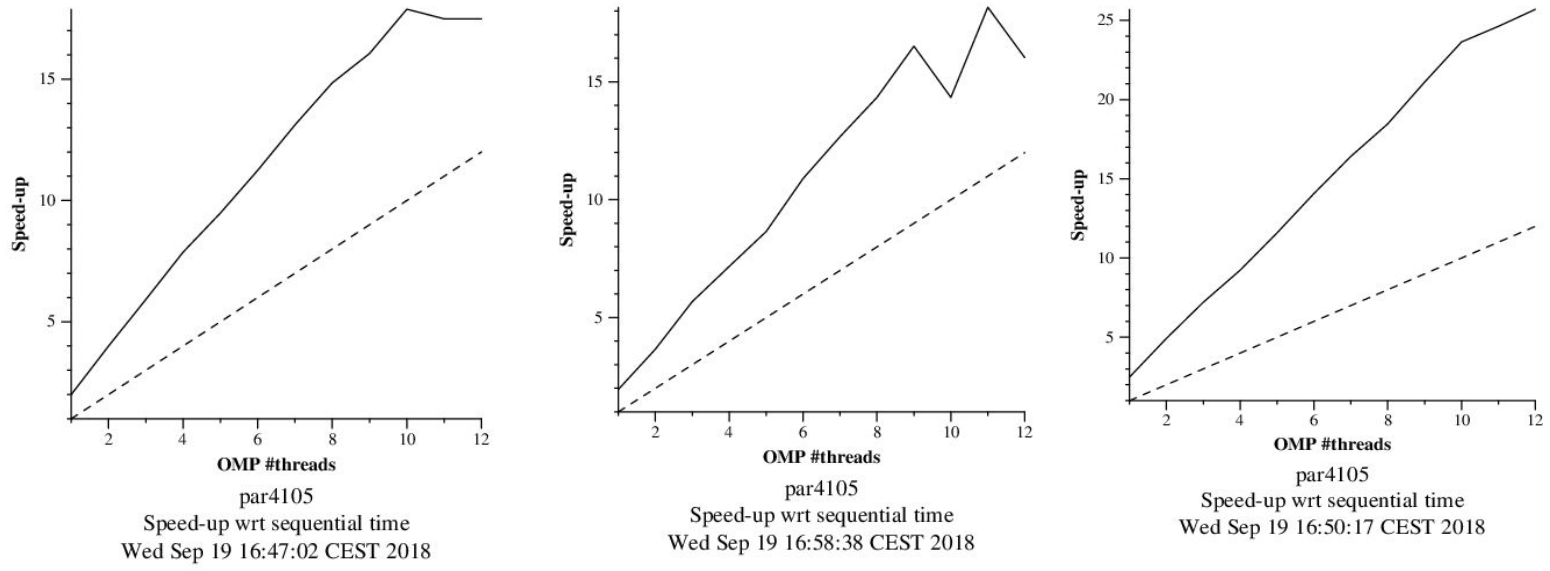


Figura 1.6 D'esquerra a dreta els gràfics del speed up respecte el número de threads en el boada-2, boada-5 i boada-6.

En quant al temps d'execució del programa es pot apreciar en la Figura 1.7 com es redueix a mesura que el número de threads aumenta.

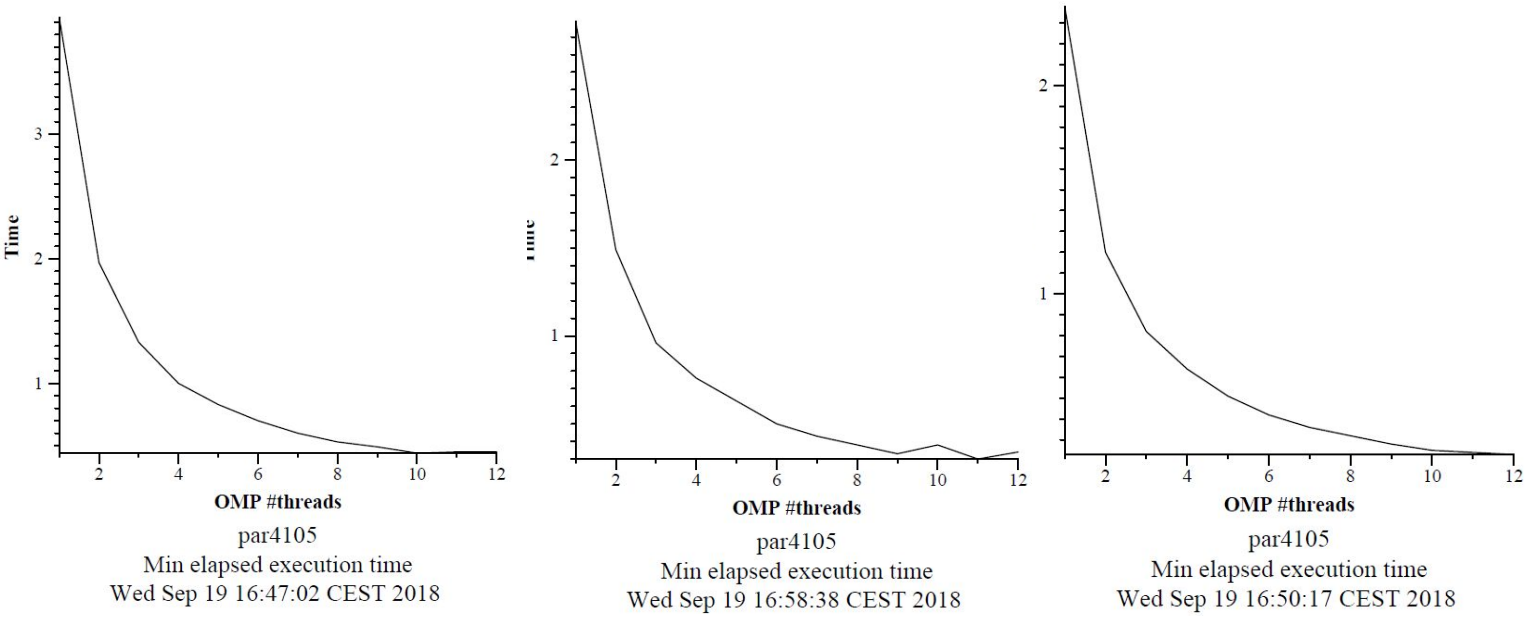


Figura 1.7 D'esquerra a dreta els gràfics del temps d'execució respecte el número de threads en el boada-2, boada-5 i boada-6.

La tendencia anteriorment observada, on es veu com clarament el *speed up* augmenta a mesura que s'afegeixen processadors, no es posible apreciar-la en els gràfics de la weak scalability, on el speed-up es manté constant o disminueix a mesura que s'incrementen el nombre de threads.

Weak Scalability

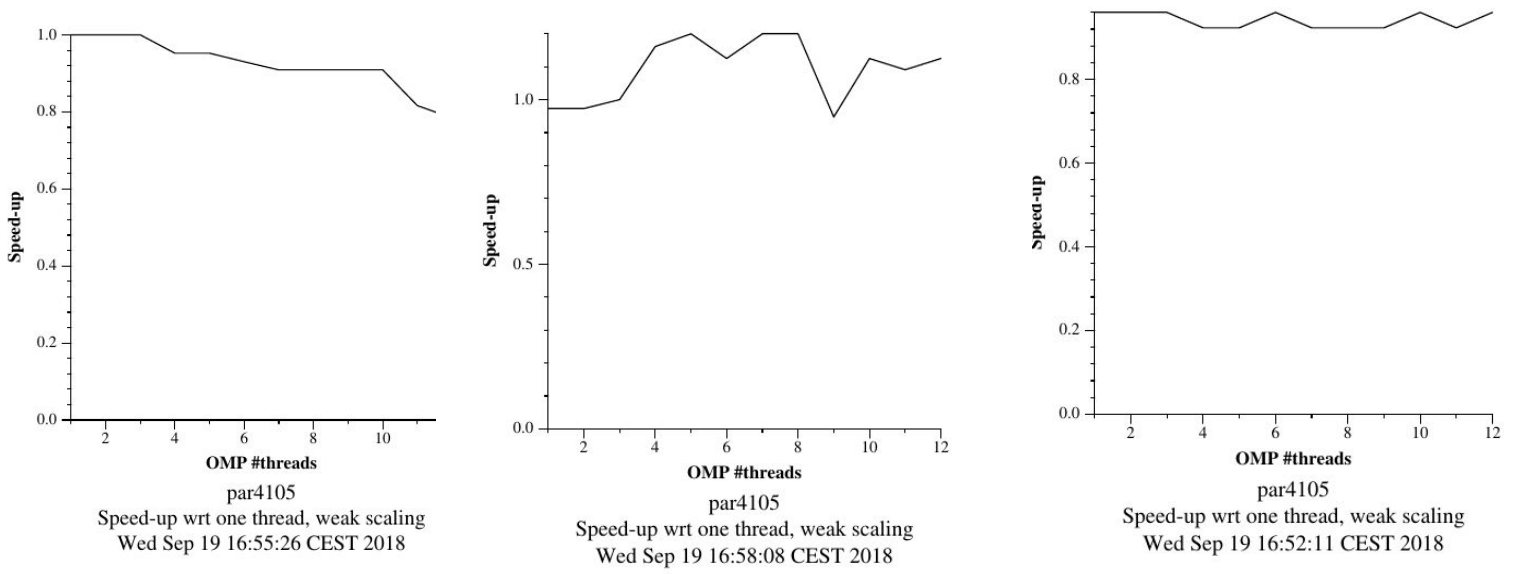


Figura 1.8 D'esquerra a dreta els gràfics del speed up respecte el número de threads en el boada-3, boada-5 i boada-7.

Analysis of task decompositions for 3DFFT

In this part of the report you should summarise the main conclusions from the analysis of task decompositions for the 3DFFT program. Backup your conclusions with the following table properly filled in with the information obtained in the laboratory session for the initial and different versions generated for 3dfft tar.c, briefly commenting the evolution of the metrics

Observant els resultats podem veure que a mesura que realitzem noves versions del programa 3DFFT aconseguim reduir la velocitat d'execució considerablement.
En la primera versió del programa l'únic que estem fent és realitzar les funcions del main de forma paral·lela. El guany que obtenim és el mateix que el que obtindriem si executèssim el programa de forma seqüencial, tal i com es pot veure en el graf de dependències de la Figura 1.9. Aquest fet es pot justificar observant que totes les funcions paral·lelitzades del main tenen dependència de les matrius *in_fttw* i *tmp_fftw*. Per tant una funció executada en un temps X determinat no pot ser executada per una CPU si abans les execucions que realitzen *loads* i *stores* en la matriu no s'han acabat de realitzar.

Figura 1.9 Graf de dependències de l'execució de la versió v1 del programa 3DFF

Version	T_1	T_∞	*Speed-up*
seq	639.780.001 ns	639.787.001 ns	0.999989
v1	639.780.001 ns	639.787.001 ns	0.999989
v2	639.780.001 ns	361.199.001 ns	1.771267
v3	639.780.001 ns	154.354.001 ns	4.144887
v4	639.780.001 ns	64.018.001 ns	9.993752
v5	639.780.001 ns	54.780.001 ns	11.679079

Figura 1.10 Taula amb els temps d'execució del programa 3DFF per a les diferents versions realitzades

Per a la següent versió, la v2, en comptes de paral·lelitzar una funció el que fem es paral·lelitzar les iteracions del bucle k de la funció *ffts1_planes*. Com es pot apreciar en la taula fent aquesta millora s'aconsegueix incrementar el *speed-up* respecte la v1.
De totes maneres aquest increment no és gaire elevat ja que, si observem les dependències en el tareador, veurem com hi ha algunes que limiten que el speed-up no sigui més elevat. Cada quadrat groc representa una tasca que s'executa en paral·lel i, si el comparem amb la v1, observarem com aconseguim més tasques que han pogut ser paral·lelitzades i, per tant, el speed-up ha de ser més elevat.

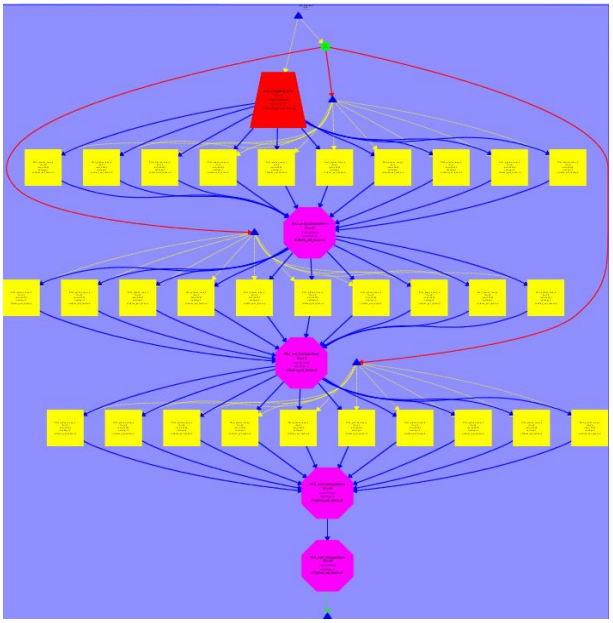


Figura 1.11 Graf de dependències de l'execució de la versió v2 del programa 3DFF

Per a la v3 el speed-up del programa gairebé es quadruplica. El fet d'aconseguir tasques més petites però més paral·lelitzables fa que el temps d'execució del programa *3DFFT* es redueixi substancialment. Cal tenir en compte que els rectangles grocs situats a la 4 fila del graf no poden ser executats pels processadors fins que els rectangles de la fila 3 han acabat la seva execució. De totes maneres si ens fixem entre aquesta versió i l'anterior observarem com es generen més tasques independents entre si que poden ser executades en paral·lel i, per tant, fa que la velocitat d'execució del programa disminueixi.

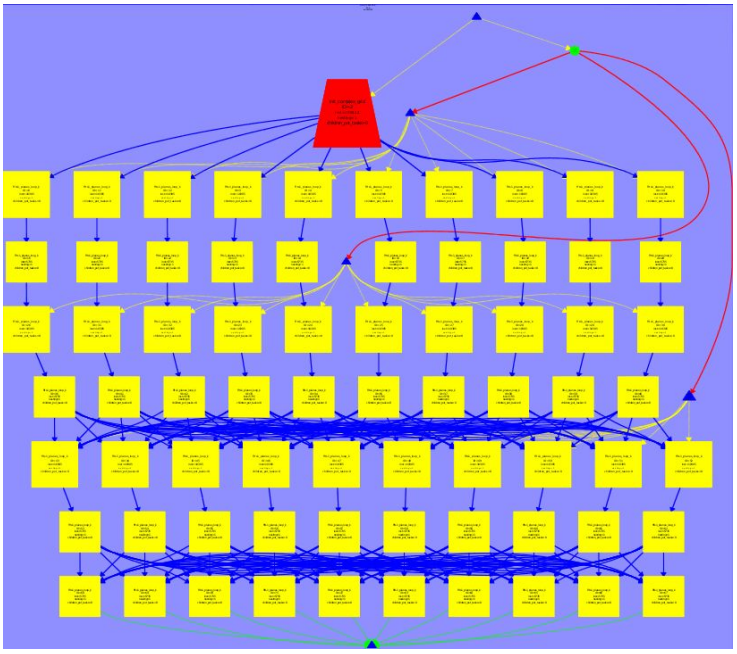


Figura 1.12 Graf de dependències de l'execució de la versió v3 del programa 3DFF

Entre la versió v3 i la versió v4 aconseguim multiplicar per 2 el *speed-up.* A simple vista observem com, entre la versió v3 i la v4, no hi ha gaire diferència si prenem com a referència els rectangles grocs. De totes maneres cal veure que en la v4 es creen 10 trapezis de color vermell mentre que en la v3 només es crea un i, fins que aquest no acaba l'execució, cap de les deu tasques grogues posteriors es pot executar. Aquest fet no passa en el graf de la Figura 1.13 on clarament es veu que per cada columna de rectangles grocs hi ha un trapezi independent als altres que no impedeix l'execució dels rectangles situats en la mateixa fila però en columnes diferents.

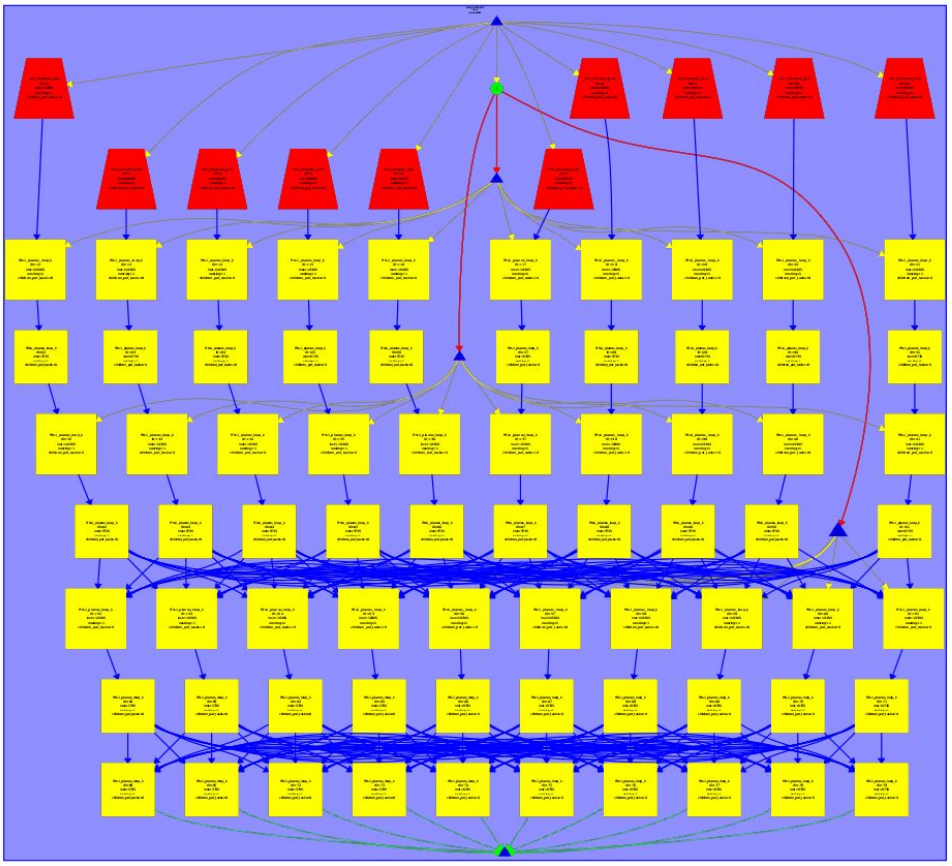


Figura 1.13 Graf de dependències de l'execució de la versió v4 del programa 3DFF

Per fer la v5 ens va ser impossible copiar el graf de dependències ja que trigava molt de temps en carregar i, quan feiem zoom, la finestra es penjava i havíem de fer de nou tot el procés. És per això que en la Figura 1.14 només apareix el graf però amb poc zoom.

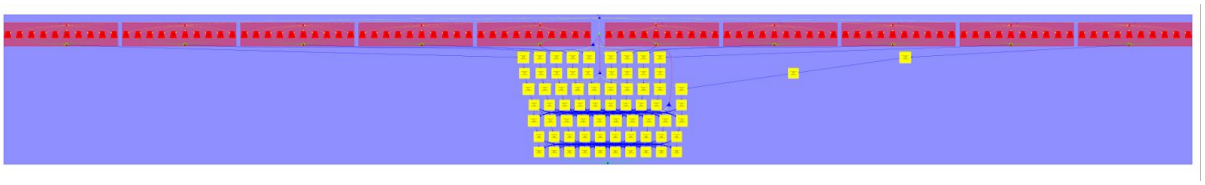


Figura 1.14 Graf de dependències de l'execució de la versió v5 del programa 3DFF

En l'enunciat no especifica quantes tasques en paral·lel s'han d'especificar. És per aquest motiu que per fer el estudi de la v5 hem fet dues versions diferents.

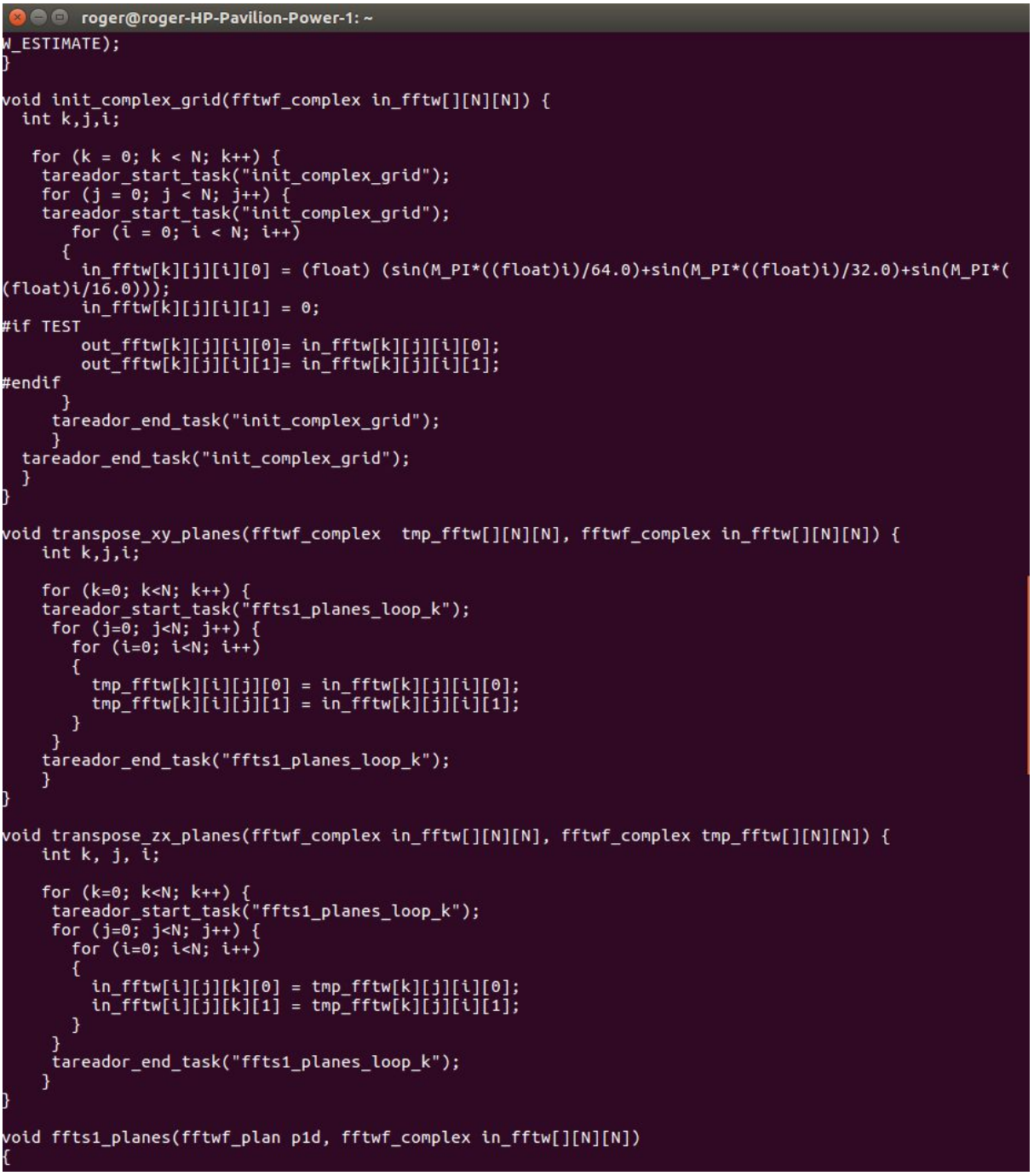


Figura 1.15 Codi v5.1 de l'arxiu 3DFFT_tar.c

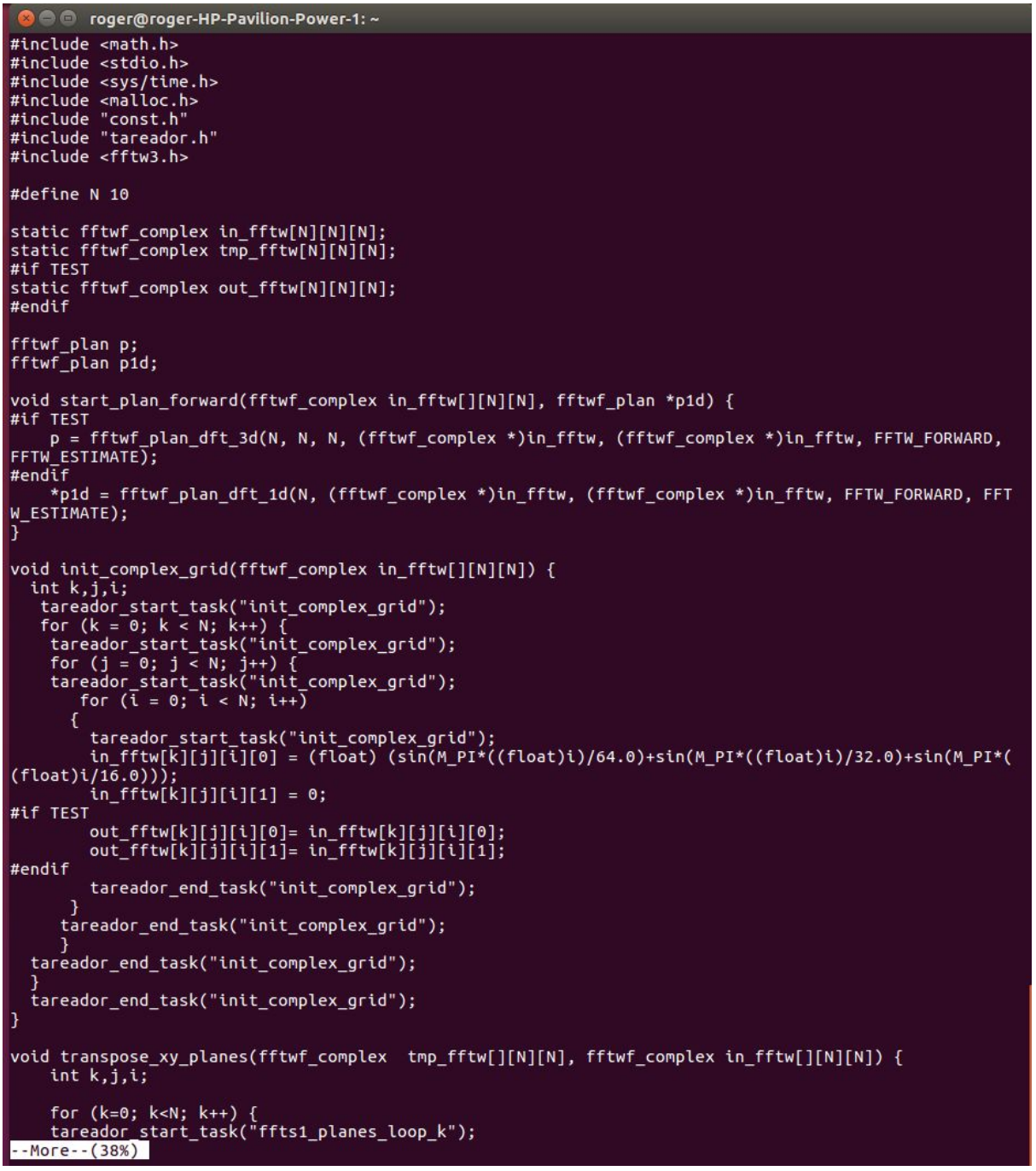
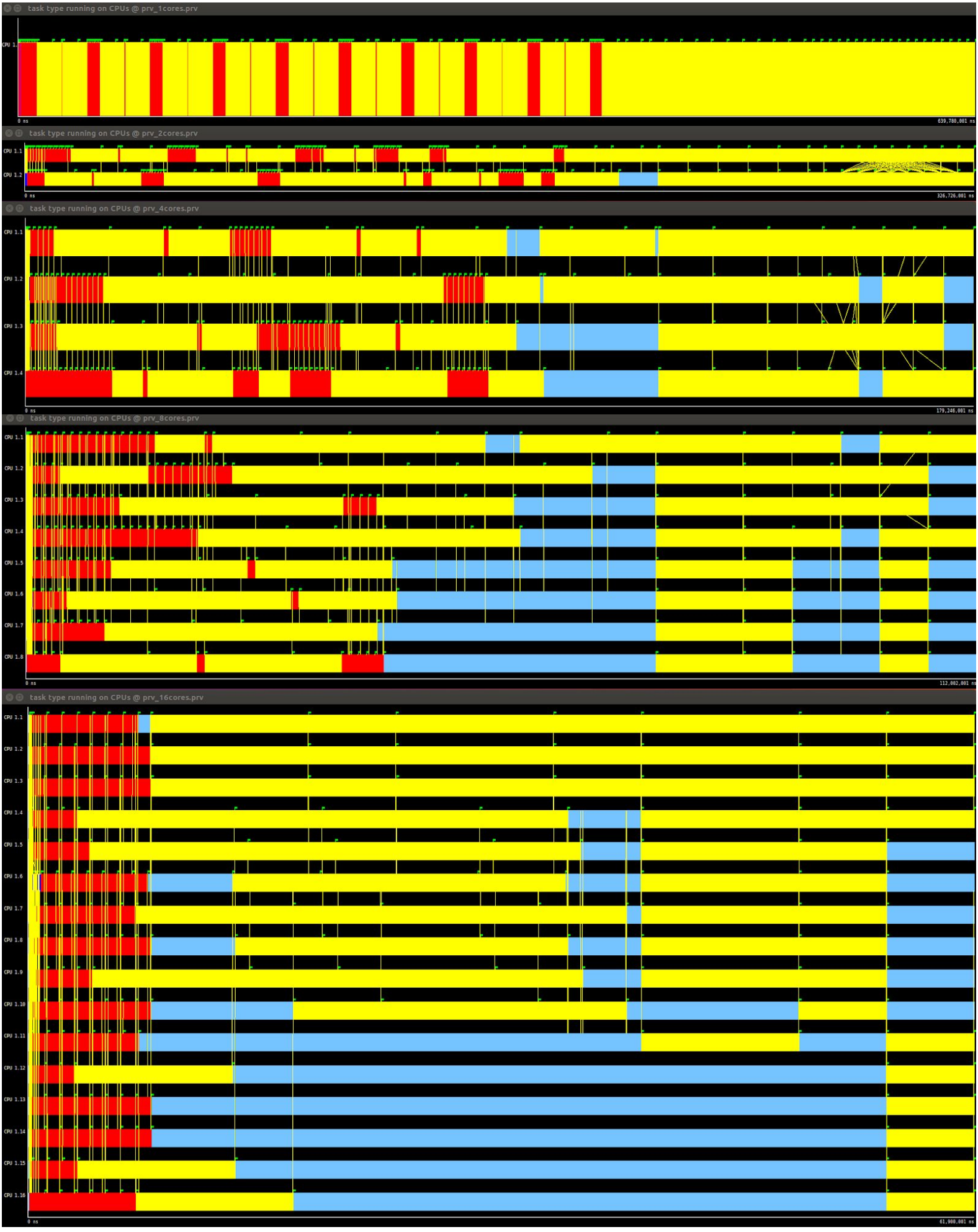


Figura 1.16 Codi v5.2 de l'arxiu 3DFFT_tar.c

versió	T_1	T_2	T_4	T_8	T_{16}	T_{32}
v4	639.780.001	320.310.001	165.389.001	91.496.001	64.018.001	64.018.001
v5.1	639.780.001	326.726.001	179.246.006	112.002.001	61.900.001	57.290.001
v5.2	639.780.001	319.914.001	165.172.001	97.499.001	60.278.001	57.124.001

Figura 1.17 Resultats de les execucions de l'arxiu 3DFFT_tar.c. Totes les dades estan expressades en nanosegons.


task type running on CPUs @ prv_1cores.prv
CPU 1.1
0 ns
639,788,001 ns
task type running on CPUs @ prv_2cores.prv
CPU 1.1
CPU 1.2
0 ns
326,726,001 ns
task type running on CPUs @ prv_4cores.prv
CPU 1.1
CPU 1.2
CPU 1.3
CPU 1.4
0 ns
179,246,001 ns
task type running on CPUs @ prv_8cores.prv
CPU 1.1
CPU 1.2
CPU 1.3
CPU 1.4
CPU 1.5
CPU 1.6
CPU 1.7
CPU 1.8
0 ns
112,002,001 ns
task type running on CPUs @ prv_16cores.prv
CPU 1.1
CPU 1.2
CPU 1.3
CPU 1.4
CPU 1.5
CPU 1.6
CPU 1.7
CPU 1.8
CPU 1.9
CPU 1.10
CPU 1.11
CPU 1.12
CPU 1.13
CPU 1.14
CPU 1.15
CPU 1.16
0 ns
61,900,001 ns

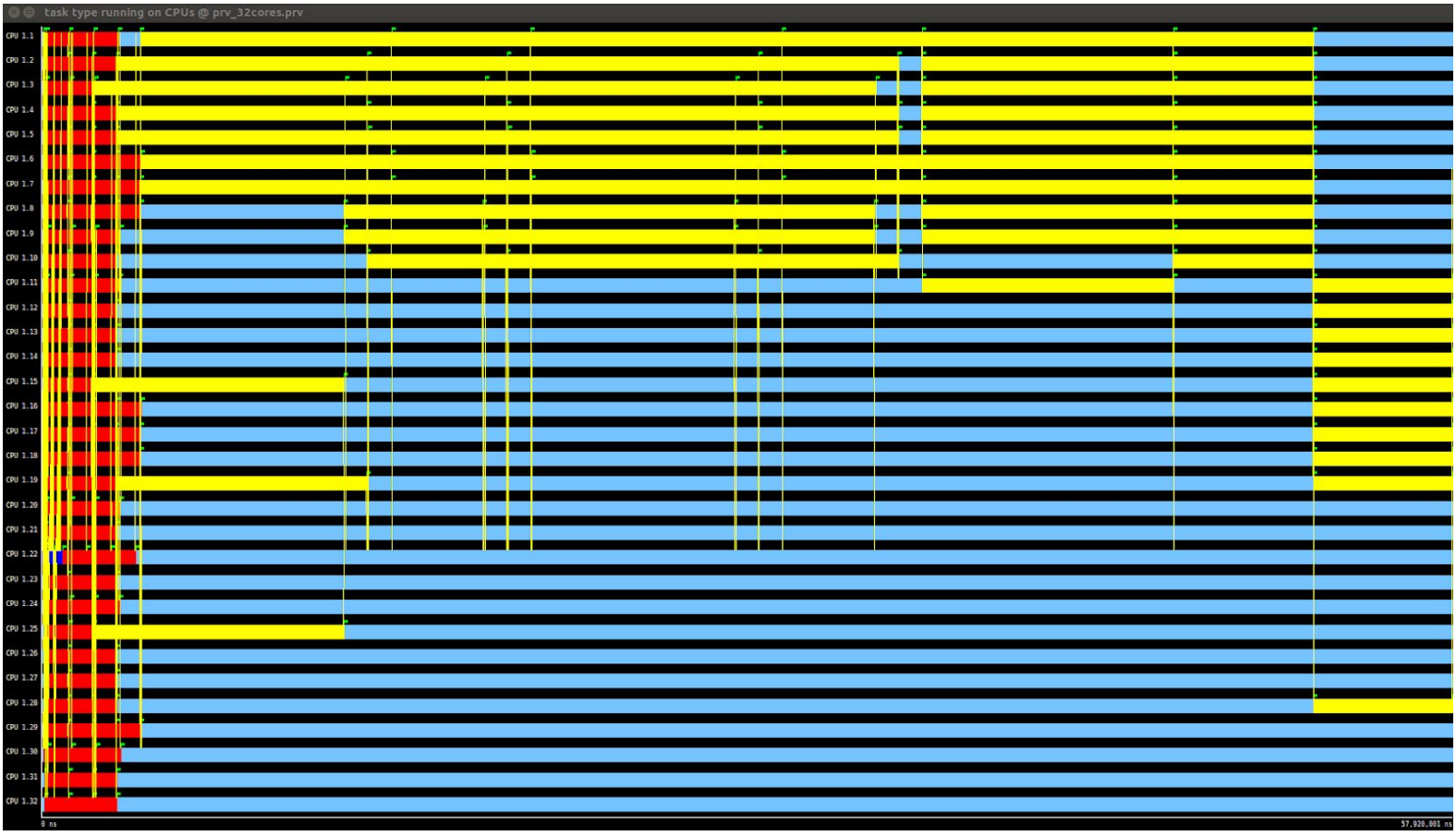
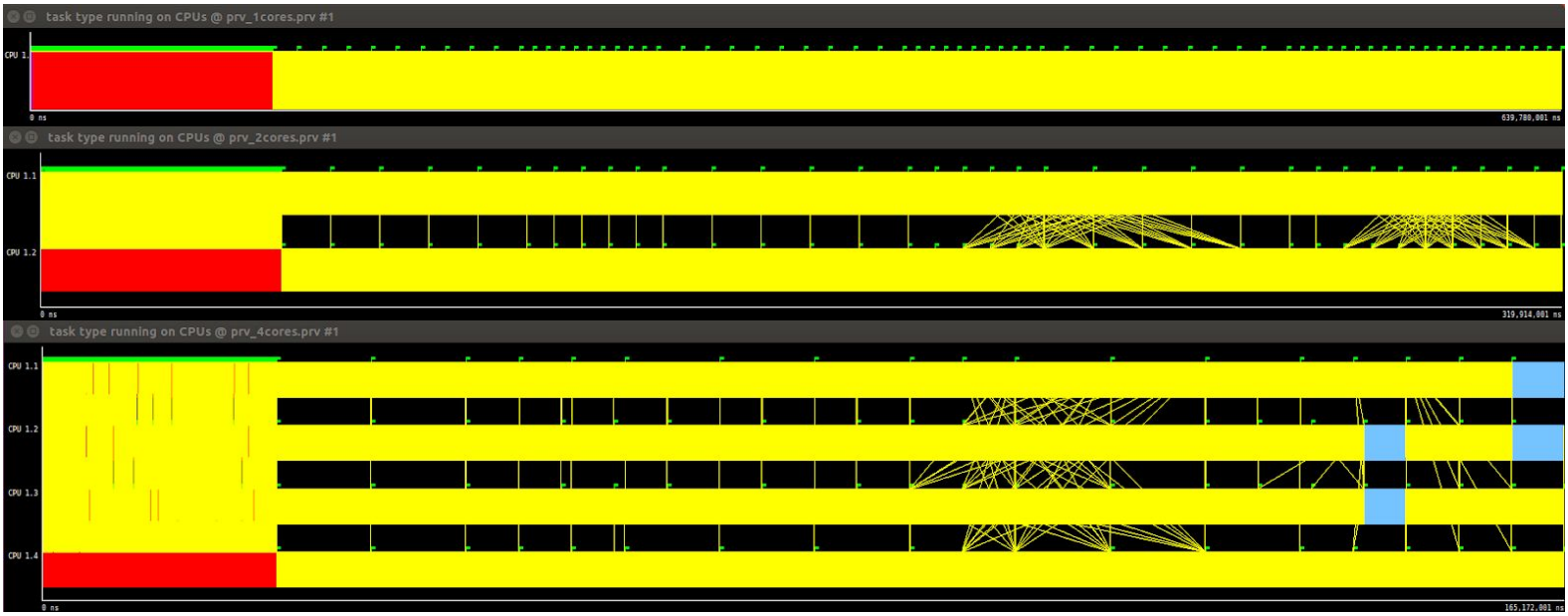
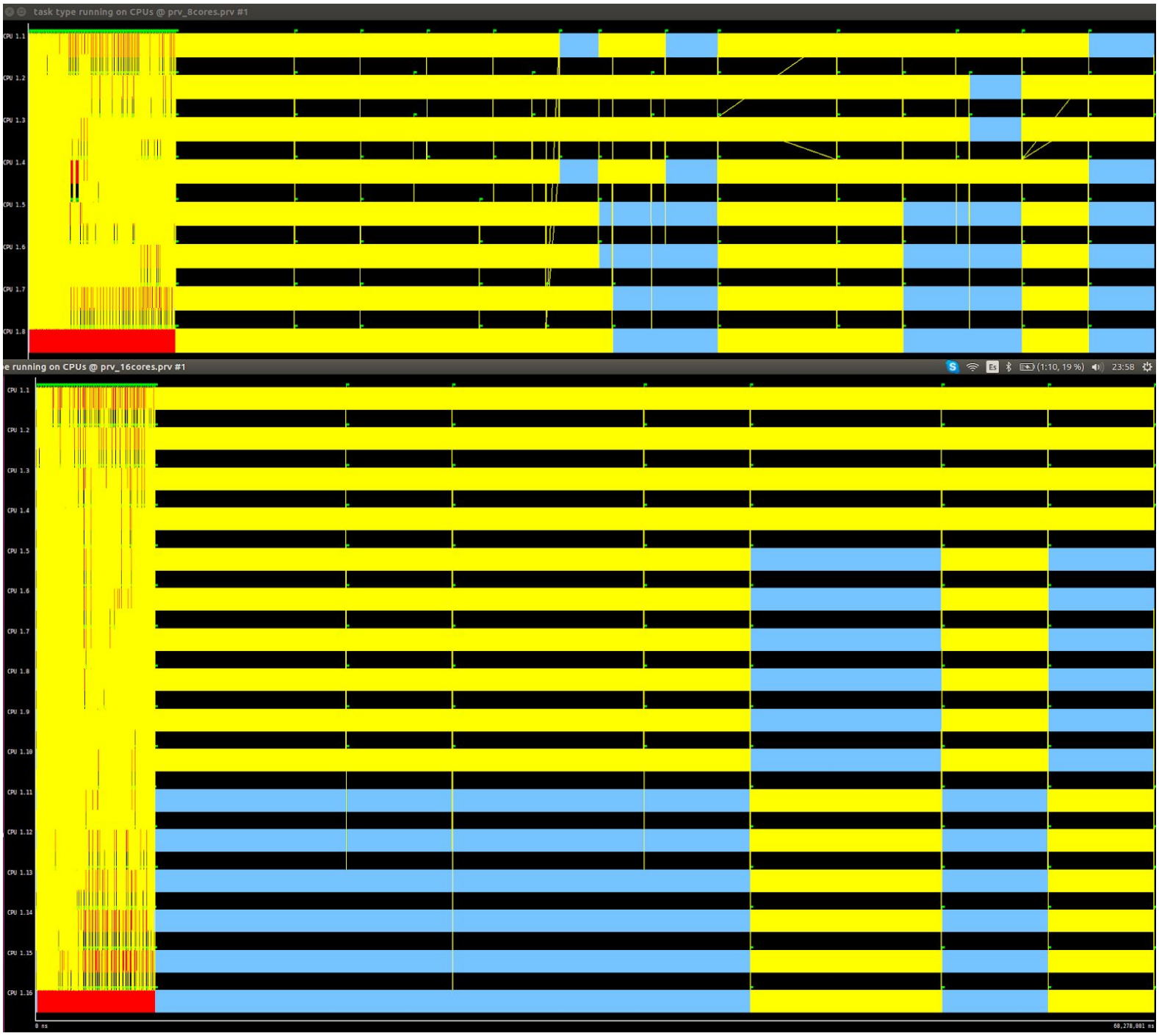


Figura 1.18 Gràfic paraver del programa 3DFFT_tar.c v5.1




task type running on CPUs @ prv_8cores.prv #1
CPU 1.1
CPU 1.2
CPU 1.3
CPU 1.4
CPU 1.5
CPU 1.6
CPU 1.7
CPU 1.8
e running on CPUs @ prv_16cores.prv #1
(1:10, 19 %)
23:58
CPU 1.1
CPU 1.2
CPU 1.3
CPU 1.4
CPU 1.5
CPU 1.6
CPU 1.7
CPU 1.8
CPU 1.9
CPU 1.10
CPU 1.11
CPU 1.12
CPU 1.13
CPU 1.14
CPU 1.15
CPU 1.16
0 ns
60,278,881 ns

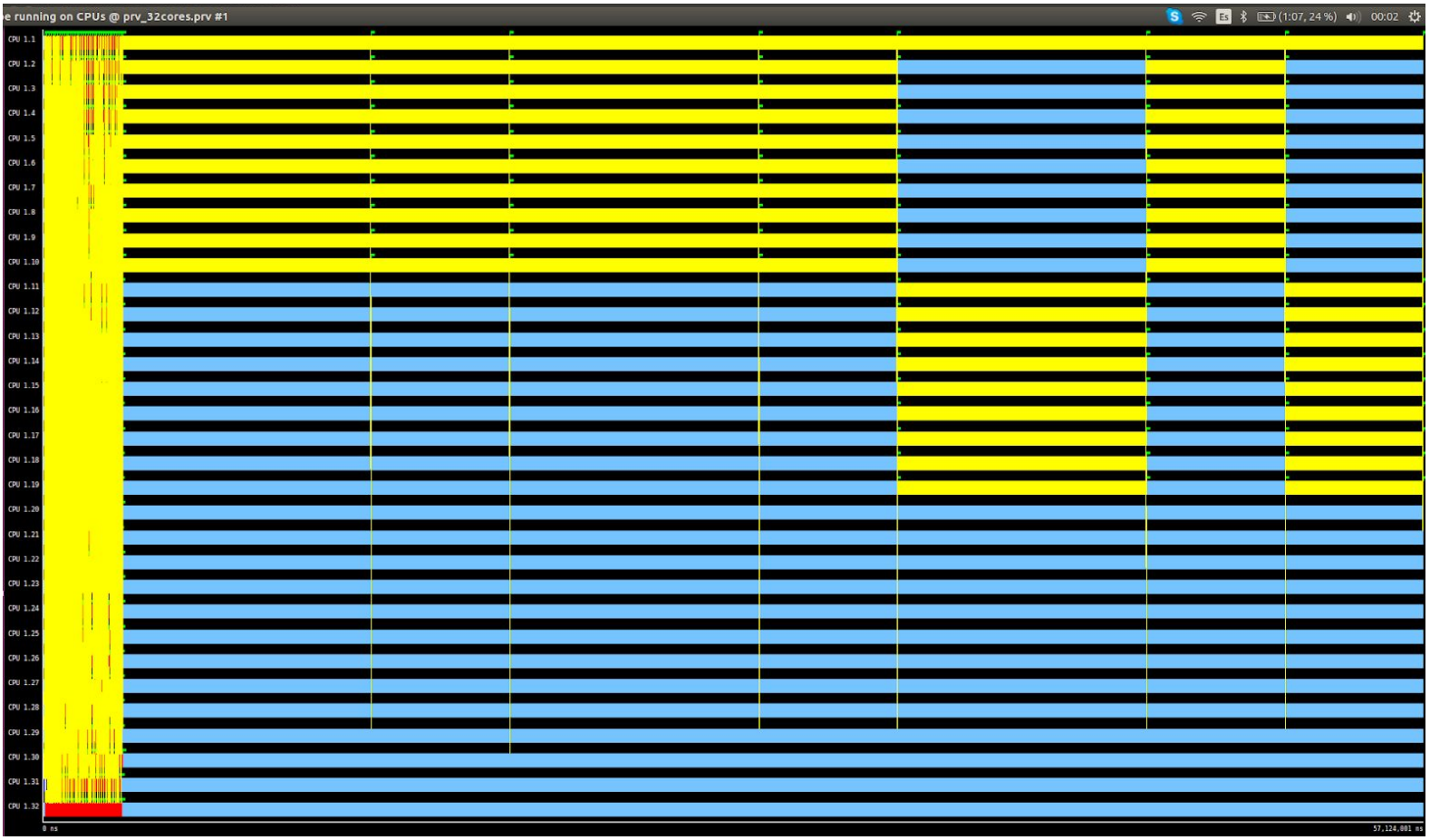


Figura 1.19 Gràfic paraver del programa 3DFFT_tar.c v5.2

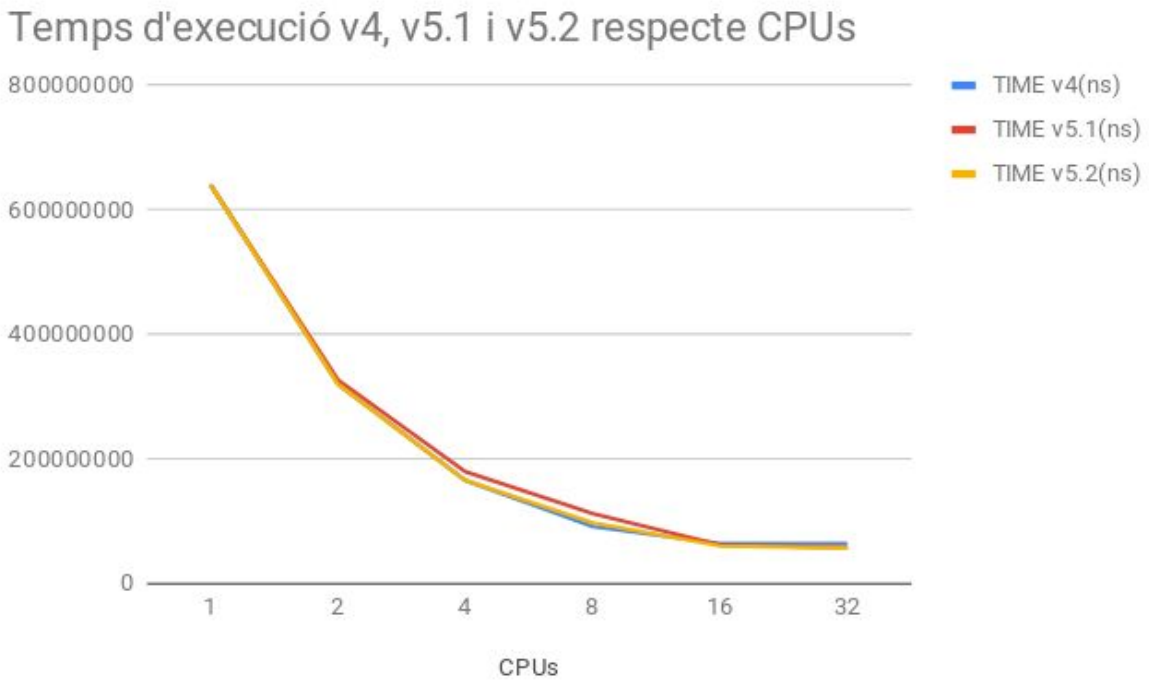


Figura 1.20 Gràfic que compara el temps d'execució del programa 3DFFT_tar.c entre la versió v5.1 i la v5.2

Com es pot apreciar amb l'observació del gràfic de la Figura 1.20 i la taula de la Figura 1.17 es veu com clarament tant la versió 4 com la versió 5 triguen, aproximadament, el mateix. Si ens fixem amb el detall és cert que la v5 s'executa amb un temps inferior a la v4 però la diferència és mínima. Entre les versions v5.1, amb menys granularitat, i la v5.2, amb més granularitat, es pot afirmar que no s'aconsegueixen resultats gaire més bons pel simple fet d'afegir més tasques en paral·lel ja que seguiran existint les mateixes dependències, que son les que ens limiten la paral·lelització del programa.

Understanding the parallel execution of 3DFFT

In this final section of your report you should comment about the actual parallel performance of 3DFFT when parallelised using OpenMP . Try to make a coherent history that shows how you optimised the code with the aim of increasing the parallel fraction of the program, reducing parallelisation overheads and improve load balancing. Accompany your explanations with the results reported in the following table which you obtained during the laboratory session. It is very important that you include the relevant Paraver captures (timelines and profiles of the % of time spent in the different OpenMP states) to support your explanations too.

	ϕ	S_∞	T_1	T_8	S_8
initial	0.867495	7.546885	2.461.877.422 ns	1.487.149.489 ns	1.6545
improved ϕ	0.8787	8.244023	2.403.635.083 ns	1.045.440.737 ns	2.2994
improved parallel overheads	0.8671	7.524454	2.439.801.827 ns	1.049.294.676 ns	2.3250
improved work-distribution overheads	0.8942	9.451795	2.385.734.191 ns	783.145.877 ns	3.0469

Figura 1.21 Taula de les diferents versions de l'arxiu 3DFFT_omp.c

Les dades de la Figura 1.21 s'aconsegueixen de la següent manera:

- Tseq inclou les fases de *Synchronization*, *Schedule and Fork*, la fase inicial i fase final del període *Running*.
- T_{par} és tot el temps restant d'execució.
- T_p és el temps total d'execució total amb p processadors.
- $S_p = T_1 / T_p$
- $\phi = T_{par}/(T_{seq} + T_{par})$
- $S_\infty = 1/(1 - \phi)$

Per realitzar la primera modificació del codi vam incloure l'etiqueta de *#pragma omp parallel* i vam establir que la variable *i* fos privada dins funció *init_complex_grid*. La millora la realitzem perquè la funció no es paral·lelitza dins del seu propi cos, sino que es paral·litzada en el main, i aquest fet fa que el valor de ϕ sigui menor.

```
void init_complex_grid(fftwf_complex in_fftw[][N][N]) {
  int k,j,i;
  for (k = 0; k < N; k++)
  #pragma omp parallel
  #pragma omp for schedule(static,1) private(i)
   for (j = 0; j < N; j++)
          for (i = 0; i < N; i++)
          {
          in_fftw[k][j][i][0] = (float)
(sin(M_PI*((float)i)/64.0)+sin(M_PI*((float)i)/32.0)+sin(M_PI*((float)i/16.0)));
          in_fftw[k][j][i][1] = 0;
#if TEST
          out_fftw[k][j][i][0]= in_fftw[k][j][i][0];
          out_fftw[k][j][i][1]= in_fftw[k][j][i][1];
#endif
          }
}
```

Figura 1.22 Codi de la versió 2 de l'arxiu 3DFFT_omp.c

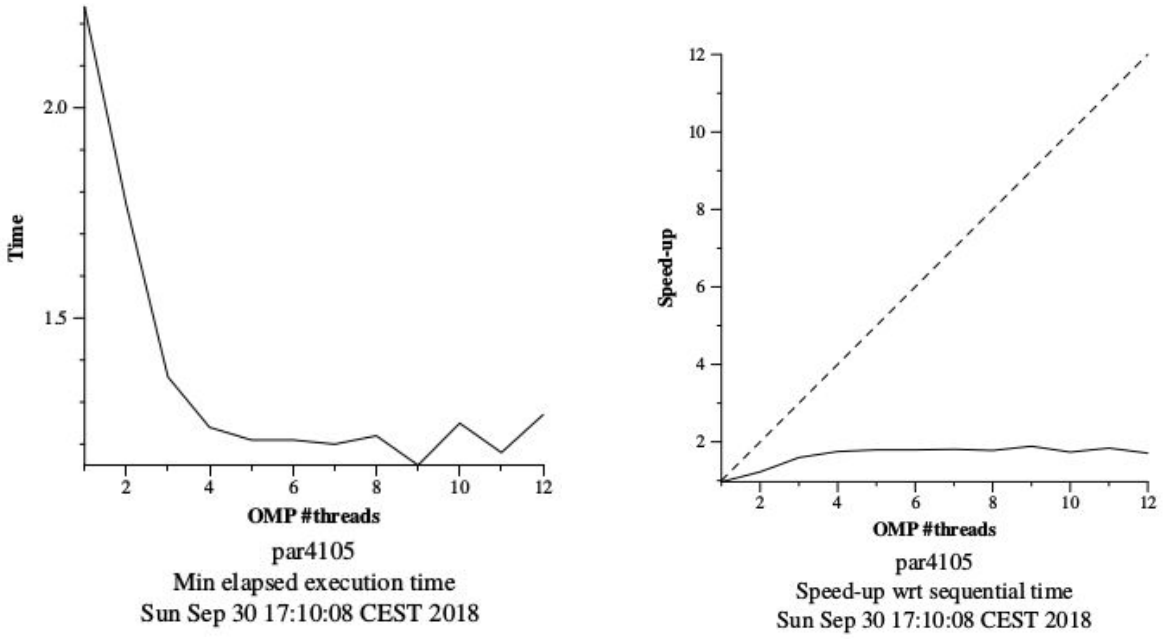


Figura 1.23 Gràfics de temps d'execució i speed-up de la versió 2 de l'arxiu 3DFFT_omp.c

Figura 1.24 Timeline de l'execució de la versió 2 de l'arxiu 3DFFT_omp.c amb una CPU

2D profile @ 3dfft_omp-1-boada-1.prv

	Running	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	99.66 %	0.06 %	0.25 %	0.03 %	0.00 %
Total	99.66 %	0.06 %	0.25 %	0.03 %	0.00 %
Average	99.66 %	0.06 %	0.25 %	0.03 %	0.00 %
Maximum	99.66 %	0.06 %	0.25 %	0.03 %	0.00 %
Minimum	99.66 %	0.06 %	0.25 %	0.03 %	0.00 %
StDev	0 %	0 %	0 %	0 %	0 %
Avg/Max	1	1	1	1	1

Figura 1.25 Perfil de l'execució de la versió 2 de l'arxiu 3DFFT_omp.c amb una CPU

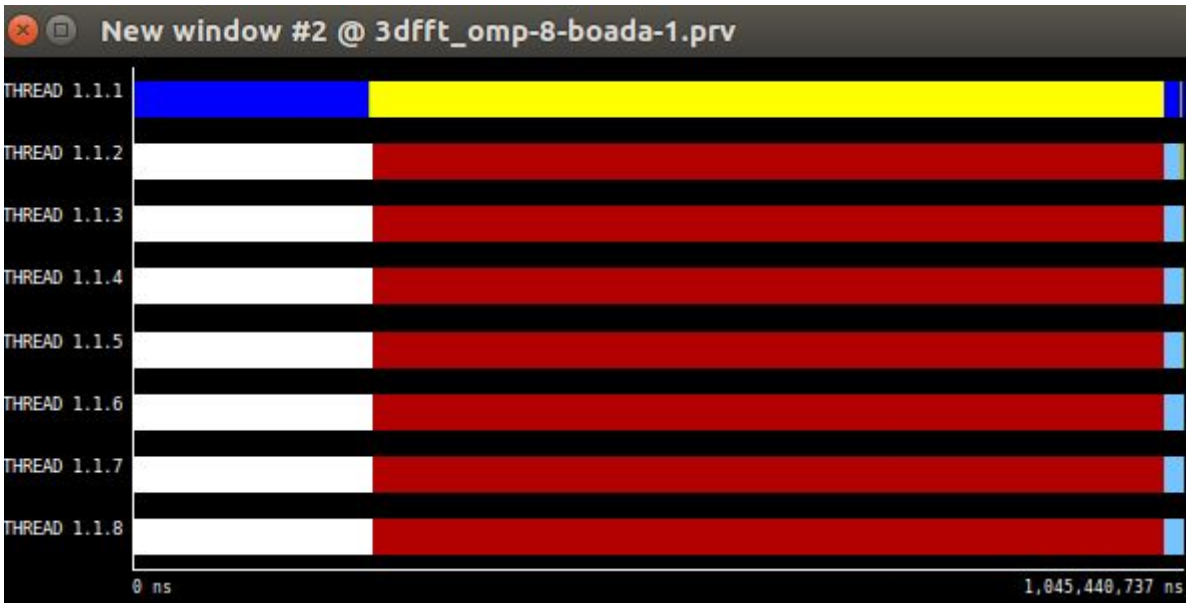


Figura 1.26 Timeline de l'execució de la versió 2 de l'arxiu 3DFFT_omp.c amb vuit CPUs

2D profile @ 3dfft_omp-8-boada-1.prv

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	89.40 %	-	9.24 %	1.28 %	0.09 %	0.00 %
THREAD 1.1.2	65.55 %	23.52 %	10.87 %	-	0.06 %	-
THREAD 1.1.3	66.75 %	23.51 %	9.68 %	-	0.06 %	-
THREAD 1.1.4	61.92 %	23.54 %	14.48 %	-	0.06 %	-
THREAD 1.1.5	57.52 %	23.55 %	18.88 %	-	0.05 %	-
THREAD 1.1.6	68.42 %	23.56 %	7.96 %	-	0.06 %	-
THREAD 1.1.7	74.19 %	23.54 %	2.21 %	-	0.06 %	-
THREAD 1.1.8	66.76 %	23.57 %	9.62 %	-	0.05 %	-
Total	550.51 %	164.79 %	82.94 %	1.28 %	0.48 %	0.00 %
Average	68.81 %	23.54 %	10.37 %	1.28 %	0.06 %	0.00 %
Maximum	89.40 %	23.57 %	18.88 %	1.28 %	0.09 %	0.00 %
Minimum	57.52 %	23.51 %	2.21 %	1.28 %	0.05 %	0.00 %
StDev	8.99 %	0.02 %	4.53 %	0 %	0.01 %	0 %
Avg/Max	0.77	1.00	0.55	1	0.70	1

Figura 1.27 Perfil de l'execució de la versió 2 de l'arxiu 3DFFT_omp.c amb vuit CPUs

Per a realitzar la versió 3 del programa teníem la intenció de reduir al màxim els overheads. Per aconseguir-ho vam reescriure els pragmas de totes les funcions privatitzant la variable j per minimitzar el nombre construccions paral·leles.

```
#pragma omp parallel
#pragma omp for schedule(static,1) private(i) private(j)
```

Figura 1.28 Codi de la versió 3 de l'arxiu 3DFFT_omp.c en la funció init_complex_grid.

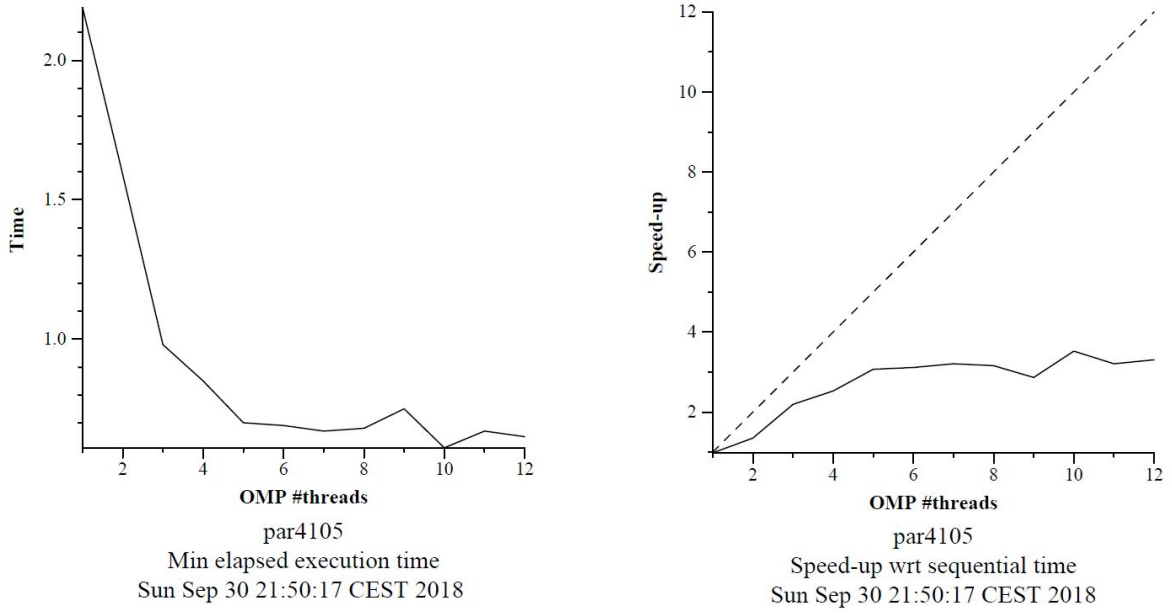


Figura 1.29 Gràfics de temps d'execució i speed-up de la versió 3 de l'arxiu 3DFFT_omp.c

Figura 1.30 Timeline de l'execució de la versió 3 de l'arxiu 3DFFT_omp.c amb una CPU

2D profile @ 3dfft_omp-1-boada-1.prv

	Running	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	99.52 %	0.20 %	0.25 %	0.03 %	0.00 %
Total	99.52 %	0.20 %	0.25 %	0.03 %	0.00 %
Average	99.52 %	0.20 %	0.25 %	0.03 %	0.00 %
Maximum	99.52 %	0.20 %	0.25 %	0.03 %	0.00 %
Minimum	99.52 %	0.20 %	0.25 %	0.03 %	0.00 %
StDev	0 %	0 %	0 %	0 %	0 %
Avg/Max	1	1	1	1	

Figura 1.31 Perfil de l'execució de la versió 3 de l'arxiu 3DFFT_omp.c amb una CPU

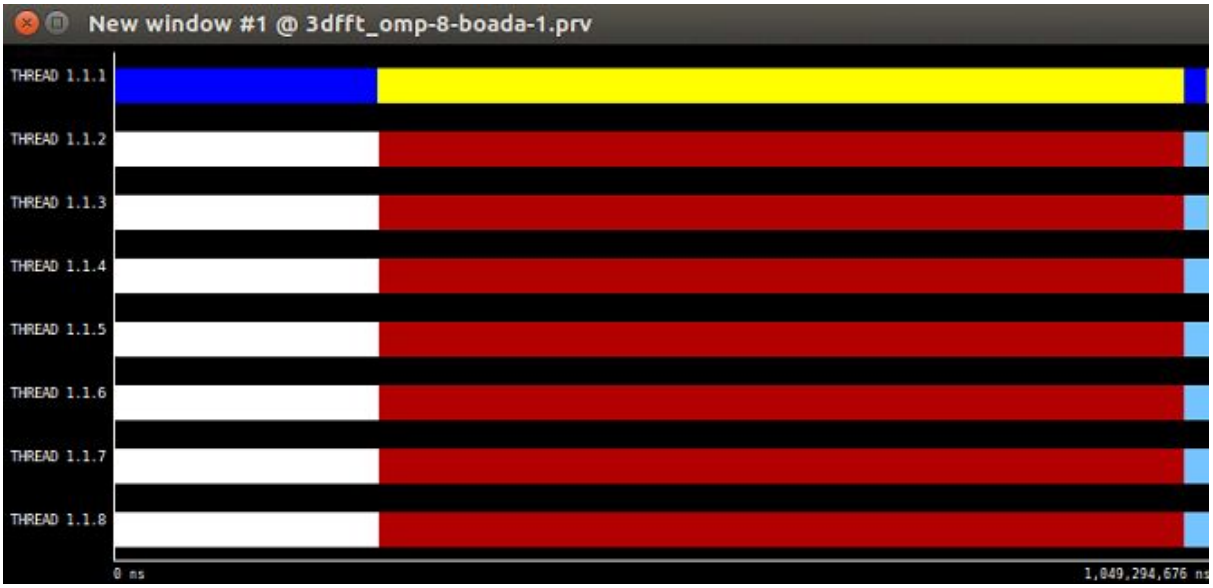


Figura 1.32 Timeline de l'execució de la versió 3 de l'arxiu 3DFFT_omp.c amb vuit CPUs

2D profile @ 3dfft_omp-8-boada-1.prv

	Running	Not created	Synchronization	Scheduling and Fork/Join
THREAD 1.1.1	95.02 %	-	3.66 %	1.24 %
THREAD 1.1.2	66.73 %	25.17 %	8.04 %	-
THREAD 1.1.3	65.85 %	25.16 %	8.93 %	-
THREAD 1.1.4	66.11 %	25.23 %	8.61 %	-
THREAD 1.1.5	61.31 %	25.15 %	13.48 %	-
THREAD 1.1.6	64.97 %	25.17 %	9.80 %	-
THREAD 1.1.7	67.46 %	25.17 %	7.32 %	-
THREAD 1.1.8	62.45 %	25.19 %	12.30 %	-
Total	549.91 %	176.24 %	72.14 %	1.24 %

Figura 1.33 Perfil de l'execució de la versió 3 de l'arxiu 3DFFT_omp.c amb vuit CPUs

En la versió 4 es demana moure els pragmas de les funcions que estaven just abans del bucle j i posar-les sobre el bucle k, per tal de paral·lelitzar les iteracions del bucle més exterior i reduir overheads i el temps d'execució.

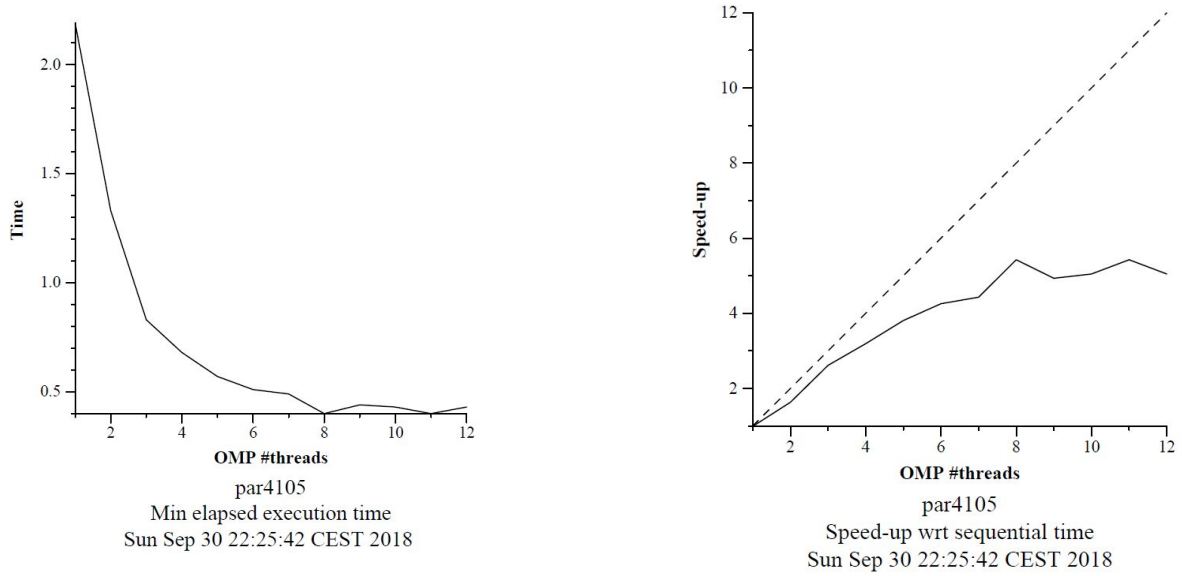


Figura 1.34 Gràfics dels temps d'execució i speed-up de la versió 4 de l'arxiu 3DFFT_omp.c

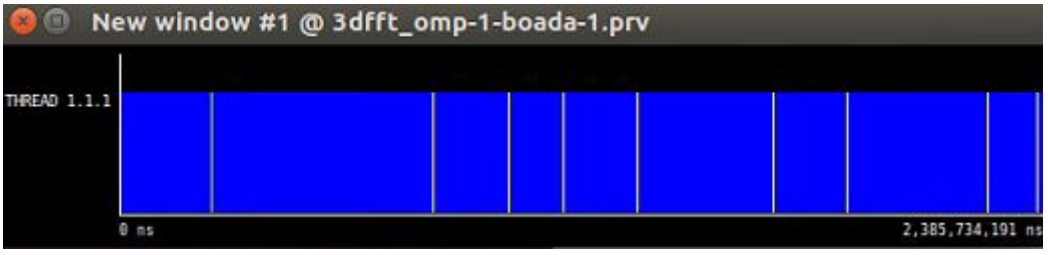
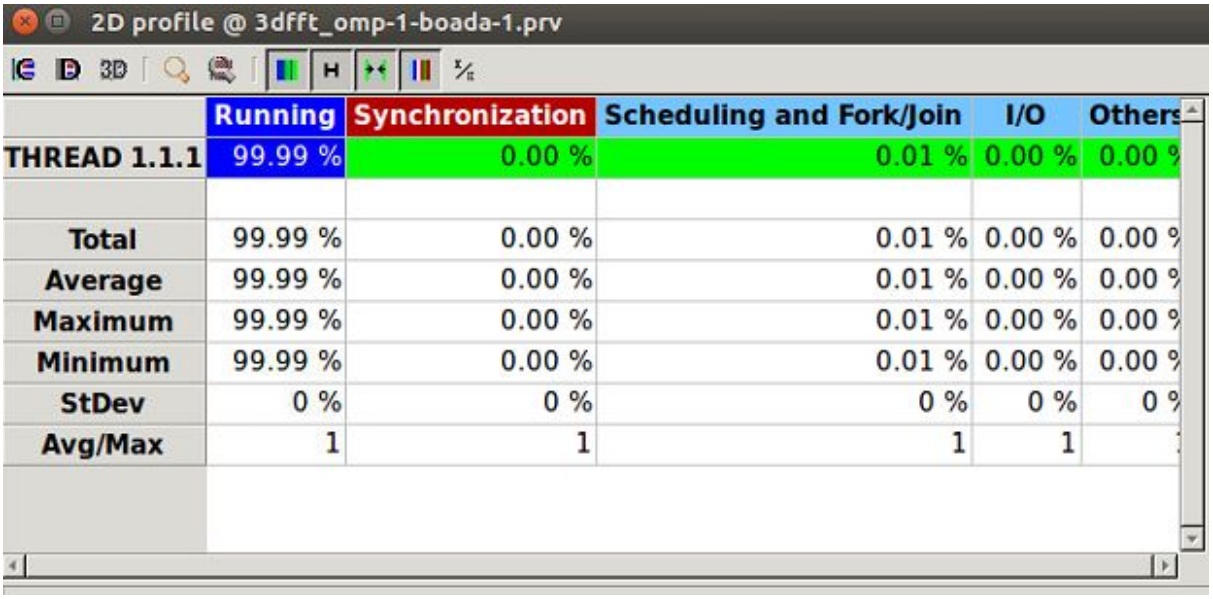


Figura 1.35 Timeline de l'execució de la versió 4 de l'arxiu 3DFFT_omp.c amb una CPU



	Running	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	99.99 %	0.00 %	0.01 %	0.00 %	0.00 %
Total	99.99 %	0.00 %	0.01 %	0.00 %	0.00 %
Average	99.99 %	0.00 %	0.01 %	0.00 %	0.00 %
Maximum	99.99 %	0.00 %	0.01 %	0.00 %	0.00 %
Minimum	99.99 %	0.00 %	0.01 %	0.00 %	0.00 %
StDev	0 %	0 %	0 %	0 %	0 %
Avg/Max	1	1	1	1	

Figura 1.36 Perfil de l'execució de la versió 4 de l'arxiu 3DFFT_omp.c amb una CPU

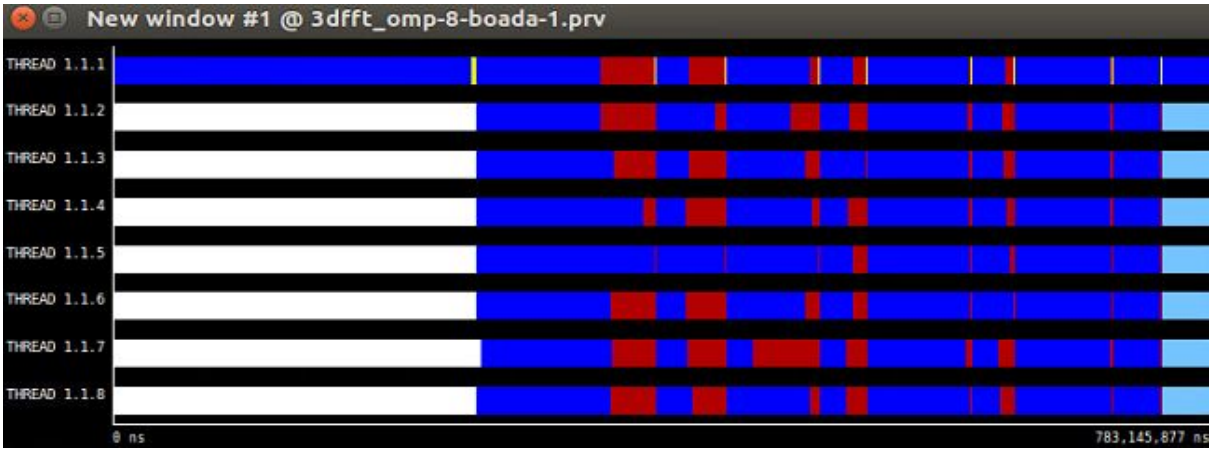
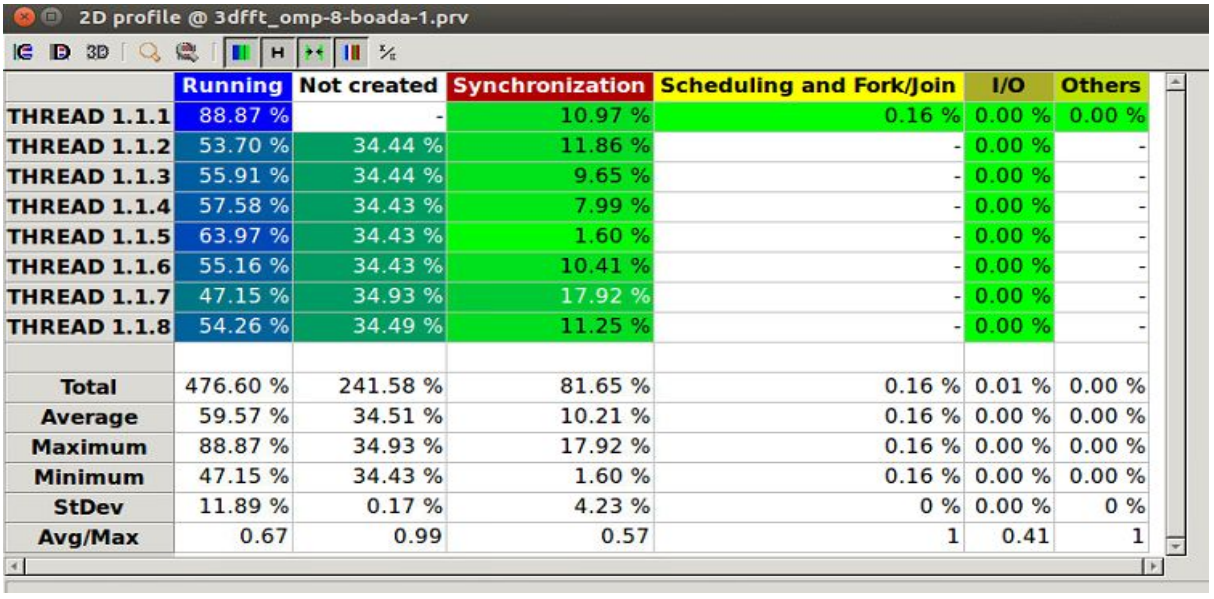


Figura 1.37 Timeline de l'execució de la versió 4 de l'arxiu 3DFFT_omp.c amb vuit CPUs



	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	88.87 %	-	10.97 %	0.16 %	0.00 %	0.00 %
THREAD 1.1.2	53.70 %	34.44 %	11.86 %	-	0.00 %	-
THREAD 1.1.3	55.91 %	34.44 %	9.65 %	-	0.00 %	-
THREAD 1.1.4	57.58 %	34.43 %	7.99 %	-	0.00 %	-
THREAD 1.1.5	63.97 %	34.43 %	1.60 %	-	0.00 %	-
THREAD 1.1.6	55.16 %	34.43 %	10.41 %	-	0.00 %	-
THREAD 1.1.7	47.15 %	34.93 %	17.92 %	-	0.00 %	-
THREAD 1.1.8	54.26 %	34.49 %	11.25 %	-	0.00 %	-
Total	476.60 %	241.58 %	81.65 %	0.16 %	0.01 %	0.00 %
Average	59.57 %	34.51 %	10.21 %	0.16 %	0.00 %	0.00 %
Maximum	88.87 %	34.93 %	17.92 %	0.16 %	0.00 %	0.00 %
Minimum	47.15 %	34.43 %	1.60 %	0.16 %	0.00 %	0.00 %
StDev	11.89 %	0.17 %	4.23 %	0 %	0.00 %	0 %
Avg/Max	0.67	0.99	0.57	1	0.41	1

Figura 1.38 Perfil de l'execució de la versió 4 de l'arxiu 3DFFT_omp.c amb vuit CPUs