

PAR Laboratory Deliverable

Lab 1: Experimental setup and tools

Víctor Asenjo Carvajal
Ferran De La Varga Antoja

Spring 2021-22



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Índex

Session 1: Experimental setup	1
1.1 Node architecture and memory	1
1.2 Strong vs. weak scalability	4
Session 2: Systematically analysing task decompositions with Tareador	7
2.1 Analysis of task decompositions for 3DFFT	7
Session 3: Understanding the execution of OpenMP programs	16
3.1 Obtaining parallelisation metrics for 3DFFT using model factors	16
3.1.1 Initial version	18
3.1.2 Improving ϕ	20
3.1.3 Reducing parallelisation overheads	22
4. Conclusions	26

Session 1: Experimental setup

1.1 Node architecture and memory

Per saber l'arquitectura de boada, primer de tot hem executat la comanda: *lscpu*. Aquesta comanda llista totes les característiques del hardware (un truc per enrecordar-se és ls com la comanda per llistar els arxius d'un directori, més el que volem llistar: la CPU. *list_cpu = lscpu*).

```
par1302@boada-1:~/PAR/sessions/lab1$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                 24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):              2
NUMA node(s):          2
Vendor ID:              GenuineIntel
CPU family:             6
Model:                 44
Model name:             Intel(R) Xeon(R) CPU           E5645  @ 2.40GHz
Stepping:               2
CPU MHz:                1600.078
CPU max MHz:            2395.0000
CPU min MHz:            1596.0000
BogoMIPS:               4800.23
Virtualization:         VT-x
L1d cache:              32K
L1i cache:              32K
L2 cache:               256K
L3 cache:               12288K
NUMA node0 CPU(s):      0,2,4,6,8,10,12,14,16,18,20,22
NUMA node1 CPU(s):      1,3,5,7,9,11,13,15,17,19,21,23
Flags:                   fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                        rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc
                        dcm pcid sse4_1 sse4_2 popcnt lahf_lm pti ssbd ibrs ilrb stibp tpr_shadow v
par1302@boada-1:~/PAR/sessions/lab1$
```

Figura 1: Captura de pantalla del terminal execució comanda *lscpu*

Fem el mateix però amb la comanda *lstopo* que ens mostra la topologia del sistema (podem emprar el mateix mnemotècnic que abans *list_topology = lstopo*). Però per una millor visualització d'aquestes dades executem:

lstopo --of fig map.fig

Amb aquesta comanda podem veure les dades gràficament de manera més clara i endreçada. Només amb *lstopo* és molt abstracte.

Per obrir aquest arxiu generat (map.fig) hem executat *xfig map.fig* i amb aquest programa es pot exportar la següent figura.

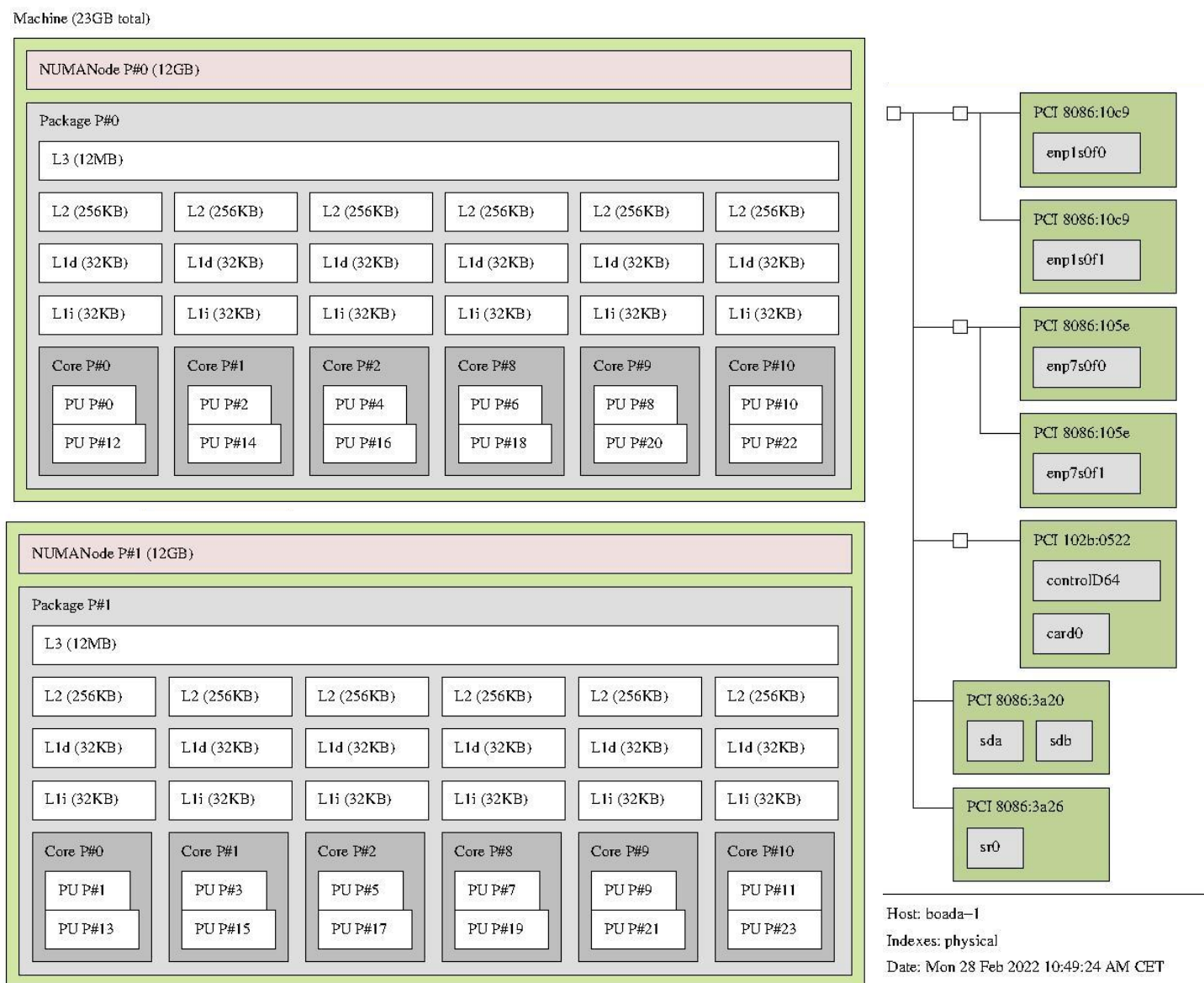


Figura 2: map.fig → *lstopo --of fig map.fig* (arquitectura memòria sockets)

Podem observar que l'arquitectura de la memòria de cada socket és exactament igual excepte per la paritat dels blocs d'aquesta.

Comprovem que la màquina marca 23 GB total perquè cada socket suma 11.5 GB (que el diagrama arrodoneix a 12 GB), donant de resultat els 23 GB totals de la memòria que marca el diagrama.

Per tant, les característiques de boada són:

	boada-1 to boada-4
Number of sockets per node	2
Number of cores per node	6
Number of threads per core	2
Maximum core frequency	2395 MHz
L1-I cache size (per-core)	32 kB
L1-D cache size (per-core)	32 kB
L2 cache size (per-core)	256 kB
Last-level cache size (per-socket)	12288 kB
Main memory size (per socket)	12 GB
Main memory size (per node)	23 GB

Figura 3: Taula de les característiques del sistema

Així doncs, podem concloure a mode de resum amb aquesta taula (Figura 3) la informació tècnica dels mòduls que utilitzarem per als laboratoris de les següents sessions i dels que podem extreure el següent:

$$Threads_{total} = 2 \frac{sockets}{node} \cdot 6 \frac{core}{socket} \cdot 2 \frac{threads}{core} = 24 \frac{threads}{node}$$

a una freqüència màxima propera a 2.4 GHz.

Com podem observar, el sistema ens mostra com si tingués 24 CPUs, però en realitat només en té 6 físiques i cada core pot processar al mateix temps 2 threads (multithreading de Intel).

Els primers quatre nodes presenten la mateixa arquitectura pel que hauríem d'obtenir els mateixos resultats en aquests quatre primers independentment de quin sigui el que estem utilitzant per a l'execució del programa, proves...

1.2 Strong vs. weak scalability

Per executar el programa `pi_omp` a boada de manera *interactive* i *queued*, hem utilitzat les comandes:

```
./run-omp.sh pi_omp 1000000000 x
```

```
sbatch ./submit-omp.sh pi_omp x
```

respectivament, on `x` era 1, 2, 4 o 8 threads.

#threads	Interactive				Queued			
	user	system	elapsed	% of CPU	user	system	elapsed	% of CPU
1	3.94	0.00	3.94	99	3.93	0.01	3.96	99
2	8.00	0.00	4.01	199	3.95	0.00	1.99	197
4	8.01	0.01	4.02	199	3.97	0.00	1.01	392
8	8.01	0.03	4.03	199	4.17	0.00	0.54	769

Figura 4: Resultats obtinguts d'executar `pi_omp` de manera *interactive* i *queued* amb diferent número de threads

En la Figura 4 podem veure els resultats que hem obtingut.

Si `pi_omp` s'executa de manera *interactive*, a mesura que afegim threads al programa, aquest cada cop s'executa més lent. Això és degut al fet que `pi_omp` s'està executant alhora amb altres programes. En canvi, si executem el programa de manera *queued*, com que el processador dedica tot el temps només a `pi_omp`, l'augment de threads té un gran impacte i millora significativament el temps d'execució, passant de 3.96 s si només té un thread a 0.54 amb 8 threads.

També es pot observar que a l'*interactive* només hi ha un core disponible. És per això que al passar d'un thread a dos, es passa de 99% a 199% de CPU perquè tal com hem vist a l'apartat anterior l'arquitectura com a màxim admet 2 threads per core. D'altra banda, en el mode *queued* no hi és aquesta restricció i es poden utilitzar més d'una CPU.

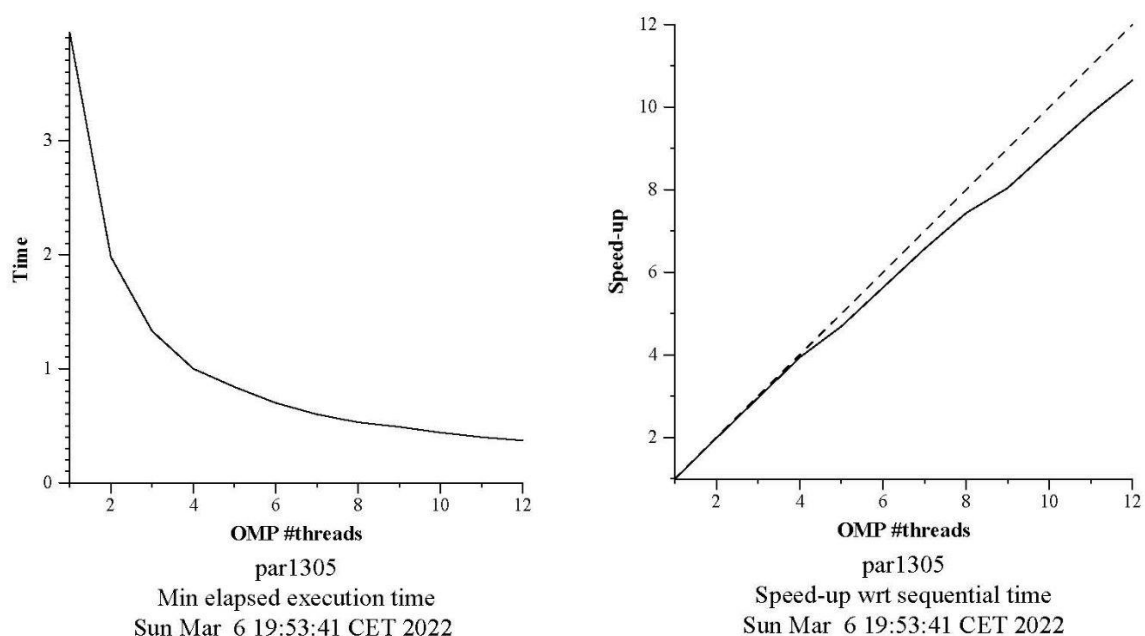


Figura 5: Variació entre el temps d'execució i el número de threads i l'*speed up* de *pi_omp* versió *strong scalability*

Com podem veure a la Figura 5, el temps d'execució del programa *pi_omp* versió *strong scalability* disminueix a mesura que augmenta el número de threads i l'*speed up* també augmenta però per sota del llindar ideal degut al *overhead*.

Si augmentem el número de threads fins a 24 (Figura 6), observem com hi ha una pujada de temps a l'execució amb 12 threads i l'*speed up* baixa sobtadament; després es recupera. Això és degut a que hi ha un canvi de node (canvi de context) que provoca molt de *overhead*.

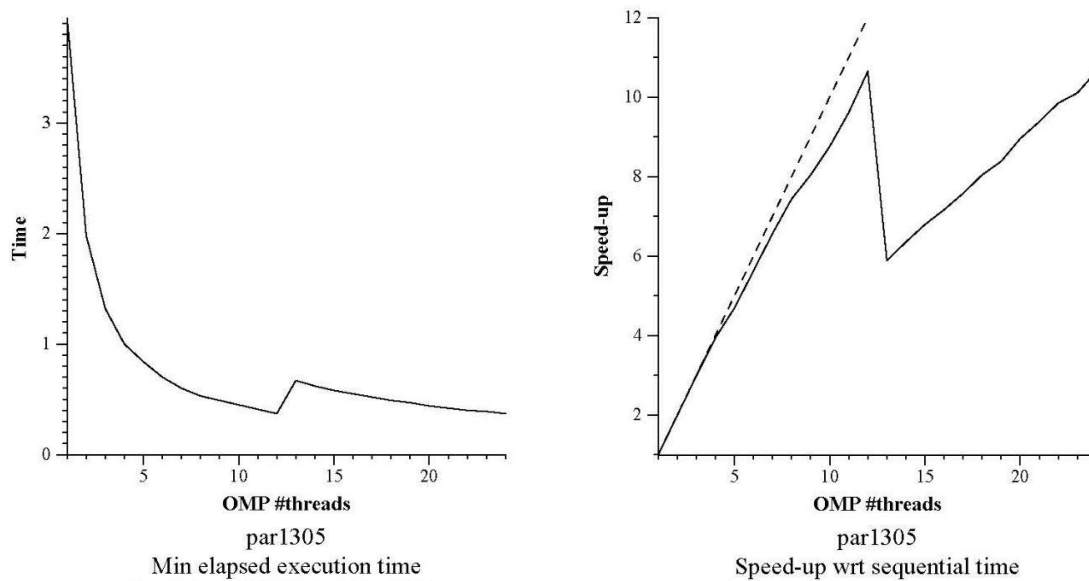


Figura 6: Variació entre el temps d'execució i el número de threads (fins a 24) i l'*speed up* de *pi_omp* versió *strong scalability*

Si ara observem la versió *weak scalability* (Figura 7), l'eficiència de paral·lelisme decreix lleugerament, però podríem dir que es manté igual. A mesura que s'augmenten els threads s'augmenta també la mida del problema i això fa que no es millori en eficiència.

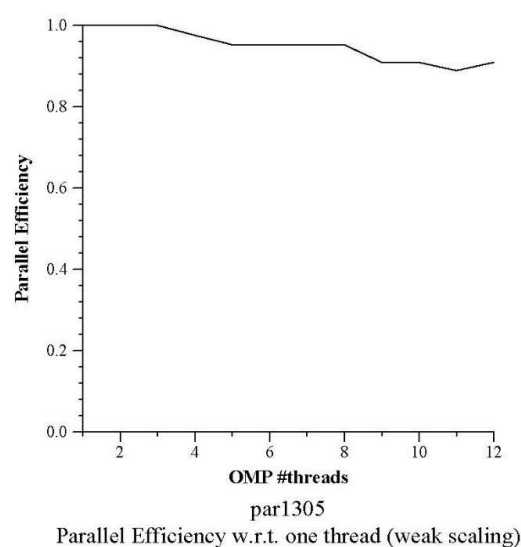


Figura 7: Variació entre l'eficiència i el número de threads de *pi_omp* versió *weak scalability*

Session 2: Systematically analysing task decompositions with Tareador

2.1 Analysis of task decompositions for 3DFFT

Version	T_1 (ns)	T_∞ (ns)	Parallelism
seq	639780001	639707001	1,000114115
v1	639780001	639707001	1,000114115
v2	639780001	361439001	1,770091217
v3	639780001	154939001	4,754360254
v4	639780001	64614001	9,901569182
v5	639780001	53269001	12,01036229

Figura 8: Taula amb les dades de les diferents versions del programa

En aquesta sessió ens hem familiaritzat amb Tareador. Aquesta eina ajuda de manera gràfica a veure el potencial de millora que pot arribar a tenir un programa determinat. Si delimitem bé el codi a analitzar per potenciar la descomposició de tasques a paral·lelitzar, es pot reduir molt el temps d'execució d'un programa.

El codi proporcionat és `3dfft_tar.c` que significa Fast Fourier Transform. Estudiem doncs quina seria la millor estratègia de paral·lelització a seguir amb un i infinits processadors. Òbviament, no tenim infinits processadors, però en el cas ideal que sí en tinguéssim es calcula el temps d'execució fins que aquest tendeix a un límit i no millora (o la millora és pràcticament nul·la).

Observem a la taula que la versió 1 i la seqüencial són la mateixa, és a dir, no hi ha paral·lelisme. El T_1 correspon al temps d'execució a un processador pel que el codi actua seqüencialment i és per aquest motiu que sempre és el mateix independentment. En canvi pel que fa al temps ideal en infinits processadors l'hem obtingut arrel de fer

simulacions amb el màxim nombre permès: 128. La versió 1 respecte la 5 és 12 vegades més lenta. Dit d'una altra manera, en el que tarda en executar-se v1 podríem arribar a executar 12 vegades v5. De v1 a v2 notem una diferència de velocitat propera al 77%.

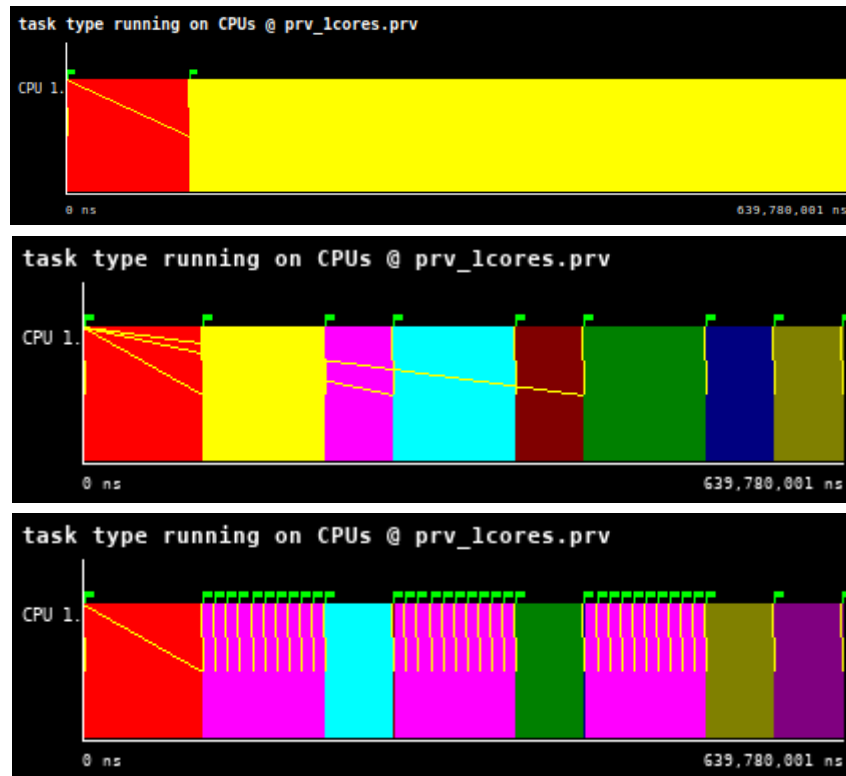


Figura 9: Execució seqüencial de les versions seq, v1 i v2 de 3dfft_tar per obtenir T_1

A continuació podem veure com han anat variant els grafs de dependències segons les versions del programa.

Pel que fa al graf de la versió 5, aquest no es veu detalladament perquè és molt gran, però sí que es pot apreciar una idea de la gran quantitat de tasques que hi ha.

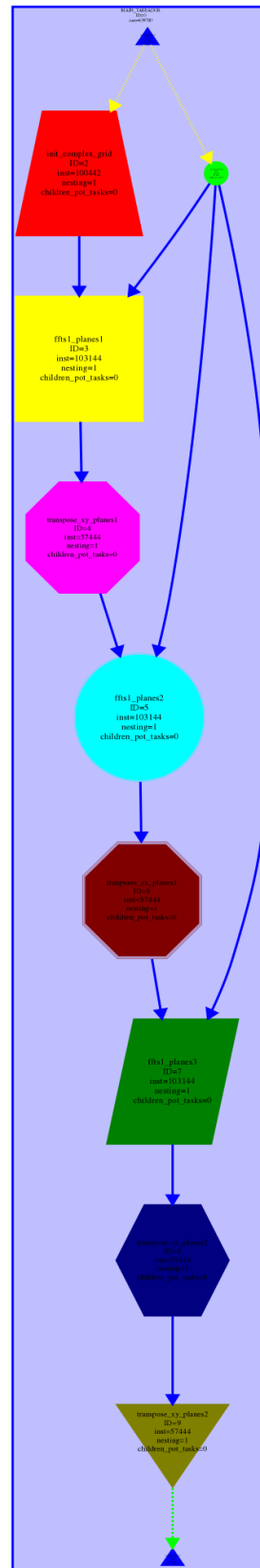


Figura 10: Graff de dependències de v1

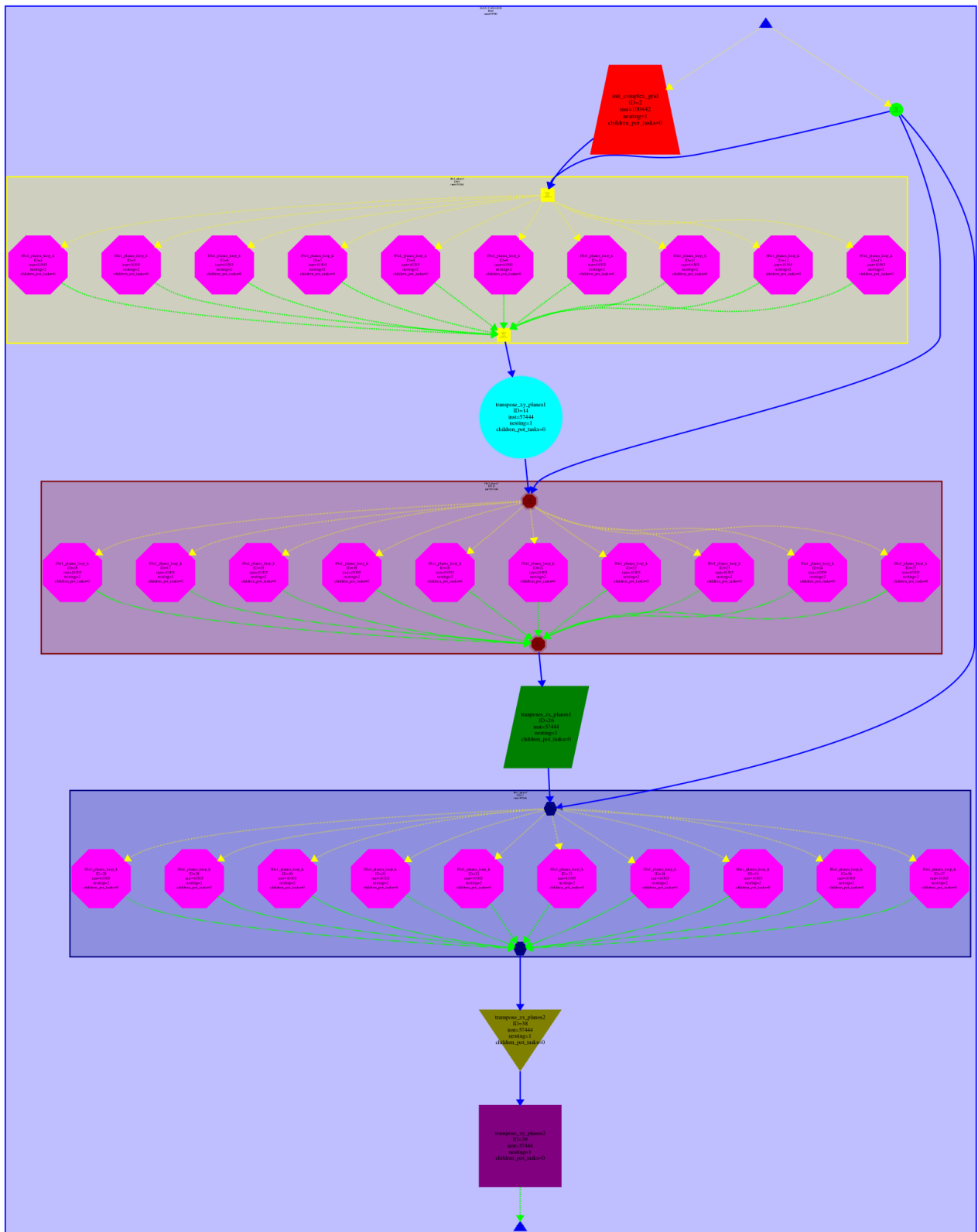


Figura 11: Graf de dependències de v2

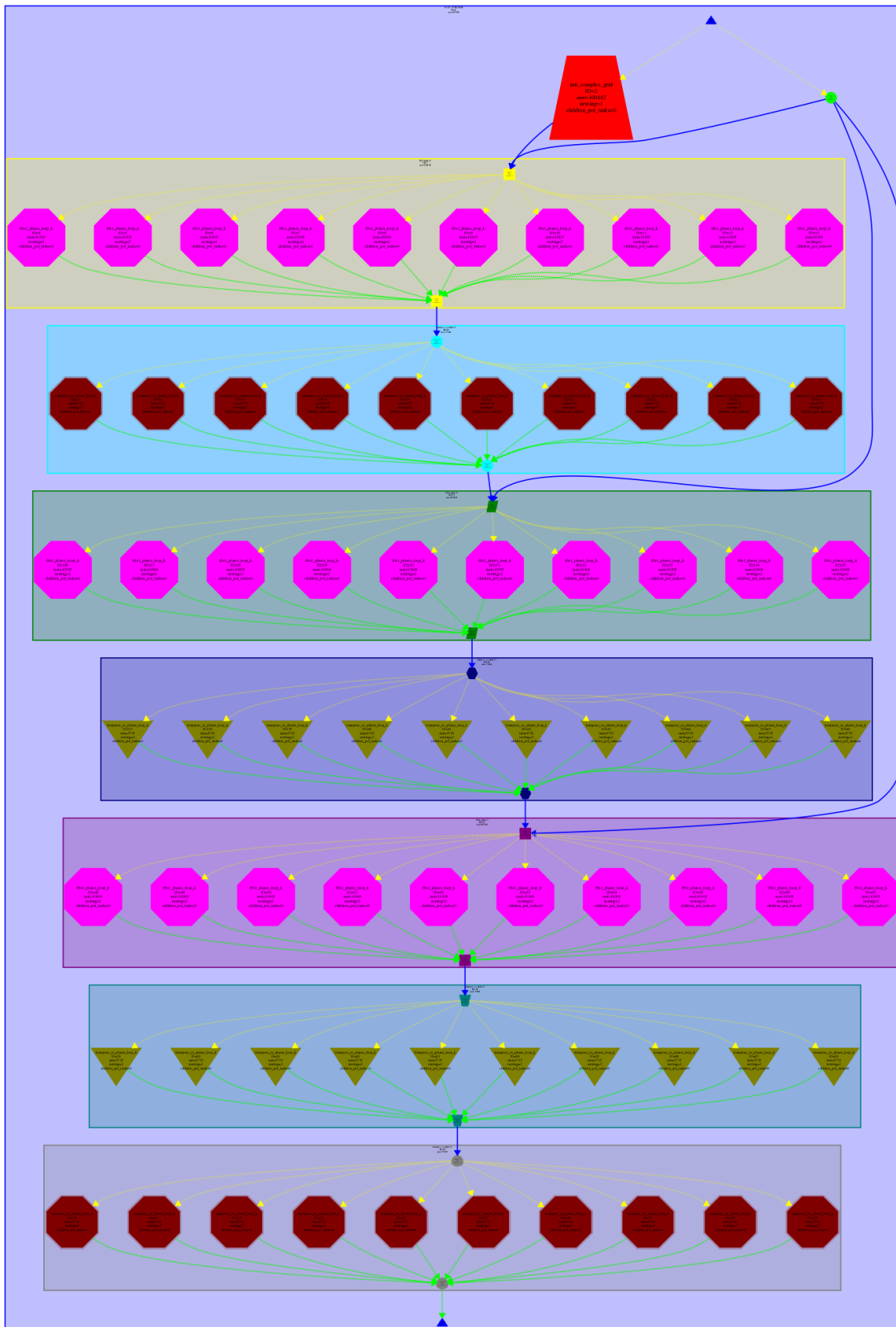


Figura 12: Graf de dependències de v3

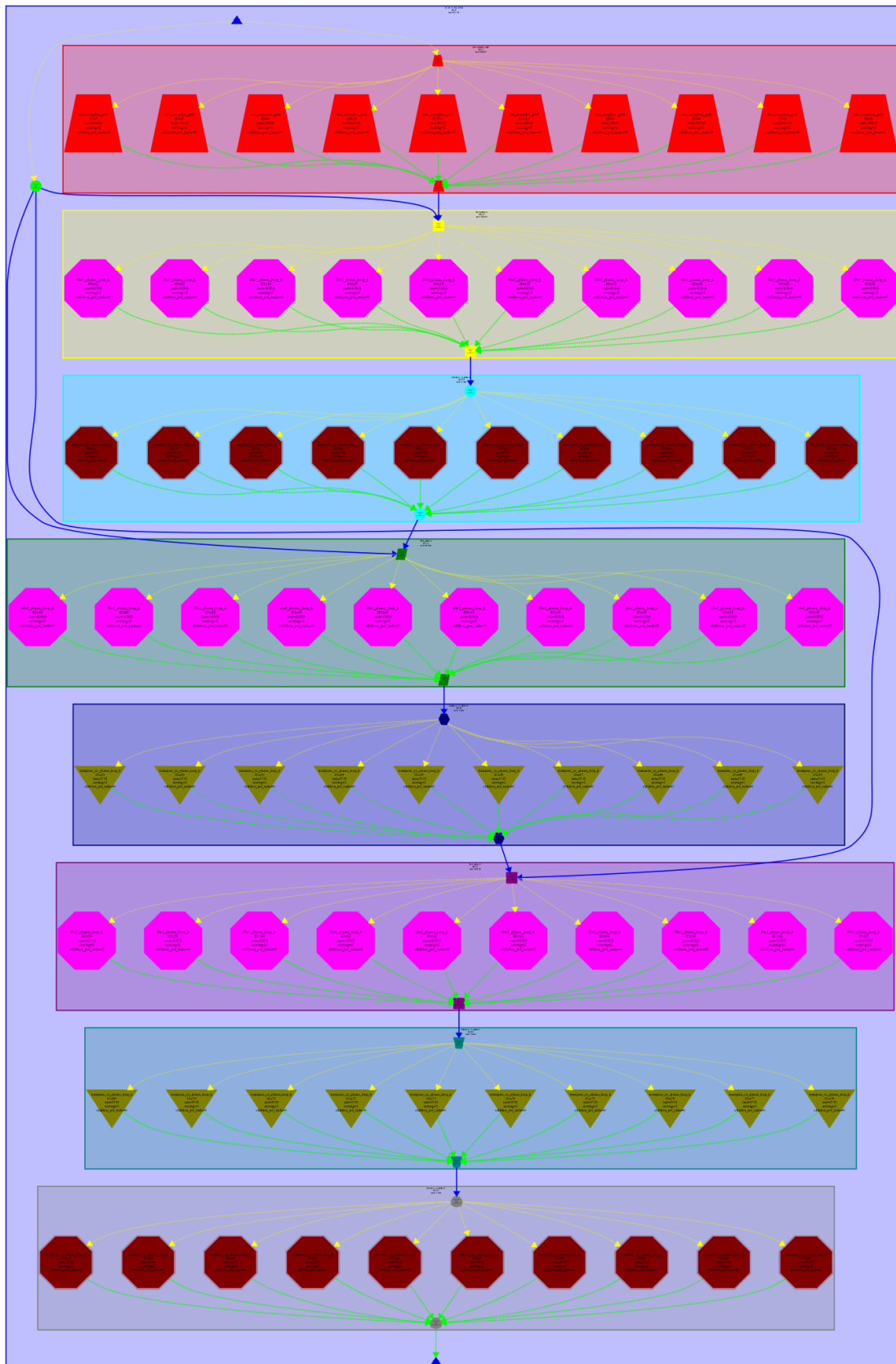


Figura 13: Graf de dependències de v4

Figura 14: Esquema del graf de dependències de v5 (molt gran i difícil d'entendre; granularitat massa fina)

num. de CPU	Temps (ns) v4	Speed-up v4
1	639780001	1
2	320257001	1,997708088
4	191992001	3,332326335
8	127992001	4,998593631
16	64614001	9,901569182
32	64614001	9,901569182

Figura 15: Temps i speed-up de la versió 4 segons el número de CPUs utilitzades

Scalability v4

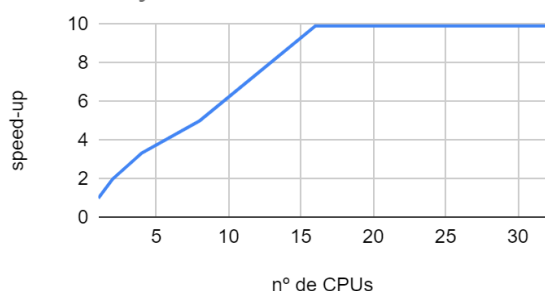


Figura 16

Scalability v4

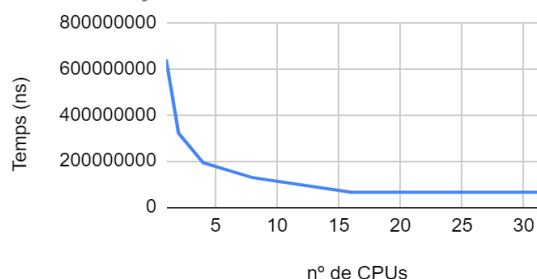


Figura 17

Com podem veure en les gràfiques de v4 (Figura 16 i 17), a partir de 16 processadors ja s'arriba al temps del camí crític (T_{∞}) i és per això que el temps i l'speed-up es mantenen constant per molt que s'afegeixin més CPUs. Més o menys l'speed-up creix de manera lineal fins a arribar al llindar de 10. En canvi, en la versió 5 del programa, a mesura que es va augmentant el número de CPUs, l'speed-up va creixent de manera logarítmica. És per això que la gràfica tendeix a que l'speed-up sigui més gran però cada cop la millora serà més petita, i és per això que si això fos el món real s'hauria de valorar si surt a compte el guany en temps i el cost econòmic de posar més processadors.

num. de CPU	Temps (ns) v5	Speed-up v5
1	639780001	1
2	309271633	2,068666935
4	162268001	3,942736689
8	90525001	7,067439867
16	69010001	9,270830195
32	59912001	10,67866188

Figura 18: Temps i speed-up de la versió 5 segons el número de CPUs utilitzades

Scalability v5

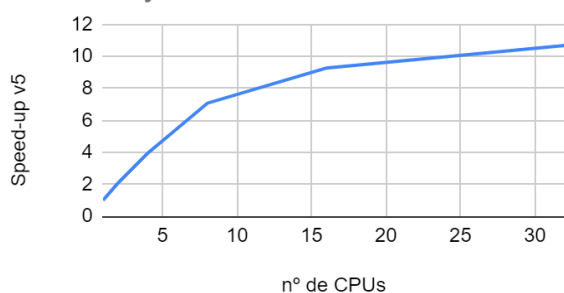


Figura 19

Scalability v5

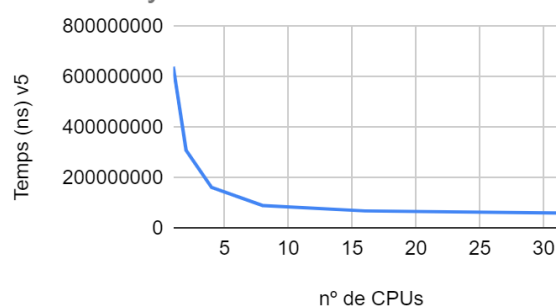


Figura 20

La diferència d'speed-up entre v4 i v5 és mínima i per tant no seria necessari una granularitat tan fina com la de v5 ja que amb la granularitat de v4 l'speed-up és casi el mateix.


```

26
27 void init_complex_grid(fftwf_complex in_fftw[][N][N]) {
28     int k,j,i;
29
30     for (k = 0; k < N; k++) {
31         //tareador_start_task("init_complex_grid_k");
32
33         for (j = 0; j < N; j++) {
34
35             for (i = 0; i < N; i++)
36             {
37                 tareador_start_task("init_complex_grid_i");
38
39                 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));
40                 in_fftw[k][j][i][1] = 0;
41                 #if TEST
42                 out_fftw[k][j][i][0]= in_fftw[k][j][i][0];
43                 out_fftw[k][j][i][1]= in_fftw[k][j][i][1];
44                 #endif
45                 tareador_end_task("init_complex_grid_i");
46             }
47         }
48     }
49     //tareador_end_task("init_complex_grid_k");
50 }
51
52
53

```

Figura 21: Codi de la primera funció de v5

Hem anat “jugant” amb diferents nivells de granularitat de tasca per veure i entendre millor quina seria la millor estratègia de paral·lelització. El comentari del loop k pertany a la versió 4. A més a la versió 5 hem acotat el fragment a analitzar de la resta de funcions al loop més intern de cada funció el que ha donat com a resultat un graf de dependències massa difícil d’analitzar, la granularitat era massa fina.

Session 3: Understanding the execution of OpenMP programs

3.1 Obtaining parallelisation metrics for 3DFFT using model factors

Durant aquesta sessió, aprendrem a utilitzar Paraver. Aquesta eina ens ajudarà a veure de manera gràfica l'execució de certs paràmetres dels nostres programes i la rellevància d'aquests.

Hem executat el codi de l'arxiu 3dfft_omp.c amb l'script d'Extrae. Això ens genera un fitxer .prv que ens permet obrir el programa Paraver i analitzar la traça delimitada en qüestió. Un cop obert el programa i seleccionada la traça, hem activat les flags i hem utilitzat la configuració més adequada per obtenir els valors de paral·lelisme.

Per omplir la taula de més endavant hem emprat les fórmules següents:

OMP_implicit_tasks_profile.cfg

$$\Phi = \frac{T_{par}}{(T_{seq} + T_{par})}$$

$$ideal S_8 = \frac{1}{(1-\Phi) + (\Phi/8)}$$

Aquesta taula és a mode de resum però analitzem-la, mirem-ho en detall. Per la llei d'Amdahl, com hi ha parts que no estan paral·lelitzades, l'speed up no és proporcional al nombre de cores. Al començament del codi posa fake parallelism i al final també per delimitar què estudiem. A la part central tenim menys del 75% paral·lelitzat amb un speed up de com a màxim 4. Un responsable és la secció paral·lela.

Si anem a la segona versió ja anem al 99%.

En la versió inicial, per fer l'anàlisi paral·lel del 74.1%, per què l'speed-up és 1.79 i no 2.857? L'eficiència global és la multiplicació dels seus components i aquests el mateix amb els que el formen. No és un problema de que hi hagi threads que no treballen, de fet en el Paraver podem observar que tots els threads treballen més o menys igual. Llavors el *usefull code* ens marca el codi útil sense overheads. Aquesta és l'eficiència dels processadors executant els codis en paral·lel.

Potser no calien tantes tasques per 12 cores. Granularitat massa fina, massa overhead.

Pel que fa a la Scalability si afegim processadors, la memòria queda saturada. Les instruccions per cicle van baixant.

Version	ϕ	ideal S_8	T1 (s)	T_8 (s)	real S_8
initial version in 3dfft_omp.c	0.741	2.857	2.25	1.26	1.79
new version with improved ϕ	0.996	7.782	2.30	0.80	3.26
final version with reduced parallelisation overheads	0.997	7.837	2.12	0.38	5.63

Figura 22: Taula resum de les versions del programa

3.1.1 Initial version

Overview of whole program execution metrics:

Number of processors	1	2	4	6	8	10	12
Elapsed time (sec)	2.25	1.78	1.28	1.25	1.26	1.28	1.30
Speedup	1.00	1.27	1.75	1.79	1.79	1.75	1.72
Efficiency	1.00	0.63	0.44	0.30	0.22	0.18	0.14

Overview of the Efficiency metrics in parallel fraction:

Number of processors	1	2	4	6	8	10	12
Parallel fraction	74.10%						
Global efficiency	95.97%	66.98%	57.40%	40.21%	30.12%	23.47%	18.92%
-- Parallelization strategy efficiency	95.97%	89.71%	84.45%	83.31%	81.47%	81.15%	80.21%
-- Load balancing	100.00%	99.50%	98.47%	98.45%	98.29%	98.33%	98.20%
-- In execution efficiency	95.97%	90.16%	85.77%	84.62%	82.88%	82.53%	81.68%
-- Scalability for computation tasks	100.00%	74.66%	67.96%	48.26%	36.97%	28.93%	23.58%
-- IPC scalability	100.00%	90.46%	77.20%	62.51%	52.42%	47.06%	41.25%
-- Instruction scalability	100.00%	99.46%	98.45%	97.46%	96.48%	95.53%	94.59%
-- Frequency scalability	100.00%	82.99%	89.42%	79.23%	73.10%	64.36%	60.44%

Statistics about explicit tasks in parallel fraction

Number of processors	1	2	4	6	8	10	12
Number of explicit tasks executed (total)	17920.0	35840.0	71680.0	107520.0	143360.0	179200.0	215040.0
LB (number of explicit tasks executed)	1.0	0.97	0.95	0.95	0.96	0.88	0.9
LB (time executing explicit tasks)	1.0	1.0	0.98	0.98	0.98	0.98	0.98
Time per explicit task (average)	88.87	59.56	32.72	30.75	30.11	30.8	31.49
Overhead per explicit task (synch %)	0.49	10.26	16.82	18.43	20.97	21.46	22.74
Overhead per explicit task (sched %)	3.72	1.23	1.61	1.6	1.76	1.75	1.9
Number of taskwait/taskgroup (total)	1792.0	1792.0	1792.0	1792.0	1792.0	1792.0	1792.0

Figura 23: modelfactors.out versió inicial

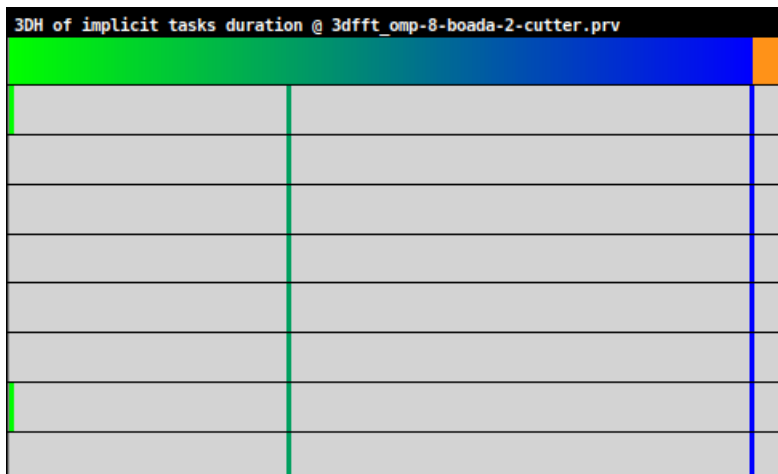


Figura 24: histogrames amb la durada de les tasques implícites versió inicial

	Running	Synchronization	Scheduling and Fork/Join
THREAD 1.1.1	89.56 %	8.80 %	1.64 %
THREAD 1.1.2	82.58 %	17.42 %	0.00 %
THREAD 1.1.3	82.38 %	17.62 %	0.00 %
THREAD 1.1.4	82.64 %	17.36 %	0.00 %
THREAD 1.1.5	81.01 %	17.04 %	1.95 %
THREAD 1.1.6	82.64 %	17.36 %	0.00 %
THREAD 1.1.7	77.96 %	15.65 %	6.39 %
THREAD 1.1.8	82.59 %	17.41 %	0.00 %
Total	661.34 %	128.66 %	10.00 %
Average	82.67 %	16.08 %	1.25 %
Maximum	89.56 %	17.62 %	6.39 %
Minimum	77.96 %	8.80 %	0.00 %
StDev	3.01 %	2.81 %	2.09 %
Avg/Max	0.92	0.91	0.20

Figura 25: 2D Thread state profile

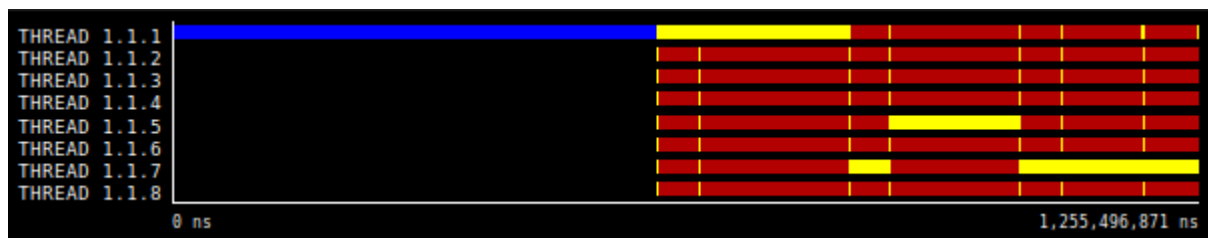


Figura 26: Timeline de l'execució amb 8 processadors.

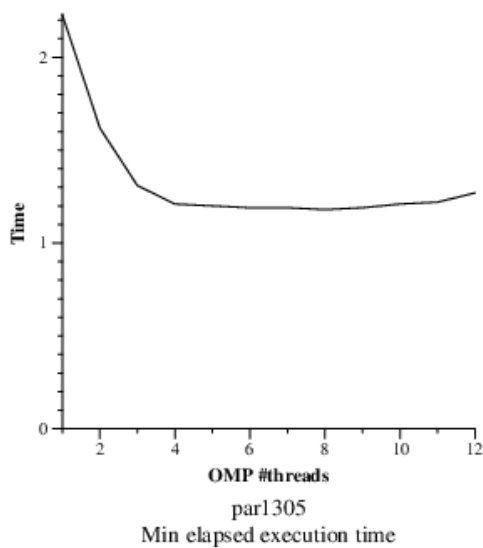


Figura 27

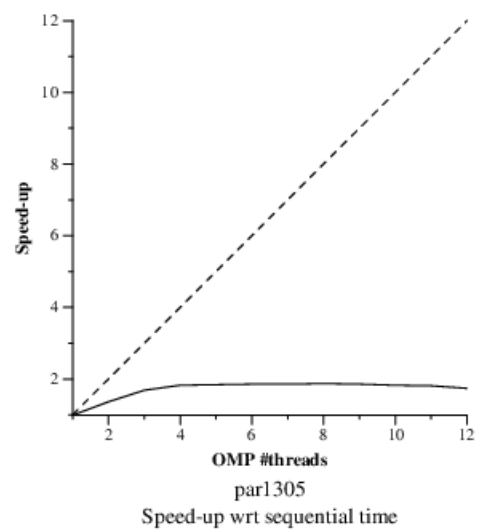


Figura 28

3.1.2 Improving ϕ

Hem fet una millora afegint paral·lelisme a la funció que tenia menys ϕ , la `init_complex_grid` tot passant d'un 74,1% de fragment de codi paral·lelitzable a un 99.6%

Overview of whole program execution metrics:								
Number of processors	1	2	4	6	8	10	12	
Elapsed time (sec)	2.30	1.62	0.82	0.72	0.70	0.71	0.73	
Speedup	1.00	1.41	2.80	3.18	3.26	3.23	3.16	
Efficiency	1.00	0.71	0.70	0.53	0.41	0.32	0.26	

Overview of the Efficiency metrics in parallel fraction:								
Number of processors	1	2	4	6	8	10	12	
Parallel fraction	99.55%							
Global efficiency	96.62%	68.06%	67.44%	51.01%	39.21%	31.10%	25.37%	
-- Parallelization strategy efficiency	96.62%	92.13%	85.09%	82.58%	80.75%	79.94%	78.10%	
-- Load balancing	100.00%	99.26%	98.26%	98.66%	98.39%	98.03%	98.13%	
-- In execution efficiency	96.62%	92.81%	86.59%	83.71%	82.06%	81.54%	79.58%	
-- Scalability for computation tasks	100.00%	73.88%	79.26%	61.77%	48.56%	38.90%	32.49%	
-- IPC scalability	100.00%	91.47%	89.28%	79.45%	68.65%	63.21%	56.89%	
-- Instruction scalability	100.00%	99.59%	98.81%	98.04%	97.29%	96.54%	95.81%	
-- Frequency scalability	100.00%	81.11%	89.84%	79.30%	72.71%	63.75%	59.60%	

Statistics about explicit tasks in parallel fraction								
Number of processors	1	2	4	6	8	10	12	
Number of explicit tasks executed (total)	20480.0	40960.0	81920.0	122880.0	163840.0	204800.0	245760.0	
LB (number of explicit tasks executed)	1.0	1.0	0.97	0.99	0.97	0.96	0.98	
LB (time executing explicit tasks)	1.0	1.0	0.98	0.99	0.98	0.98	0.98	
Time per explicit task (average)	107.51	72.81	33.93	29.04	27.72	27.69	27.64	
Overhead per explicit task (synch %)	0.42	7.57	16.0	19.44	22.06	23.17	25.95	
Overhead per explicit task (sched %)	3.08	0.99	1.55	1.68	1.8	1.94	2.1	
Number of taskwait/taskgroup (total)	2048.0	2048.0	2048.0	2048.0	2048.0	2048.0	2048.0	

Figura 29: modelfactors.out improved version

Com podem veure, l'speed-up ha millorat, passant d'un 1.79 a un 3.26.

En la següent taula podem veure en blau el % de temps que els threads estan executant codi útil (més o menys un 80% del temps) i en verd el % de temps que s'estan sincronitzant (més o menys un 18%).

	Running	Synchronization	Scheduling and Fork/Join
THREAD 1.1.1	81.14 %	17.45 %	1.40 %
THREAD 1.1.2	81.59 %	18.40 %	0.00 %
THREAD 1.1.3	81.89 %	18.11 %	0.00 %
THREAD 1.1.4	81.54 %	18.46 %	0.00 %
THREAD 1.1.5	80.17 %	16.90 %	2.94 %
THREAD 1.1.6	81.52 %	18.47 %	0.01 %
THREAD 1.1.7	76.80 %	15.95 %	7.25 %
THREAD 1.1.8	81.47 %	18.53 %	0.00 %
Total	646.11 %	142.27 %	11.62 %
Average	80.76 %	17.78 %	1.45 %
Maximum	81.89 %	18.53 %	7.25 %
Minimum	76.80 %	15.95 %	0.00 %
StDev	1.57 %	0.88 %	2.41 %
Avg/Max	0.99	0.96	0.20

Figura 30: 2D Thread state profile

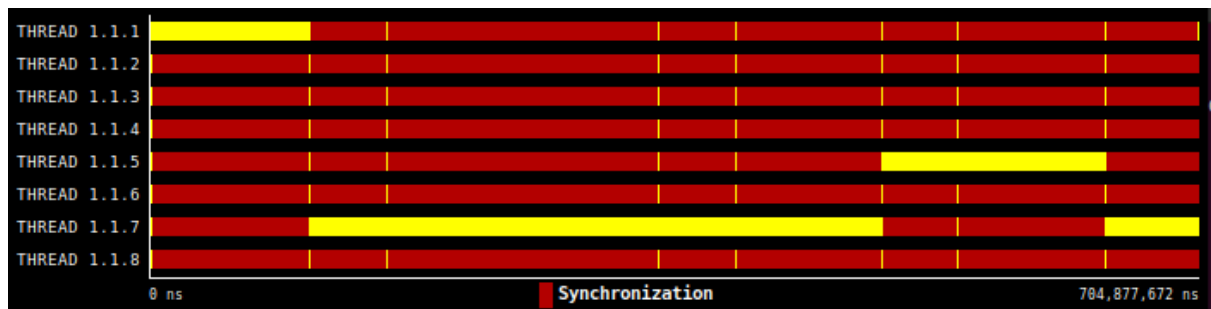


Figura 31: Timeline de l'execució amb 8 processadors.

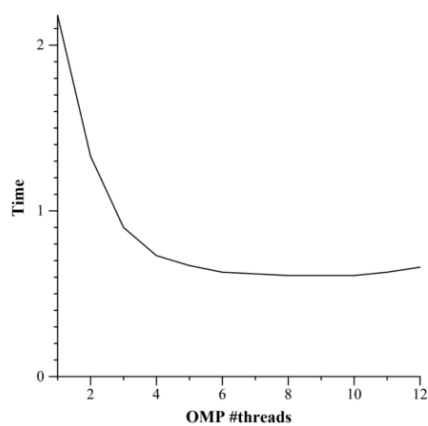


Figura 32

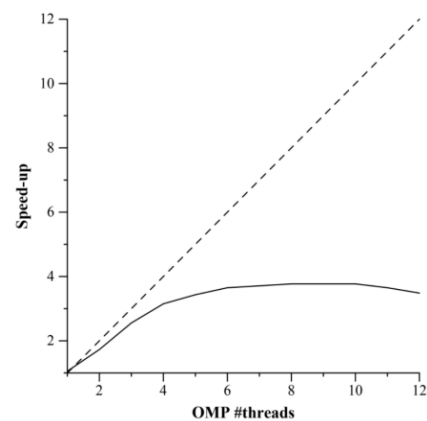


Figura 33

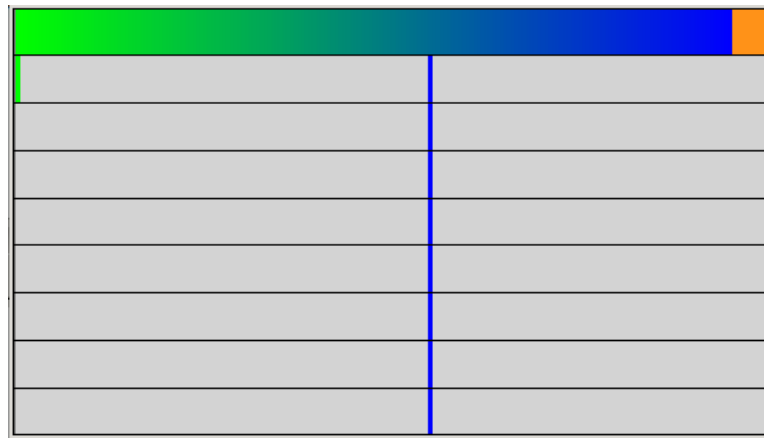


Figura 34: histogrames amb la durada de les tasques implícites versió millorada

3.1.3 Reducing parallelisation overheads

Aquesta versió redueix l'*overhead* i té un gran impacte en el rendiment global obtingut. Ho aconseguim augmentant la granularitat de les tasques. D'aquesta manera hi ha menys canvis de context al no haver de treballar amb tasques tan fines i repartides entre els processadors.

Overview of whole program execution metrics:								
Number of processors	1	2	4	6	8	10	12	
Elapsed time (sec)	2.12	1.22	0.62	0.46	0.38	0.34	0.31	
Speedup	1.00	1.74	3.45	4.58	5.63	6.29	6.79	
Efficiency	1.00	0.87	0.86	0.76	0.70	0.63	0.57	

Overview of the Efficiency metrics in parallel fraction:								
Number of processors	1	2	4	6	8	10	12	
Parallel fraction	99.97%							
Global efficiency	99.90%	87.10%	86.24%	76.31%	70.39%	62.90%	56.57%	
-- Parallelization strategy efficiency	99.90%	98.51%	96.26%	97.35%	97.48%	97.13%	96.44%	
-- Load balancing	100.00%	99.28%	97.51%	97.82%	98.17%	98.04%	97.04%	
-- In execution efficiency	99.90%	99.22%	98.72%	99.52%	99.29%	99.08%	99.38%	
-- Scalability for computation tasks	100.00%	88.41%	89.58%	78.38%	72.22%	64.75%	58.66%	
-- IPC scalability	100.00%	95.77%	90.73%	82.93%	76.55%	72.37%	65.06%	
-- Instruction scalability	100.00%	100.00%	99.99%	99.99%	99.99%	99.98%	99.98%	
-- Frequency scalability	100.00%	92.31%	98.74%	94.53%	94.35%	89.49%	90.18%	

Statistics about explicit tasks in parallel fraction								
Number of processors	1	2	4	6	8	10	12	
Number of explicit tasks executed (total)	80.0	160.0	320.0	480.0	640.0	800.0	960.0	
LB (number of explicit tasks executed)	1.0	1.0	0.98	0.98	0.98	0.89	0.94	
LB (time executing explicit tasks)	1.0	0.99	1.0	0.98	0.98	0.99	0.99	
Time per explicit task (average)	26521.56	14999.03	7400.45	5638.62	4589.85	4094.86	3766.66	
Overhead per explicit task (synch %)	0.0	1.48	3.84	2.66	2.52	2.86	3.61	
Overhead per explicit task (sched %)	0.09	0.02	0.03	0.03	0.03	0.03	0.03	
Number of taskwait/taskgroup (total)	8.0	8.0	8.0	8.0	8.0	8.0	8.0	

Figura 35: modelfactors.out reducing parallelisation overheads version

L'speed up ha millorat. Ha passat de 3.26 de la versió millorada a 5.63. L'eficiència ha millorat. Per exemple amb 8 threads, en la versió inicial l'eficiència era un 0,22, en la versió millorada era un 0,4 i ara ha passat a ser 0,70.

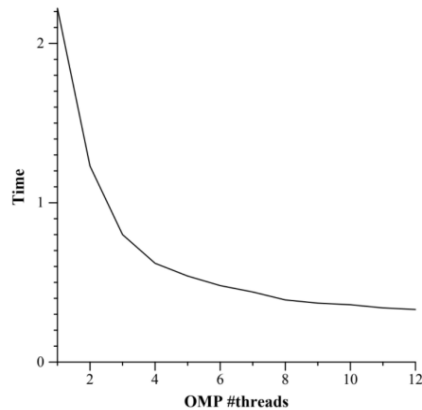


Figura 36

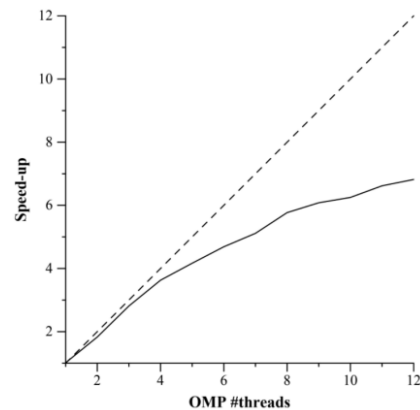


Figura 37

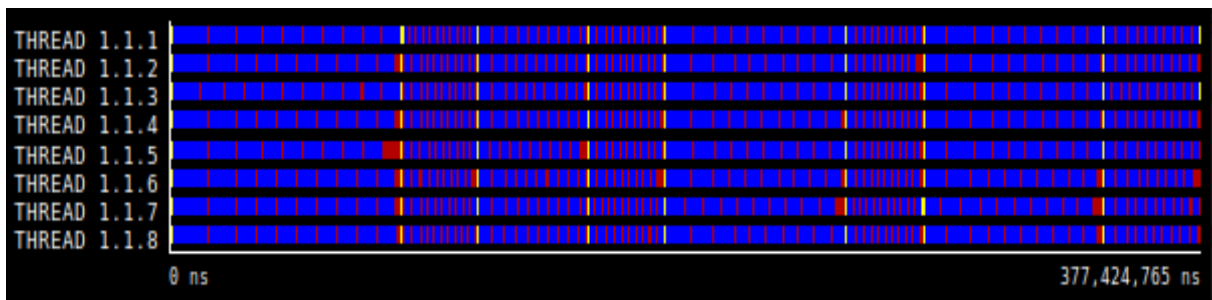
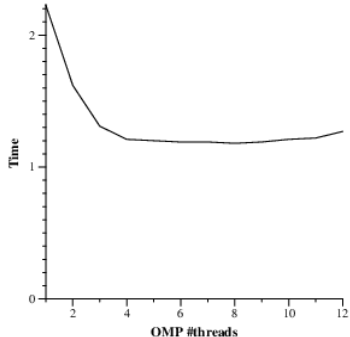
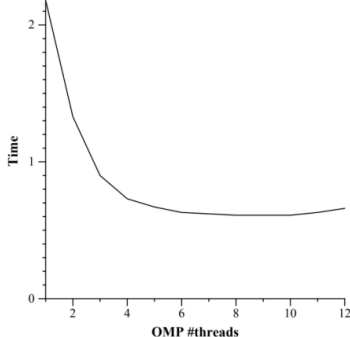
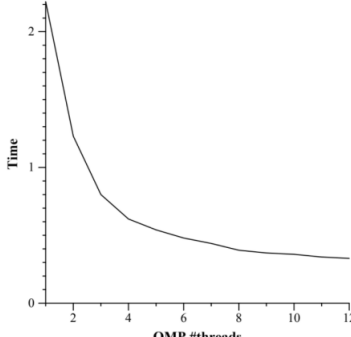
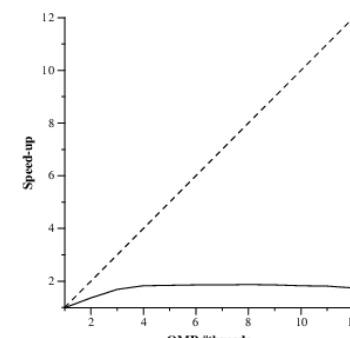
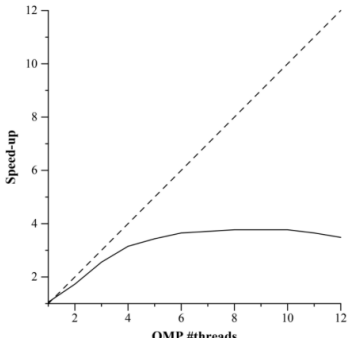
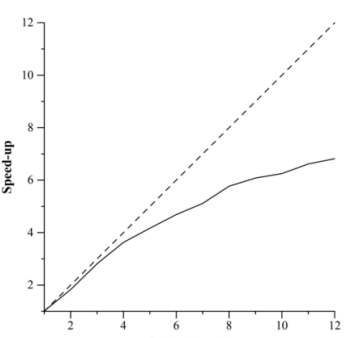


Figura 38: Timeline de l'execució amb 8 processadors.

	Running	Synchronization	Scheduling and Fork/Join
THREAD 1.1.1	99.30 %	0.63 %	0.08 %
THREAD 1.1.2	97.42 %	2.57 %	0.01 %
THREAD 1.1.3	98.91 %	1.03 %	0.06 %
THREAD 1.1.4	97.83 %	2.16 %	0.01 %
THREAD 1.1.5	96.46 %	3.50 %	0.04 %
THREAD 1.1.6	95.82 %	4.18 %	0.01 %
THREAD 1.1.7	96.75 %	3.15 %	0.10 %
THREAD 1.1.8	97.60 %	2.40 %	0.01 %
Total	780.08 %	19.62 %	0.30 %
Average	97.51 %	2.45 %	0.04 %
Maximum	99.30 %	4.18 %	0.10 %
Minimum	95.82 %	0.63 %	0.01 %
StDev	1.11 %	1.12 %	0.03 %
Avg/Max	0.98	0.59	0.38

Figura 39: 2D Thread state profile

Recollim les diferents figures de les gràfiques dels apartats anteriors de les diferents versions per a poder realitzar la comparació d'aquestes més còmodament.

Initial version	Improving ϕ	Reducing parallelisation overheads
 <p>Figura 27</p>	 <p>Figura 32</p>	 <p>Figura 36</p>
 <p>Figura 28</p>	 <p>Figura 33</p>	 <p>Figura 37</p>

La versió inicial no té marge de millora. El seu temps d'execució queda pla a partir dels 4 threads i lleugerament empitjora a partir dels 12. És per aquest motiu que veiem com l'speed-up queda estancat als 4 threads i baixa als 12.

La segona versió té una considerable millora respecte a la primera versió. Aconseguint un millor temps però el comportament és molt similar (per no dir el mateix) que el de la primera execució: a l'arribar a 4 threads s'estanca i als 12 empitjora; de la mateixa manera passa amb l'speed-up. Però sembla ser més eficient per la millora de temps.



Finalment, arribem a la darrera versió on sí que notem un comportament diferent dels anteriors. A les figures 36 i 37 veiem com, tot i baixar el ritme de millora, se segueix reduint el temps d'execució i l'speed-up millora. L'speed-up comença molt aprop de la línia ideal. A partir dels 4 threads es desvia una mica pel que ja hem comentat anteriorment: anem acumulant overhead tot i que al principi aconseguim despistar-lo per la major granularitat de les tasques però després ja és impossible.

4. Conclusions

Aquest primer laboratori ha servit per a fer una primera aproximació i contacte amb les eines que emprarem durant el que resta de curs a les sessions de laboratori.

Hem començat de pràcticament no saber establir connexió amb els nodes del boada a entendre i aplicar algunes estratègies per reduir el temps d'execució dels nostres programes. Per establir les estratègies utilitzarem recursos com el Paraver que són de molta utilitat a l'hora d'analitzar codi i entendre les dependències d'aquest.

D'altra banda, hem vist que no per tenir més processadors implica que el nostre programa tingui menor temps d'execució; ni que aquest sigui proporcional al nombre de processadors. Tampoc granularitat més fina de tasques implica major paral·lelització i menor temps d'execució. A vegades val la pena estalviar temps d'overhead amb major granularitat.