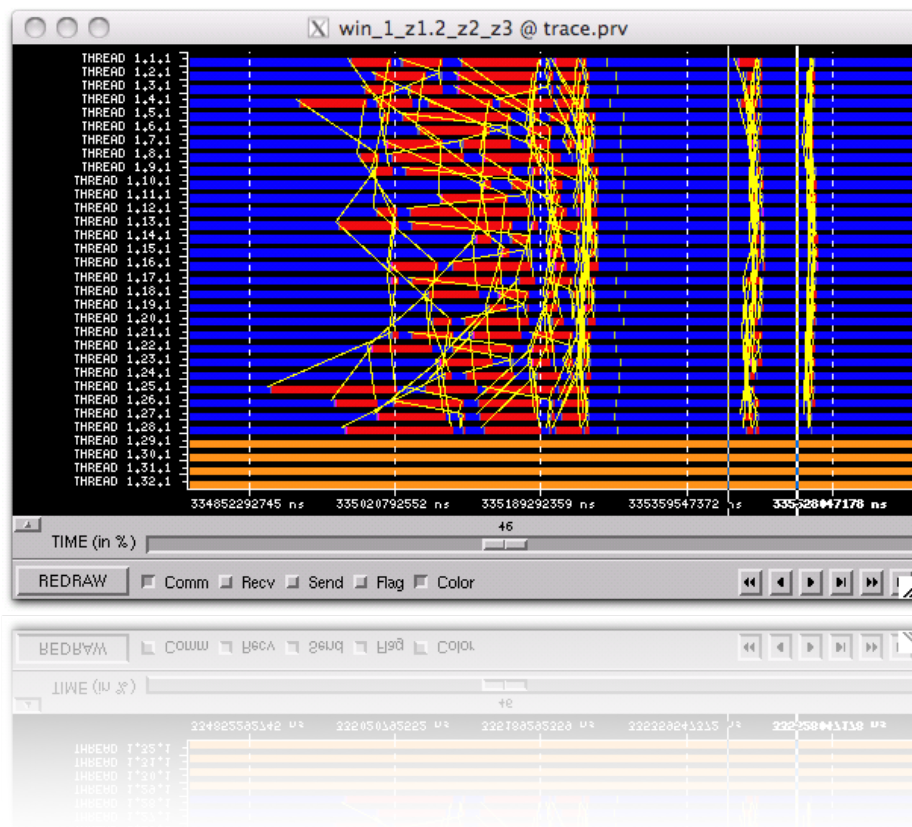


LABORATORI 0

First deliverable



GRUP 12 - 06

Daniel Gil
Carles Viñeta

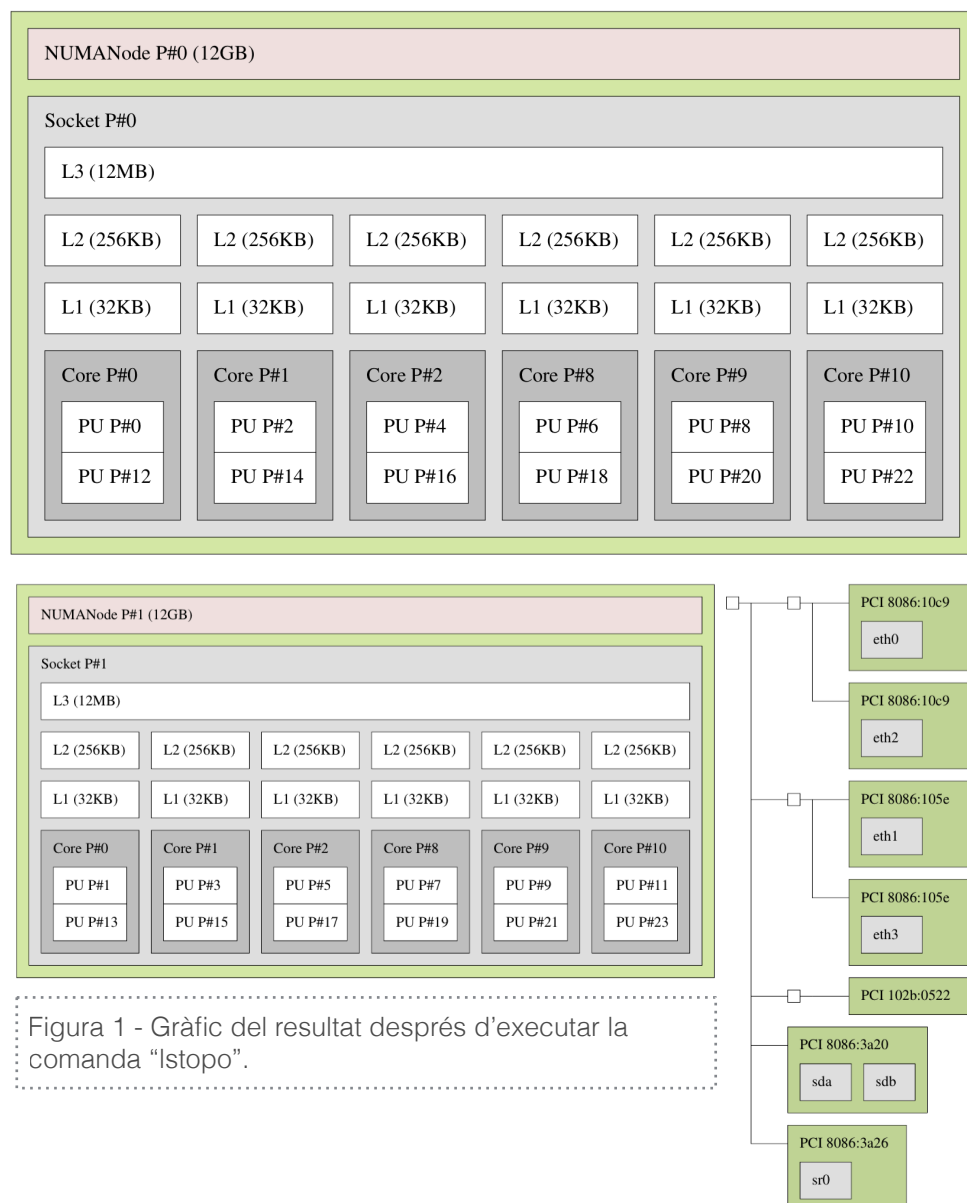
1.- Draw and briefly describe the architecture of the computer in which you are doing this lab session (number of sockets, cores per socket, threads per core, cache hierarchy size and sharing, and amount of main memory).

Comandes que hem executat

```
# lscpu
# lstopo --of fig map.fig
# xfig map.fig
```

Socket(s): 2
Core(s) per socket: 6
Thread(s) per core: 2
Cache: 12MB (L3 - shared) + 256KB (L2 - private) + 32KB (L1 - private)
MemTotal: 24628820 kB (24GB) (12GB/NUMANode)
CPU(s): 24
On-line CPU(s) list: 0-23

Machine (24GB)



2.- Describe what do you need to add to your program to measure the elapsed execution time between a pair of points in the program, clearly indicating the library header file that needs to be included, the library functions that need to be invoked, the data structure and its fields.

La biblioteca en la qual es declara és `<sys/time.h>`.

Els camps són:

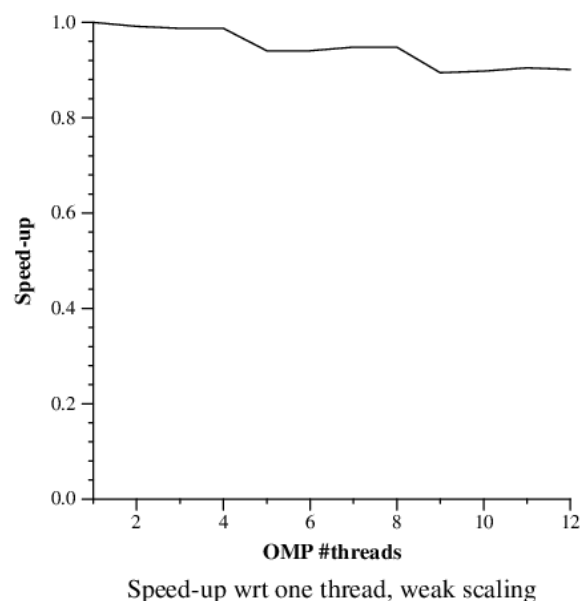
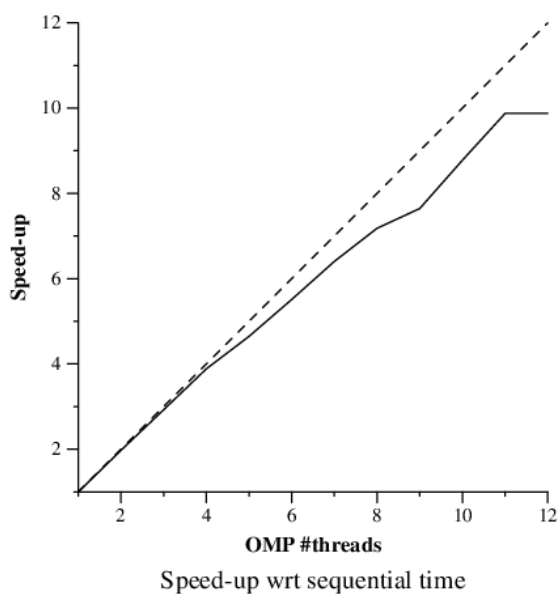
- **long int tv_sec** (representa el nombre de segons sencers de temps transcorregut).
- **long int tv_usec** (Aquest és la resta del temps transcorregut (una fracció d'un segon), representada com el nombre de microsegons. Sempre és de menys d'un milió).

3.- Plot the speed-up obtained when varying the number of threads (strong scalability) and problem size (weak scalability) for pi omp.c. Reason about how the scalability of the program.

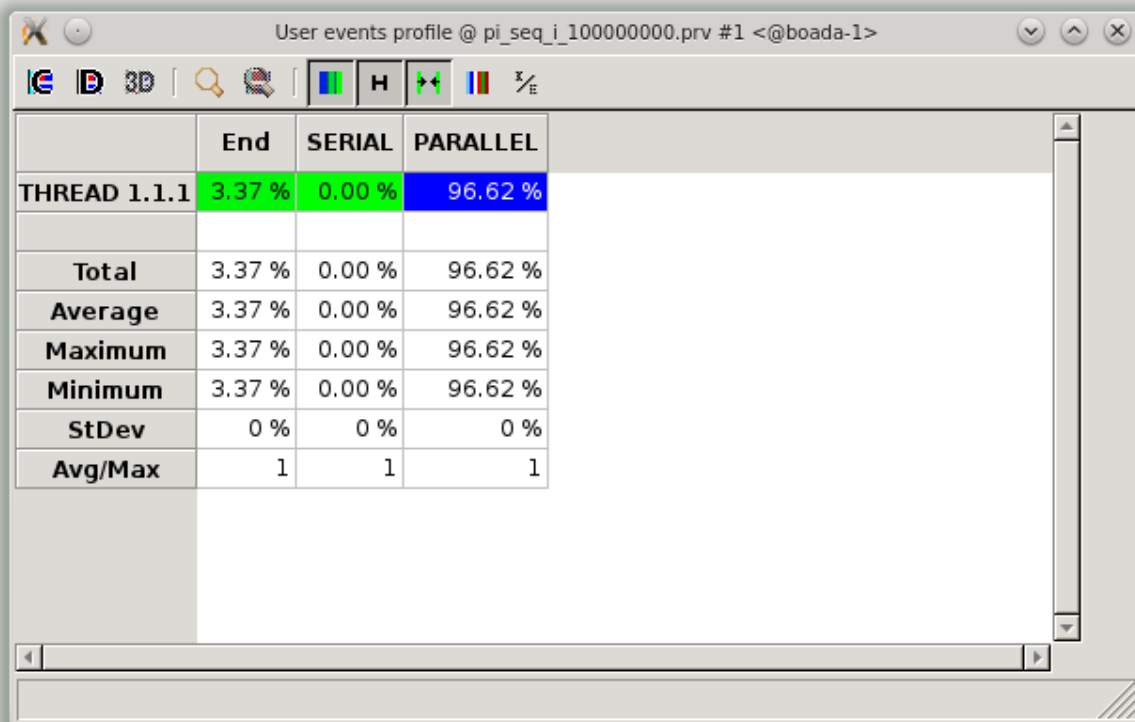
Comandes executades per aconseguir els grafs del speed-up

```
qsub -l execution submit-strong-omp.sh
qsub -l execution submit-weak-omp.sh
```

```
gs pi_omp-1000000000-1-12-3-strong-omp.ps
gs pi_omp-1000000000-1-12-3-weak-omp.ps
```



4.- From the instrumented version of pi seq.c, and using the appropriate Paraver configuration file, obtain the value of the parallel fraction ϕ for this program when executed with 100.000.000 iterations.

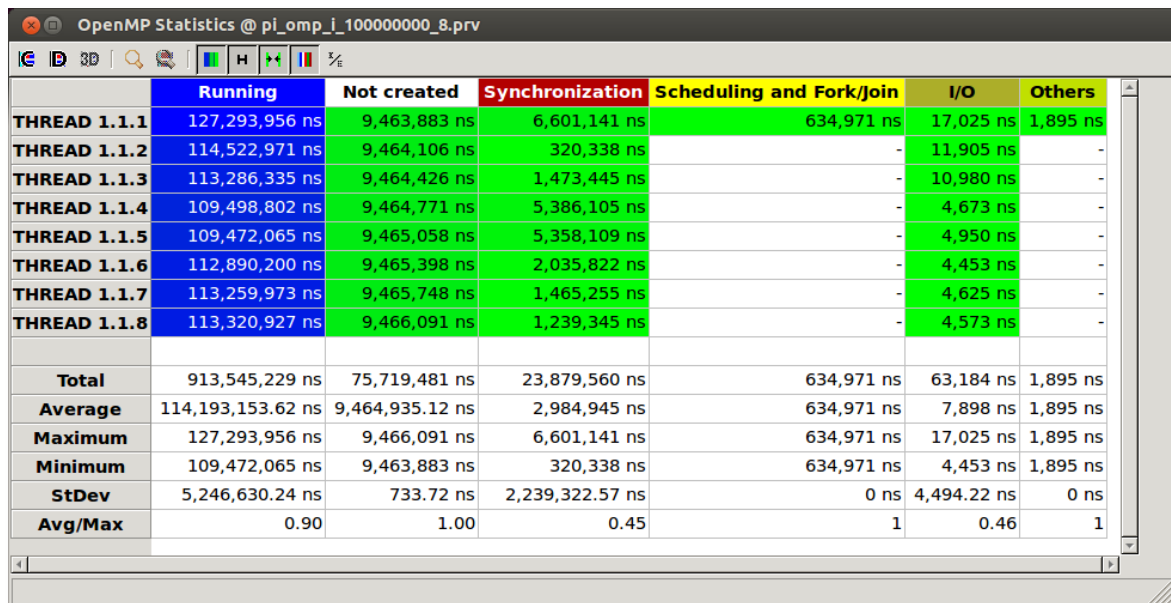


The screenshot shows a window titled "User events profile @ pi_seq_i_100000000.prv #1 <@boada-1>". It contains a table with the following data:

	End	SERIAL	PARALLEL
THREAD 1.1.1	3.37 %	0.00 %	96.62 %
Total	3.37 %	0.00 %	96.62 %
Average	3.37 %	0.00 %	96.62 %
Maximum	3.37 %	0.00 %	96.62 %
Minimum	3.37 %	0.00 %	96.62 %
StDev	0 %	0 %	0 %
Avg/Max	1	1	1

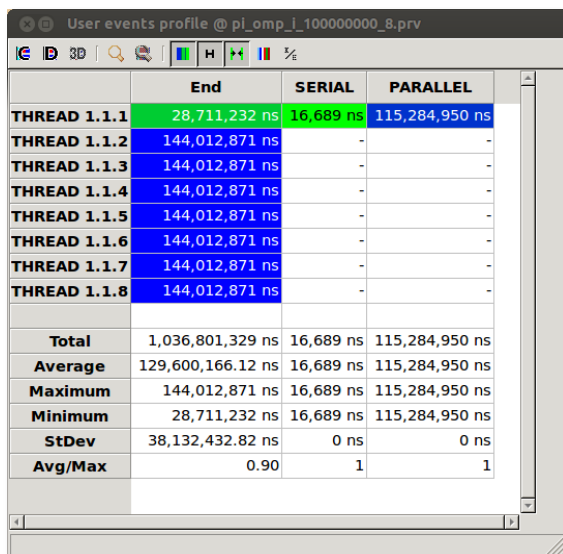
Figura 2 - Amb la configuració user events profile, es pot veure que el percentatge del temps d'execució paral·lelitzable és d'un 96,62%

5.- From the instrumented version of pi omp.c, and using the appropriate Paraver configuration file, show a profile of the % of time spent in the different OpenMP states ONLY during the execution of the parallel region (not considering the sequential part before and after) when using 8 threads and for 100.000.000 iterations.



	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	127,293,956 ns	9,463,883 ns	6,601,141 ns	634,971 ns	17,025 ns	1,895 ns
THREAD 1.1.2	114,522,971 ns	9,464,106 ns	320,338 ns	-	11,905 ns	-
THREAD 1.1.3	113,286,335 ns	9,464,426 ns	1,473,445 ns	-	10,980 ns	-
THREAD 1.1.4	109,498,802 ns	9,464,771 ns	5,386,105 ns	-	4,673 ns	-
THREAD 1.1.5	109,472,065 ns	9,465,058 ns	5,358,109 ns	-	4,950 ns	-
THREAD 1.1.6	112,890,200 ns	9,465,398 ns	2,035,822 ns	-	4,453 ns	-
THREAD 1.1.7	113,259,973 ns	9,465,748 ns	1,465,255 ns	-	4,625 ns	-
THREAD 1.1.8	113,320,927 ns	9,466,091 ns	1,239,345 ns	-	4,573 ns	-
Total	913,545,229 ns	75,719,481 ns	23,879,560 ns	634,971 ns	63,184 ns	1,895 ns
Average	114,193,153.62 ns	9,464,935.12 ns	2,984,945 ns	634,971 ns	7,898 ns	1,895 ns
Maximum	127,293,956 ns	9,466,091 ns	6,601,141 ns	634,971 ns	17,025 ns	1,895 ns
Minimum	109,472,065 ns	9,463,883 ns	320,338 ns	634,971 ns	4,453 ns	1,895 ns
StDev	5,246,630.24 ns	733.72 ns	2,239,322.57 ns	0 ns	4,494.22 ns	0 ns
Avg/Max	0.90	1.00	0.45	1	0.46	1

Figura 3 - Configuració OpenMP Profile (en nanosegons)



	End	SERIAL	PARALLEL
THREAD 1.1.1	28,711,232 ns	16,689 ns	115,284,950 ns
THREAD 1.1.2	144,012,871 ns	-	-
THREAD 1.1.3	144,012,871 ns	-	-
THREAD 1.1.4	144,012,871 ns	-	-
THREAD 1.1.5	144,012,871 ns	-	-
THREAD 1.1.6	144,012,871 ns	-	-
THREAD 1.1.7	144,012,871 ns	-	-
THREAD 1.1.8	144,012,871 ns	-	-
Total	1,036,801,329 ns	16,689 ns	115,284,950 ns
Average	129,600,166.12 ns	16,689 ns	115,284,950 ns
Maximum	144,012,871 ns	16,689 ns	115,284,950 ns
Minimum	28,711,232 ns	16,689 ns	115,284,950 ns
StDev	38,132,432.82 ns	0 ns	0 ns
Avg/Max	0.90	1	1

Figura 4 - Configuració User Events Profile (en nanosegons)

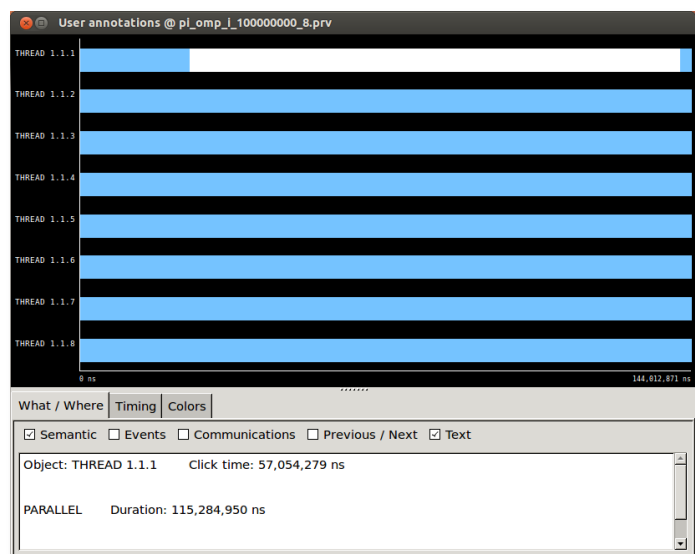


Figura 5 - Configuració User Events (en nanosegons)

Per agafar la part paral·lela **Running**, hem agafat el valor total de la figura 3 (127,293,956 ns). Aquest temps, inclou també la part no paral·lela i per tant li haurem de restar la part seqüencial (fet a la següent pàgina). Per la part de **Synchronization** el valor 6,601,141 ns i, finalment per la part de **Scheduling and Fork/Join** el valor 634,971 ns.

A continuació mostrem els càlculs realitzats.



Figura 6 - Execució de la traça (pi_omp.c)

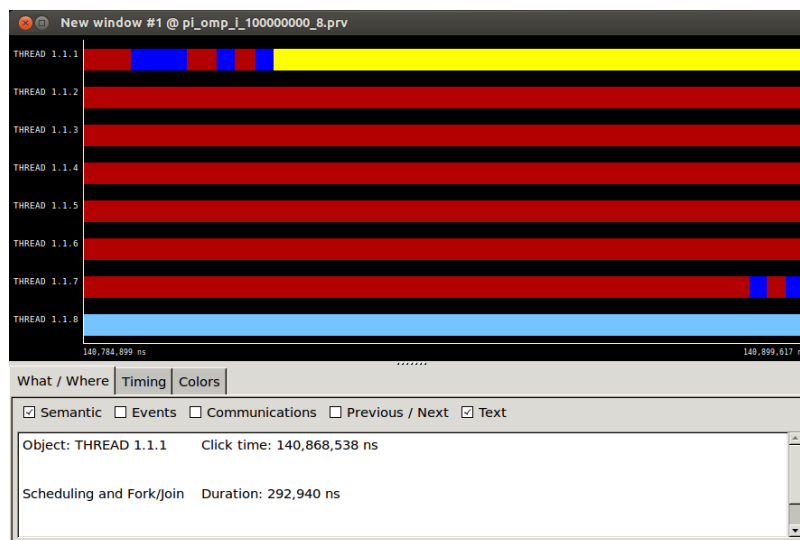


Figura 7 - Imatge ampliada del temps d'execució de la traça (pi_omp.c)

Temps total paral·lel = 115,284,950ns

RUNNING

$127,293,956\text{ns} - 16,284,995\text{ns} - 2,816,215\text{ns} = 108,192,746\text{ns}$

$108,192,746\text{ns} / 115,284,950\text{ns} = 0.93848 * 100 = \mathbf{93.85\%}$

SYNCHRONIZATION

$6,601,141\text{ns} / 115,284,950\text{ns} = 0.05725 * 100 = \mathbf{5.73\%}$

SCHEDULING AND FORK/JOIN

$634.971\text{ns} / 115,284,950\text{ns} = 0.00551 * 100 = \mathbf{0.55\%}$

7.- Include the source code for function dot product in which you show the Tareador instrumentation that has been added to study the potential parallelism in the code. This instrumentation has to appropriately define tasks and filter the analysis of variable(s) that cause the dependence(s).

Simuleu desactivar la dependència de la variable acc amb el tareador de la següent manera i crear una tasca per cada execució del bucle.

```
void dot_product (long N, double A[N], double B[N], double *acc){
    double prod;

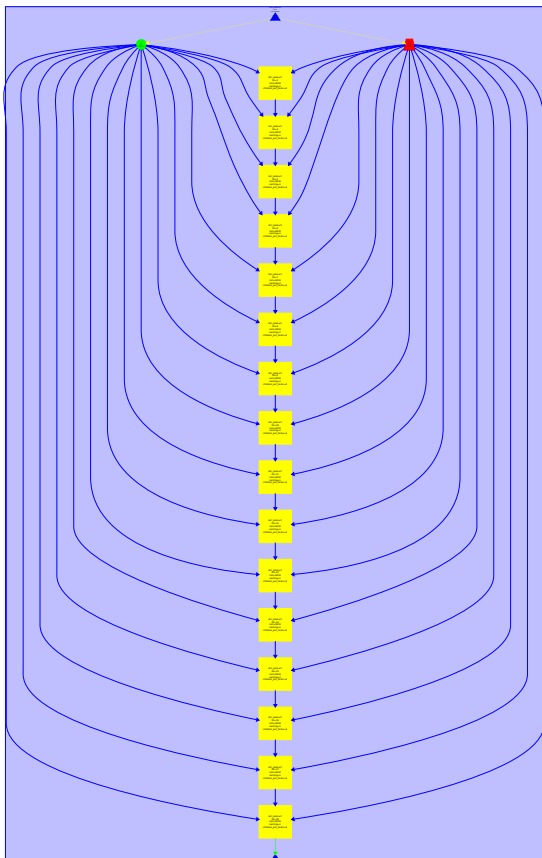
    *acc=0.0;
    int i;
    for (i=0; i<N; i++) {
        tareador_start_task("dot_product");
        prod = my_func(A[i], B[i]);

        tareador_disable_object(acc);
        *acc += prod;
        tareador_enable_object(acc);

        tareador_end_task("dot_product");
    }
}
```

Figura 8 - Codi de dot_product.c modificat (desactivades les variables dependents) i executant una tasca per execució.

8.- Capture the task dependence graph and execution timeline (for 8 processors) for that task decomposition.



Abans de fer les modificacions pertinents per eliminar les dependències, podem observar gràcies al programa Tareador, que les dependències de cada tasca, depenen de l'anterior. És a dir, apart de les dependències d'A i de B que no les podem anul·lar, podem observar la dependència de la variable acc.

A la següent imatge (Figura 10) podrem observar el graf de dependències una vegada filtrada la variable acc. D'aquesta manera estem simulant que les tasques ja no dependrien d'aquesta variable global.

Figura 9 - Graf de dependències sense filtrar les variables dependents.

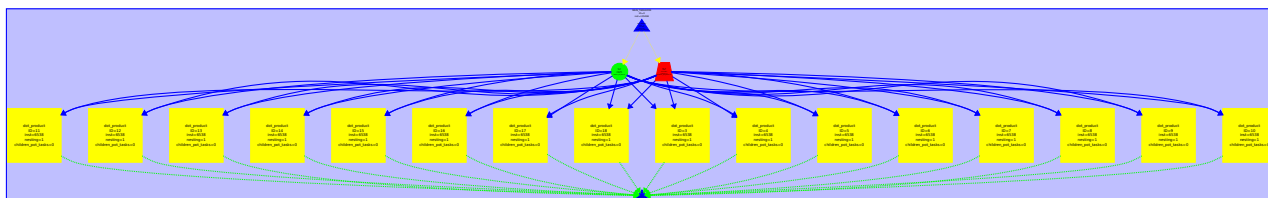


Figura 10 - Graf de dependències una vegada filtrada la variable dependent acc.

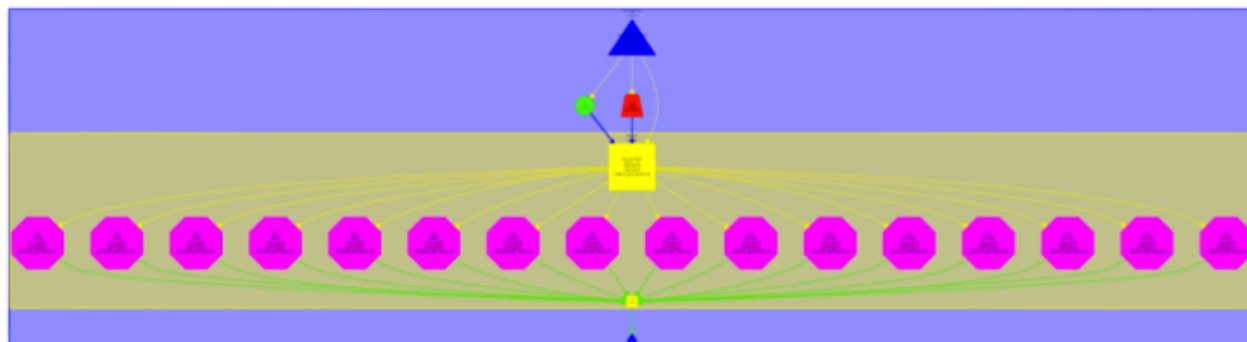


Figura 11 - Graf de dependències una vegada filtrada la variable dependent acc (repetit per aconseguir la següent figura).

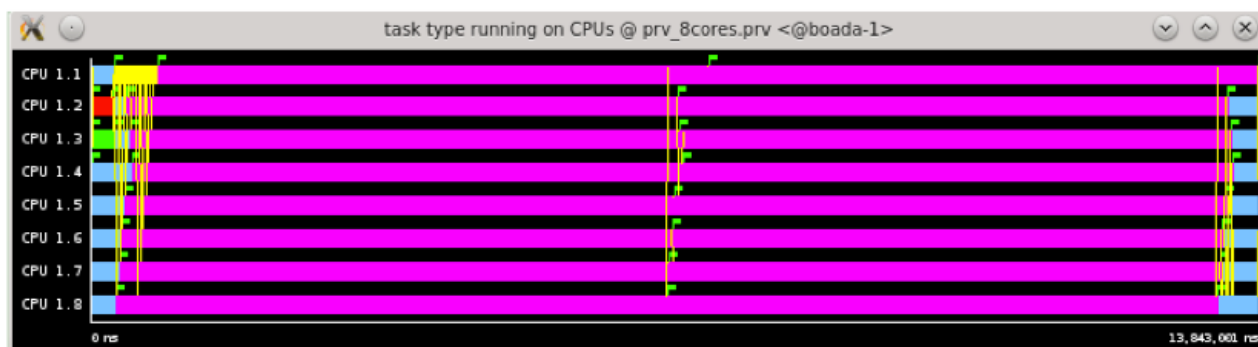
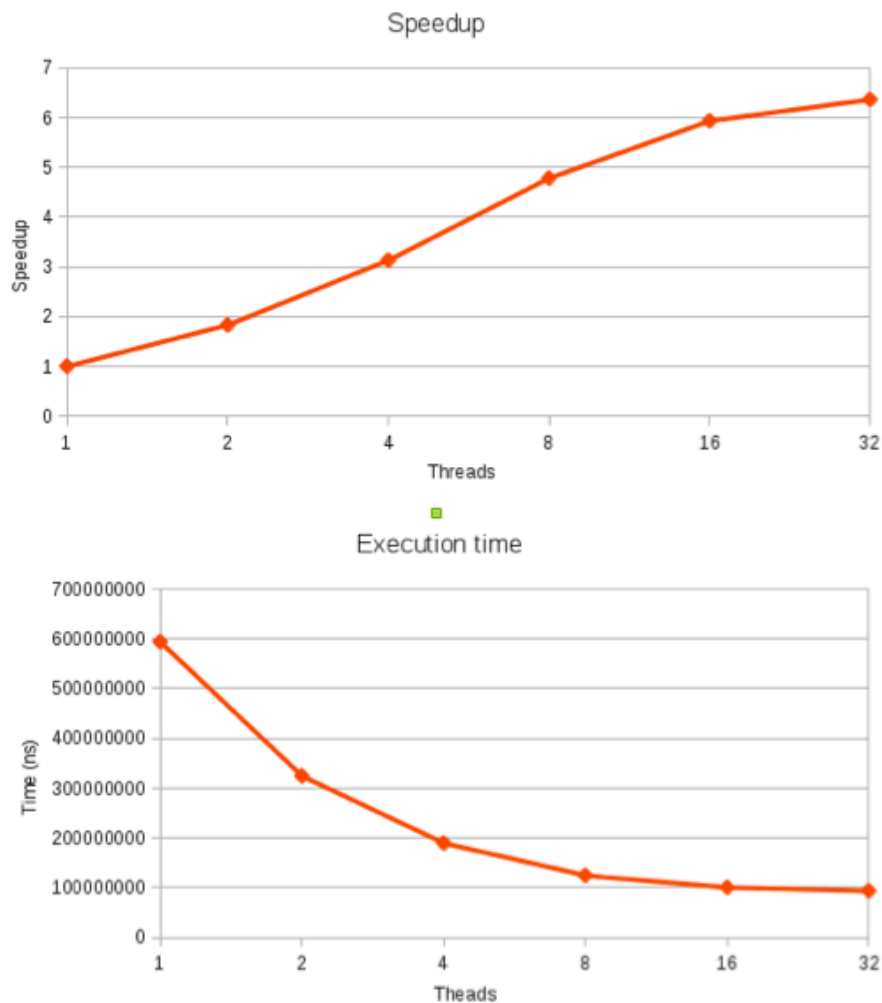


Figura 12 - Timeline de dot_product.c una cop filtrades les variables dependents.

9.- Complete the following table for the initial and different versions generated for 3dfft seq.c.

Version	T_1	T_∞	Parallelism
seq	1.074.011.001 ns	802.112.001 ns	1,34
v1	1.075.098.001 ns	802.334.001 ns	1.34
v2	1.076.922.001 ns	729.824.001 ns	1.47
v3	1.117.452.001 ns	375.207.001 ns	2.97
v4	1.120.698.001 ns	319.570.001 ns	3.51

10.- With the results from the parallel simulation with 2, 4, 8, 16 and 32 processors, draw the execution time and speedup plots for version v4 with respect to the sequential execution (that you can estimate from the simulation of the initial task decomposition that we provided in 3dfft seq.c, using just 1 processor).



Podem observar que la versió seqüencial sempre s'executa en un core independentment dels que tinguem per tant el temps d'execució no varia. En canvi en la versió 4 (paral·lela) a mesura que augmentem el nombre de cores disminueix encara més el temps d'execució.

processors	v4	seq	speed-up
2	579.539.001 ns	806.607.001 ns	1,392
4	405.649.001 ns	802.112.001 ns	1,98
8	345.549.001 ns	802.112.001 ns	2,32
16	319.570.001 ns	802.112.001 ns	2,51
32	319.570.001 ns	802.112.001 ns	2,51

Extres

1.1.1: Quin és el nombre de sockets, cores per socket i threads per core en un cor de la màquina ?

Nombre de sockets: 2

Cores/socket: 6

Threads/core: 2

(24 threads com a màxim)

1.1.2 Quina és la capacitat de memòria d'un node de la màquina i cada NUMA node ?

Memòria total del node: 24,628820 GB

NUMA node: 12 GB

1.1.3: Quins són els nivells de memòria cau L1, L2, L3, privada o compartida per cada core/socket?

L1 cache: 32

L2 cache: 256K

L3 cache: 12288K

1.2. Execució seqüencial i paral·lela del càlcul de Pi amb 1000000000 iteracions i speedup.

SEQÜENCIAL

Number pi after 1000000000 iterations = 3.141592653589971

Total execution time: 7.875957s

CPU → 100%

PARAL·LEL

Number pi after 1000000000 iterations = 3.141592653589769

Total execution time: 1.043445s

CPU → 786%

Speedup

$7.875957s / 1.043445s = 7,55\%$

1.3.1 Després de compilar l'arxiu pi_omp.c, què us diu el compilador ?

El compilador ens mostra 3 warnings sobre la paral·lelització del codi d'aquest arxiu. Tot i això ens ha generat l'executable.

Per solucionar els warnings hem afegit el flag -fopenmp al Makefile i ha compilat correctament.

1.3.2.1 Quina és la comanda del temps sobre el temps de CPU, temps transcorregut (elapsed time) i el % de CPU usada.

La comanda és run-omp.sh, nom: pi_omp, nombre d'iteracions i nombre de threads utilitzats.

1.3.2.2. Indica la capçalera de la llibreria on està declarada l'estructura timeval i quins són els seus camps.

La estructura timeval està declarada a sys/time.h i consta de 2 camps ambdós long int. El primer es diu tv_sec i representa el nombre en segons del temps transcorregut. L'altre camp es diu tv_usec i representa la resta de temps en ms (que no arriba als segons).