**FIRST DELIVERABLE**

**PAR-FIB**

**Membres:**

*Víctor Pérez Martos*

*Albert Suàrez Molgó*

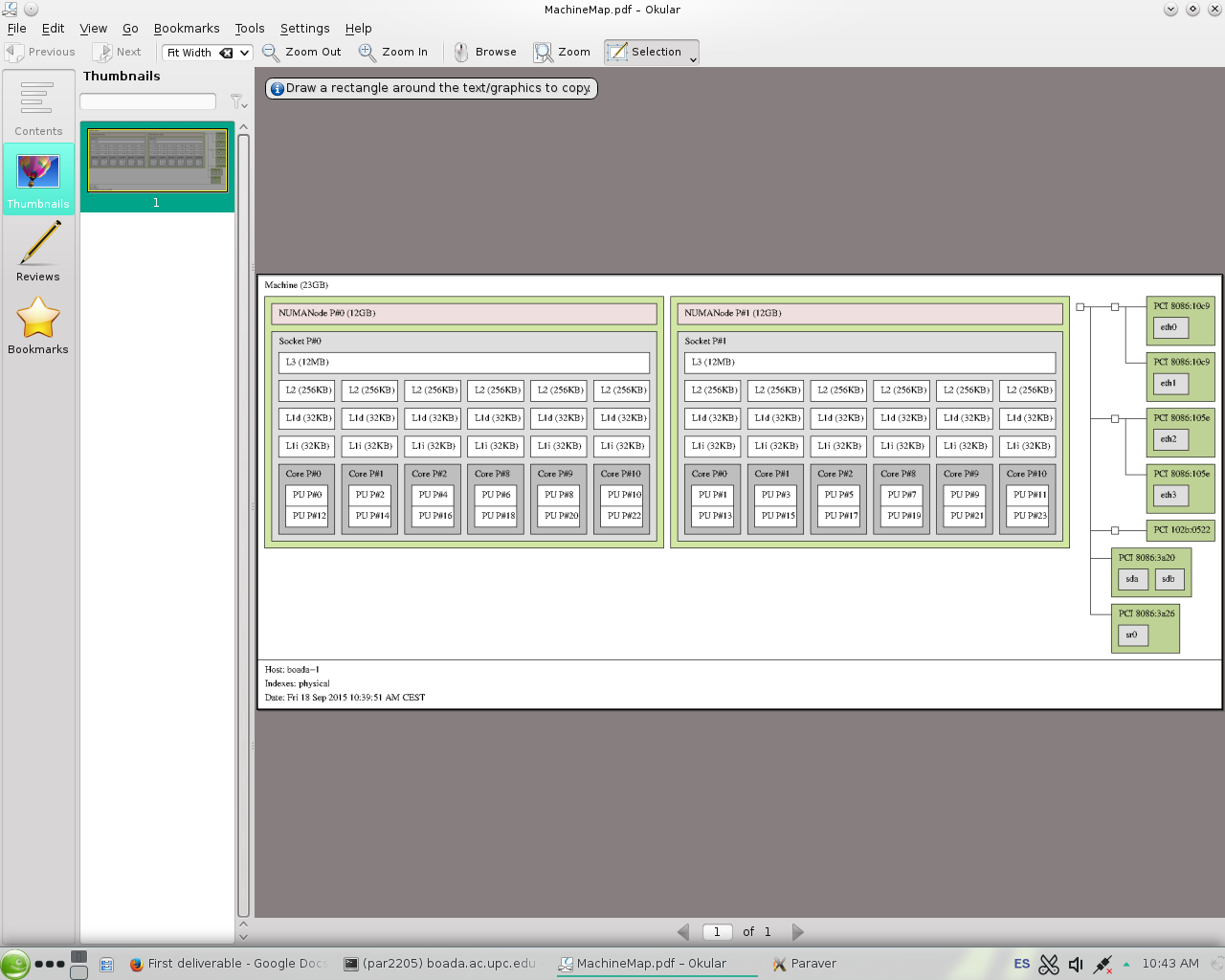
**Grup:** *par2205*

**Data:** *09/10/2015*

**Curs acadèmic:** *QT-2015/16*

Node architecture and memory

**1.**

*Imatge 1*

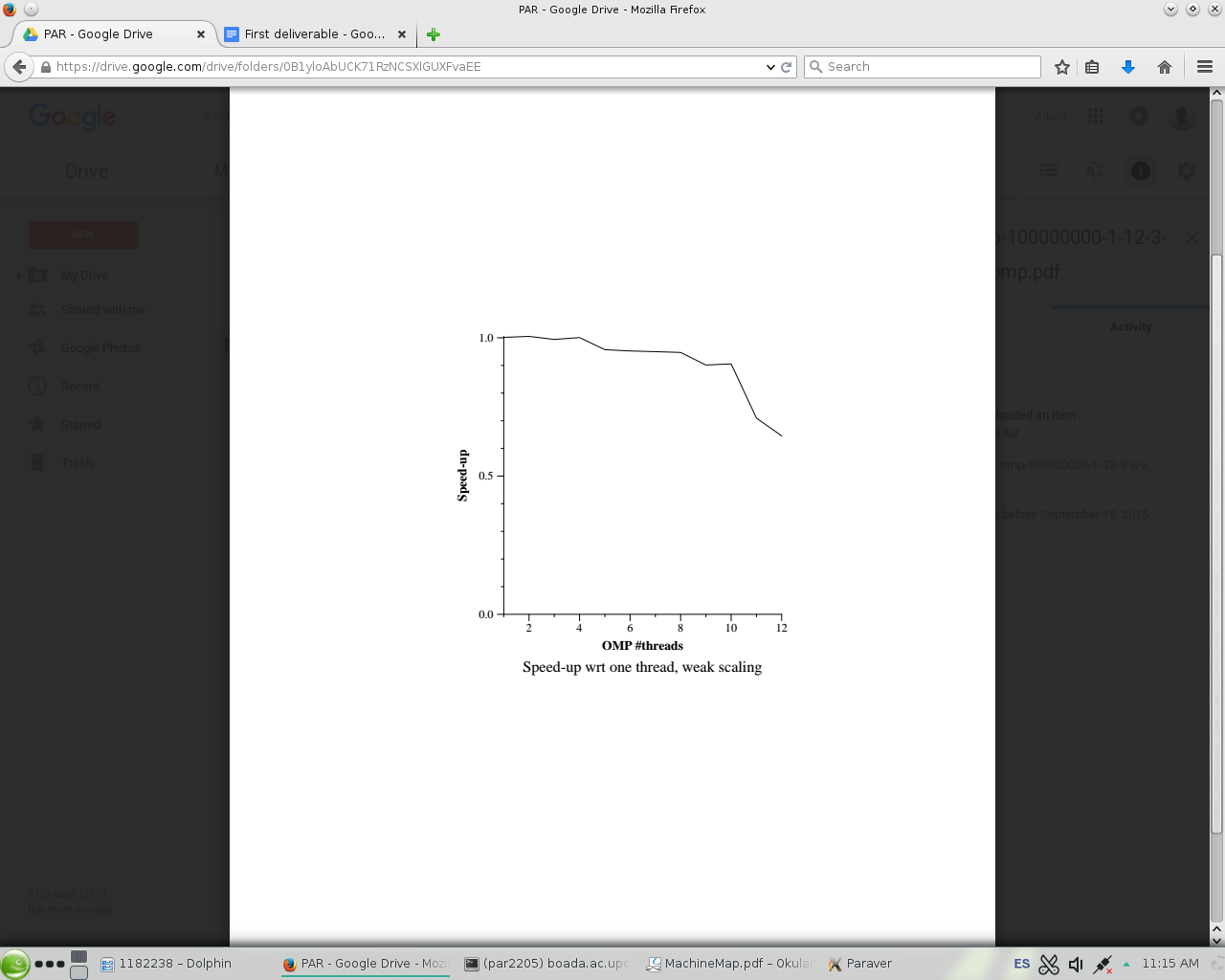
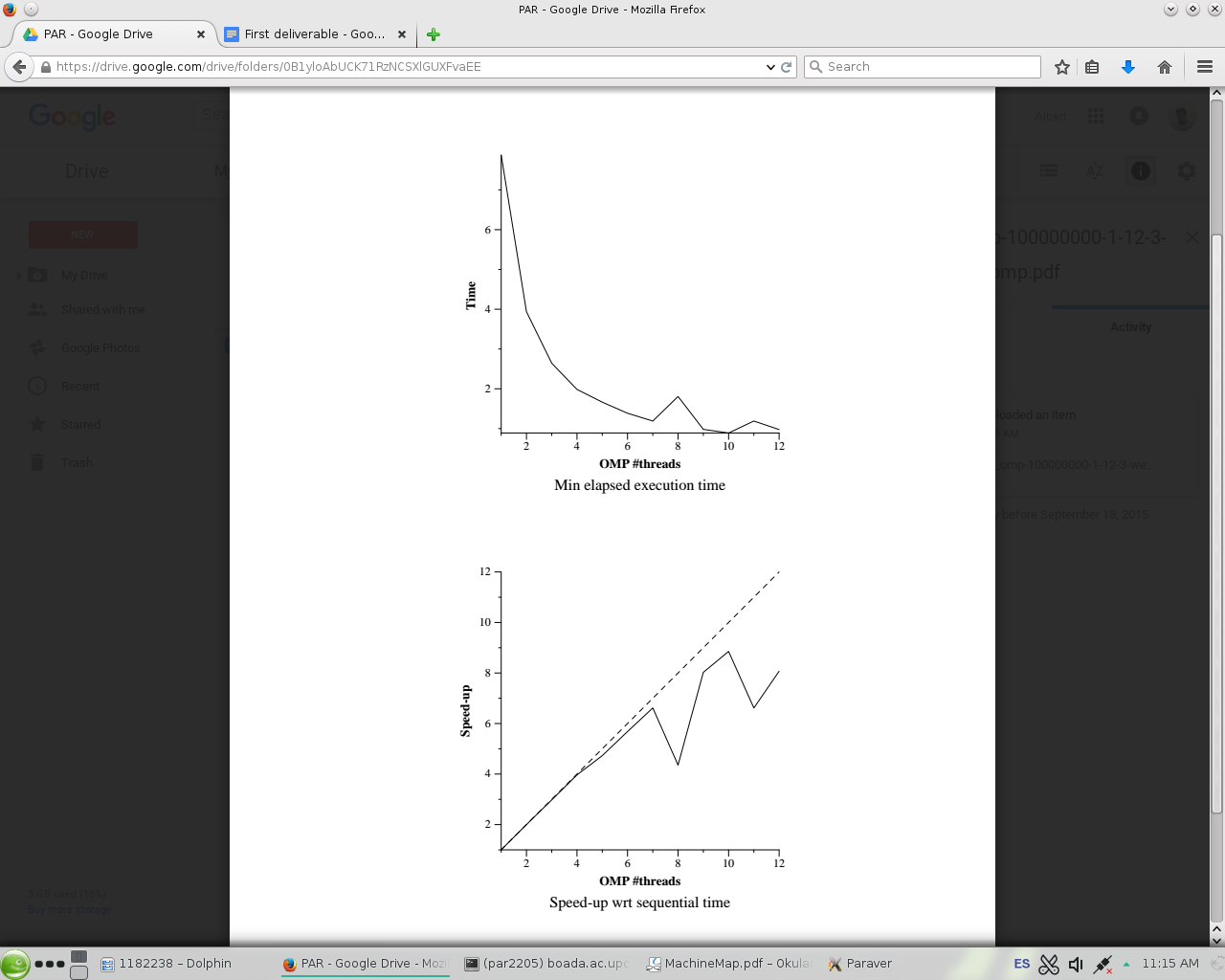
L’ordinador, tal i com mostra la imatge 1, té una memòria de 23GB distribuida en 2 nodes de 12GB aproximadament. La màquina està repartida en 2 sockets, on en cada socket conté 6 cores, i a cada un d’ells trobem 2 threads.

Quant a la distribució de la memòria cache, veiem que cada core disposa de una L1 d’instruccions, una L1 de dades i una L2. Addicionalment, existeix una memòria compartida L3 entre els cores d’un socket.

Timing sequential and parallel executions

**2.**

Per mesurar el temps d’execució del programa entre dos punts s’utilitza la llibreria *sys/time.h*. D’aquesta llibreria s’utilitzen les funcions *gettimeofday()* i indiquem l’interval a mesurar mitjançant la comanda *START\_COUNT\_TIME* per indicar l’inici i *STOP\_COUNT\_TIME* per marcar el final.

**3.**

*Gràfic 1* *Gràfic 2*

En el gràfic 1 es mostra la variació del speedup del programa respecte el nombre de threads que actuen, *weak scalability*. I tal com es veu, marca una tendencia gairebé rectilinia a mesura que augmenten els threads, fins arribar al punt de 10 threads on baixa l’speedup considerablement.

D’altra banda, en el gràfic 2 es pot observar el comportament del programa amb *strong scalability*. A mesura que augmenten els threads, hauria d’augmentar proporcionalment l’speedup. Però en Gràfic 2 es veuen perturbacions degut al multiaccès a la màquina *boada*.

Tracing sequential and parallel executions

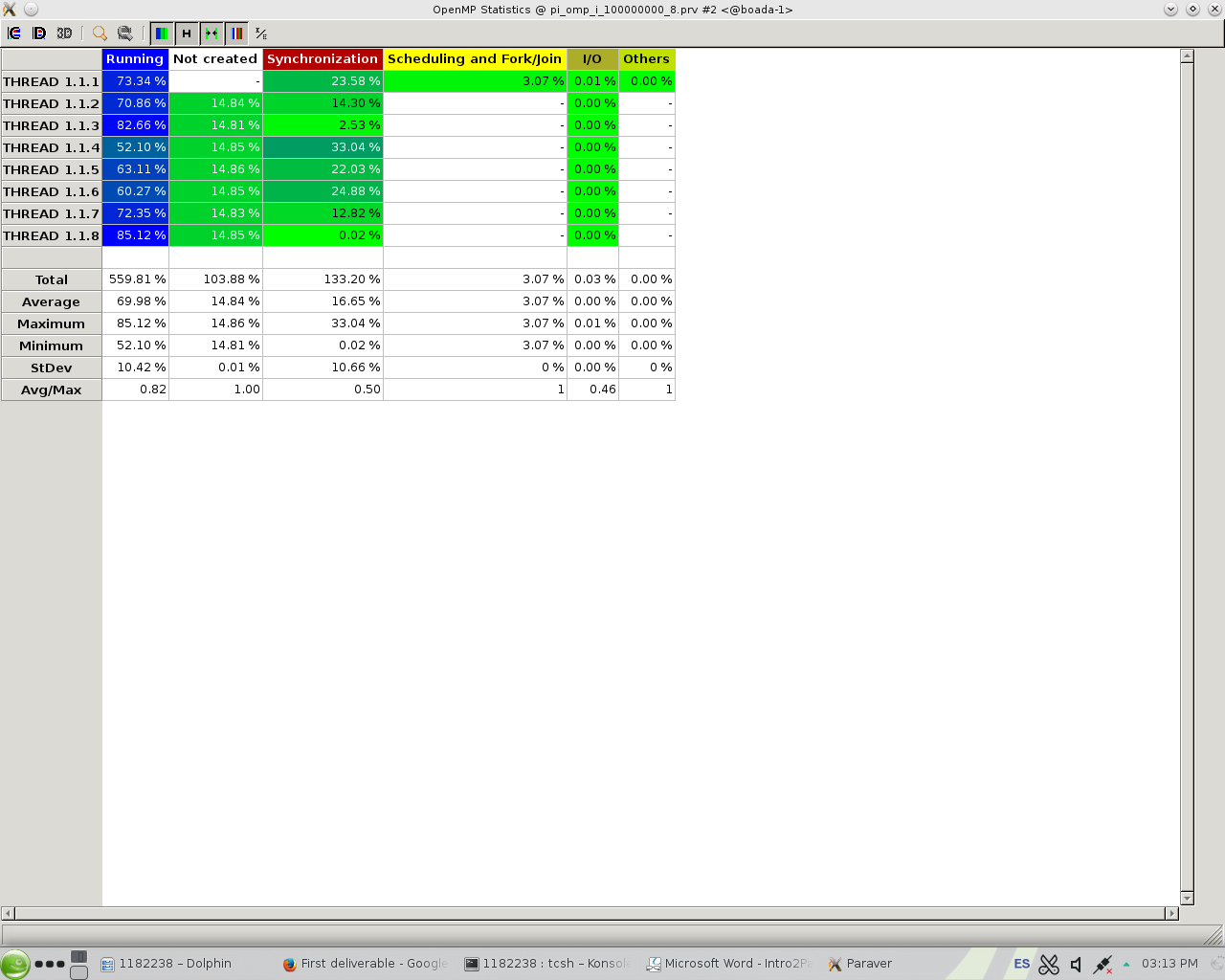
**4.**



*Imatge 2*

Donat que el programa és una versió seqüèncial només està executat per un sol thread. Així doncs, no consta de secció paral·lela (*Φ*).

**5.**



*Imatge 3*

Donada la funció de Paraver, obtenim el resultats mostrats a la imatge 3.

Visualizing the task graph and data dependences

**7.**

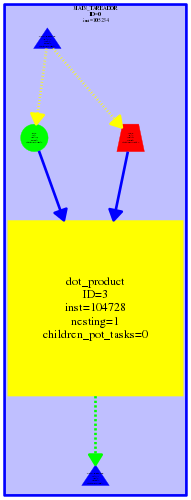
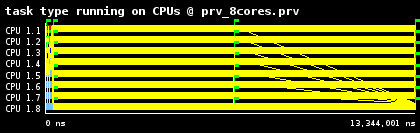
Per tal d’iniciar el tareador cal fer una crida a la funció *tareador\_ON ()* que s’acabarà amb la funció *tareador\_OFF()*.

Per indicar quin codi volem estudiar, cal marcar-ho mitjançant les crides següents:

*tareador\_start\_task("codi\_a\_estudiar");*

*tareador\_end\_task("codi\_a\_estudiar");*

**8.**

*Imatge 4*

En la imatge 5 veiem el graf de dependències de la funció *dot\_product* on es reflexa dues tasques petites que s’executen paral·lelament i una molt més gran que s’inicía quan les dues anteriors acaben.

En la imatge 4 ens mostra el timeline de l’execució d’aquesta funció mitjançant 8 cores.

*Imatge 5*

Analysis of task decompositions

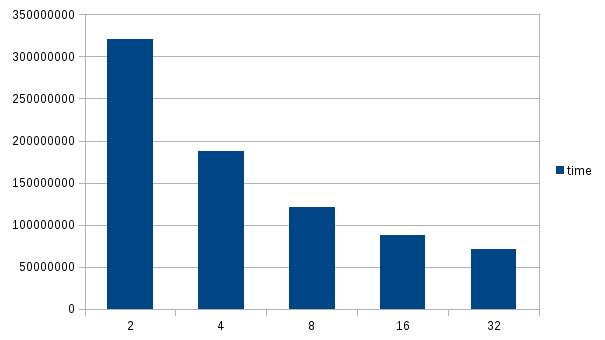
**9.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **T1** | **T∞** | **Parallelism** |
| *seq* | 593.772 ins | 593.758 ins | 1,000024 |
| *v1* | 593.772 ins | 593.758 ins | 1,000024 |
| *v2* | 593.772 ins | 315.188 ins | 1,884 |
| *v3* | 593.772 ins | 108.352 ins | 5,48 |
| *v4* | 593.772 ins | 63.464 ins | 9,35 |

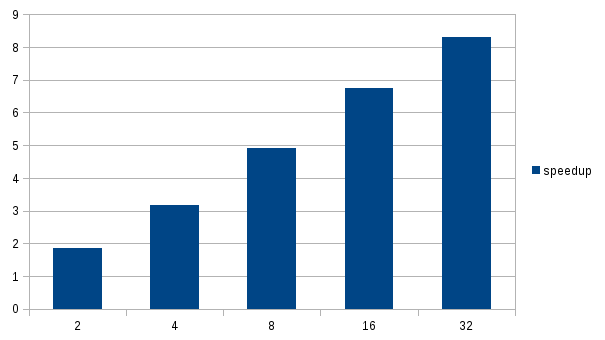
*Taula 1*

A la Taula 1 podem apreciar com una menor granularitat de descomposició a l’hora de paral·lelitzar pot fer que obtinguem un paral·lelisme mes alt.

**10.**



*Gràfic 3*



*Gràfic 4*

En el gràfic 3 veiem com a mesura que augmentem el numero de cores, disminueix el temps d’execució. Simultàniament, en el gràfic 4 ens mostra com a mesura que augmentem el número de cores, augmenta l’speedup respecte la versió seqüèncial.