Part I: OpenMP questionnaire

When answering to the questions in this questionnaire, please DO NOT simply answer with yes, no or

a number; try to minimally justify all your answers. Sometimes you may need to execute several times

in order to see the effect of data races in the parallel execution.

## A) Basics

**1.hello.c**

1. How many times will you see the "Hello world!" message if the program is executed with "./1.hello"?

24, el màxim de threads del processador.

2. Without changing the program, how to make it to print 4 times the "Hello World!” message?

Afegint a la sentencia #pragma omp parallel la condició num\_threads(4).

**2.hello.c: Assuming the OMP NUM THREADS variable is set to 8 with "export OMP NUM THREADS=8"**

1. Is the execution of the program correct? (i.e., prints a sequence of "(Thid) Hello (Thid) world!" being Thid the thread identifier) Which data sharing clause should be added to make it correct?.

No, però ho podem arreglar afegint la condició private(id) a la sentencia #pragma omp parallel.

2. Are the lines always printed in the same order? Could the messages appear intermixed?

No, poden apareixer en diferent ordre perquè s'han executat al mateix temps.

**3.how many.c: Assuming the OMP NUM THREADS variable is set to 8 with "export OMP NUM THREADS=8".**

1. How many "Hello world ..." lines are printed on the screen?

16 linies.

2. If the if(0) clause is commented in the last parallel directive, how many "Hello world ...” lines are printed on the screen?

15 linies fixes, i de 1 a 4 variables en funció del srand().

**4.data sharing.c**

1. Which is the value of variable x after the execution of each parallel region with different data-sharing attribute (shared, private and firstprivate)?

Amb shared, el valor depen a cada procés, però el valor final és 8 perquè tots els threads fan la suma. Amb private, el valor de tots és 1, ja que cada un només fa el +1 a un valor indefinit. En el cas de firstprivate, tots fan una copia de la variable amb el valor que tenia, i al sumar 1, dona 72 per a tots.

2. What needs to be changed/added/removed in the first directive to ensure that the value after the first parallel is always 8?.

#pragma omp atomic abans del x++.

**5.parallel.c**

1. How many messages the program prints? Which iterations is each thread executing?

26 missatges. El thread 0 fa 8 iteracions, el thread 1 en fa 7, el thread 2 en fa 6 i el thread 3, 5.

2. Change the for loop to ensure that its iterations are distributed among all participating threads.

Hem afegit una linea d'openmp abans del bulce for amb: #pragma omp for schedule(static) i hem forçat el bucle a fer 26 iteracions, de tal manera que han quedat distrubuides de forma gairabé equitativa( 6, 6, 7, 7).

**6.datarace.c (execute several times before answering the questions)**

1. Is the program always executing correctly?

No, a vegades no és correcte.

2. Add two alternative directives to make it correct. Explain why they make the execution correct.

Amb #pragma omp single

Amb #pragma omp barrier

**7.barrier.c**

1. Can you predict the sequence of messages in this program? Do threads exit from the barrier in any specific order?

El thread 0 dormirà per 2 segons i llavors haurà d'esperar a que els altres acabin de dormir.

El thread 1 dormirà 5 segons i llavors haurà d'esperar a que els altres acabin de dormir.

El thread 2 dormirà 8 segons i llavors haurà d'esperar a que el thread 3 acabi de dormir.

El thread 3 dormirà 11 segons i llavors no haurà d'esperar a ningú, per el que serà al primer a dir que ja estan tots desperts. Després ho faran els altres.

## B) Worksharing

**1.for.c**

1. How many and which iterations from the loop are executed by each thread? Which kind of schedule is applied by default?

2 iteracions per a cada thread. S'assigna el seu numero d'id multiplicada per 2 i la seguent a cada thread. Per defecte s'aplica schedule static.

2. Which directive should be added so that the first printf is executed only once by the first thread that finds it?.

#pragma omp single abans del printf que distribueix.

**2.schedule.c**

1. Which iterations of the loops are executed by each thread for each schedule kind?

En el primer cas, toquen 4 iteracions a cada thread, totes seguides. En el cas de static,2, s'assignaran chunks de 2 iteracions seguides a cada thread dos vegades, és a dir, primer tots els threads faran 2 iteracions seguides de les 6 primeres i llavors 2 iteracions seguides de les ultimes 6.

En el cas de dynamic,2, els threads agafen chunks de 2 (iteracions seguides) però al acabar en tornen a agafar un altre, sense repartir la carrega entre processadors. En el cas de guided,2, els threads agafen chunks de diferent mida, primer un de 4, llavors un de 3, un altre de 2, un altre de 2 i una de 1.

**3.nowait.c**

1. How does the sequence of printf change if the nowait clause is removed from the first for directive?

Ara s'espera a que s'acabi el bucle 1 per a començar el bucle 2.

2. If the nowait clause is removed in the second for directive, will you observe any difference?

No, ja que ja acaba el procés igualment.

**4.collapse.c**

1. Which iterations of the loop are executed by each thread when the collapse clause is used?

El primer thread fa les 4 primeres, els altres en fan 3 de consecutives.

2. Is the execution correct if the collapse clause is removed? Which clause (different than collapse) should be added to make it correct?.

No, ja que només la i és protegida per la clausula for. Podem fer privada la j amb un private(j).

**5.ordered.c**

1. Can you explain the order in which printf appear?

Si no hi ha la clausula ordered, els missatges apareixen a mesura que els threads els agafen de forma dinamica i per tant despordenats.

En canvi, al utilitzar la clausula ordered, les iteracions apareixen ordenades encara que realment són assignades de forma dinamica.

2. How can you ensure that a thread always executes two consecutive iterations in order during the execution of the ordered part of the loop body?

Modificant la clausula per dynamic,2 o static,2.

**6.doacross.c**

1. In which order are the ”Outside” and ”Inside” messages printed?

Outisde es va fent cada quan el thread pot, però per a Inside, s'ha d'esperar a que hi hagi claculada l'Outside de i -2.

2. In which order are the iterations in the second loop nest executed?

L'iteració es calcula quan les que té a dalt i a l'esquerra ja han estat calculades.

3. What would happen if you remove the invocation of sleep(1). Execute several times to answer in the general case.

Com que ja no ha d'esperar, tots els que ja es poden calcular, els calculca imediatament.

## C) Tasks

**1.serial.c**

1. Is the code printing what you expect? Is it executing in parallel?

Si, fa el calcul de Fibbonacci, però no ho fa en paralel, ja que només ho calcula el thread 0.

**2.parallel.c**

1. Is the code printing what you expect? What is wrong with it?

No. Cada thread agafa el valor n-1 i n-2 d'un privat i per tant no està ben calculat.

2. Which directive should be added to make its execution correct?

#pragma omp single abans del while, ja que així la creació de tasques només la fa un thread.

3. What would happen if the firstprivate clause is removed from the task directive? And if the firstprivate clause is ALSO removed from the parallel directive? Why are they redundant?

En els dos casos funciona, ja que cada thread accedeix a un valor diferent de la llista de resultats.

4. Why the program breaks when variable p is not firstprivate to the task?

El programa segueix funcionant correctament.

5. Why the firstprivate clause was not needed in 1.serial.c?

Perquè no funcionava en paralel, ho feia tot un sol thread.

**3.taskloop.c**

1. Execute the program several times and make sure you are able to explain when each thread in the threads team is actually contributing to the execution of work (tasks) generated in the taskloop.

El thread 2 crea T1 i T2. El thread 3 crea T3 i T4. Finalment thread 3 acaba la creació de TL.

Part II: Parallelization overheads

1. Which is the order of magnitude for the overhead associated with a parallel region (fork and join) in OpenMP? Is it constant? Reason the answer based on the results reported by the pi omp parallel.c code.

És de l'ordre de microsegons. Incrementa el temps d'overhead el número de threads que s'han de crear, però el temps per creació de cada thread disminueix a mesura que hi ha més threads.

2. Which is the order of magnitude for the overhead associated with the creation of a task and its synchronization at taskwait in OpenMP? Is it constant? Reason the answer based on the resultsreported by the pi omp tasks.c code.

La màgnitud és de l'ordre de 0.1 microsegons. En aquest cas, és constant per a qualsevol número de tasques, ja que han d'esperar el mateix tant si s'en crea una més com si s'en creen moltes més.

3. Based on the results reported by the pi omp taskloop.c code, If you have to generate tasks out of a loop, what seems to be better: to use task or taskloop? Try to reason the answer.

Taskloop, tot i que la diferència de temps de sincronització és mínima, però el taskloop ja l'implementa OpenMP, en canvi, amb task s'ha d'afegir el taskwait al final.

4. Which is the order of magnitude for the overhead associated with the execution of critical regions in OpenMP? How is this overhead decomposed? How and why does the overhead associated with critical increase with the number of processors? Identify at least three reasons that justify the observed performance degradation. Base your answers on the execution times reported by the pi omp.c and pi omp critical.c programs and their Paraver execution traces.

L'ordre de magnitud és de segons. L'overhead es divideix en 3 fases, la d'espera, la de lock, i finalment la d'unlock. El incrementar el número de processadors, els threads que hi han en execució han d'esperar més per a poder fer el lock, per el que el temps d'overhead augmenta considerablement. Raons: 1. El fet de compartir una variable que s'actualitza en una regió critica. 2. El fet de que més threads hagin d'accedir a aquesta dada. 3. El fet de que les variables, que són diferents per a cada processador, s'han d'ajuntar.

5. Which is the order of magnitude for the overhead associated with the execution of atomic memory accesses in OpenMP? How and why does the overhead associated with atomic increase with the number of processors? Reason the answers based on the execution times reported by the pi omp.c and pi omp atomic.c programs.

Es tracta d'un ordre de màgnitud de segons, d'uns 30 segons en comparació amb el codi en que es defineix una regió critica al tractarse de 10000000 iteracions.

Al haver-hi més threads, és possible que n'hi hagi més intentant a accedir a la mateixa dada alhora, per el que els temps d'overhead són més grans.

6. In the presence of false sharing (as it happens in pi omp sumvector.c), which is the additional average time for each individual access to memory that you observe? What is causing this in- crease in the memory access time? Reason the answers based on the execution times reported by the pi omp sumvector.c and pi omp padding.c programs. Explain how padding is done in pi omp padding.c.

La causa de que el temps d'accés a memòria augmenti és el fet de que els accessos són a la mateixa linia de cache, y això força als altres processos a actualitzar el bloc, i el temps augmenta considerablament.

El que es fa al utilitzar padding, és fer accessos a posicions de caché que no estan en la mateixa linea, separant les posicions on és guarda el resultat parcial una distància suficient com per a que quedin en una línia diferent.