Second Deliverable

Eric Casanovas

par2110

10/04/2018

OpenMP questionnaire

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 vegades, 1 per cada thread que tenim.

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

OMP_NUM_THREADS=4 ./1.hello, com que fem servir 4 threads només sortiran 4 “Hello World!”.

2.hello.c:

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, per a la correcta execució del codi hauriem d’afegir “private (id)” per a que la variable id no sigui compartida pels threads.

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

No, Sí. Els missatges apareixeran en diferents ordres cada vegada.

3.how_many.c:

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

8 (first parallel) + 2 (second parallel) + 3 (third parallel) + 2 (fourth parallel) + 1 (fifth parallel) = 16 línies.

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

Mateixes línies del 1r al 4t però al 5è tindrem un nombre de línies aleatori entre 1 i 4. -> 16 - 19 línies.

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)?

Shared: indefinit (8 normalment però no sempre), Private: 5, FirstPrivate: 71.

Tant al private com firstprivate com que cada thread té una copia local al final la x no es modificarà. En canvia al shared podria donar-se el cas que 2 threads vagin a

veure el valor de x (per exemple 4) i els 2 threads l'incrementessin 1(x = 5) i el tornarien a guardar, doncs es com si perdessim 1 increment i x valdria 7 al final de l a execució.

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 parallel reduction(+:x). Farà que cada thread sumi un a x i al final se sumin tots cosa que farà que sempre doni 8.

5.parallel.c

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

26 messages, 8 thread1 (del 0 al 7), 7 thread2 (del 1 al 7), 6 thread3 (del 2 al 7), 5 thread4 (del 3 al 7).

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

for (int i=id; i < N; i=i+NUM_THREADS). Així farem que el thread1 executi la iteració 0 i 4, el thread2 la 1 i 5, el thread3 la 2 i 6, i el thread4 la 3 i 7. Doncs haurem executat les 8 iteracions del bucle amb 4 threads fent que cada thread faci 2 iteracions.

6.datarace.c:

1. Is the program always executing correctly?

No, com que la x la comparteixen tots els threads i tots ells executen el mateix(++x) variarà el resultat.

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

- Colocant un "#pragma omp atomic" a l'inici del bucle for i abans del ++x. Així provocaríem que només ho fes un thread.
- Afegir a "#pragma omp parallel private(i)" un "reduction (+:x)" així al final de la execució sumariem tots els valors de x i reornaria el valor correcte

7.barrier.c:

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

Si, perquè sabrem quants segon se'n van a dormir cada thread.

No, ja que el barrier el que fa es que es quedi bloquejats els threads que arribin fins que tots els threads arribin allà.

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?

Cada thread executa 2 iteracions del bucle i els printf es faran en un ordre aleatori.

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

Colocar la línia "#pragma omp single" abans del printf així només ho executarà el 1r thread que arribi.

2.schedule.c:

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

Loop1 (static): es reparteixen les i iteracions (12) del bucle entre el nombre de threads (3) que tenim 12/3 = 4 iteracions per thread

Loop2 (static,2): es reparteixen les iteracions (12) del bucle de 2 en 2 per cada thread que tenim (3). En aquest cas, el thread0 farà la iteració 0, 1, 6 i 7, el thread1 el 2, 3, 8 i 9, i el thread 2 el 4, 5, 10 i 11.

Loop3 (dynamic, 2): cada thread quan arriba agafa 2 iteracions del bucle i així fins acabar el bucle, per això observem que si executem el problema pot donar-se que cada cop un thread faci diferents iteracions i fin i tot que algun thread faci mes iteracions que un altre.

Loop4 (guided, 2): igual que el dynamic, només que cada vegada que un thread agafi iteracions decreixerà el chunk fins a un mínim(el nombre passat després del guided).

3.nowait.c:

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

Amb el nowait fem que no se sincronitzin els threads al final del 1r bucle i pot ser que es comenci el loop2 sense haver acabat el loop1, en canvi si esborrem el nowait, farem que se sincronitzi i farà 1r el loop1 i quan acabi farà el loop2.

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

No tindrem cap canvi perquè no tenim més codi després del 2n loop.

4.collapse.c:

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

Thread0 → 4 iteracions, altres threads→ 3 iteracions

Es volen paralelitzar els 2 bucles i com tenim en total 25 iteracions (5x5) els repartim entre els 8 threads i tindríem 3 iteracions per thread i faltaria 1 iteració que li donaríem al thread0.

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

No, caldria privatitzar les variables i, j per a que cada thread tingues una copia local de i, j.

#pragma omp for private(i, j)

5.ordered.c:

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

No se sap perquè el dynamic implica que cada vegada que tinguem un thread sense fer res li asignara una iteracio del bucle i doncs podria ser que un thread anes més rapid que un altre i això fa que no sapiguem l'ordre.

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?

Canviant el shedule(dynamic) per schedule(dynamic, 2).

6.doacross.c:

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

L'outside s'executa quan pot el thread, i l'inside quan s'ha executat la iteració i-2.

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

S'executa despres de que s'executin les seves dependencies que son la cel·la superior i la de l'esquerra de la matriu.

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

S'executa en ocasions una iteració del bucle posterior anteriorment, per exemple la 4-1 hauria d'executar-se anteriorment a la 2-2 i sense l'sleep podria ser que el thread que fa la 2-2 l'executes més rapid que la del 4-1, doncs l'sleep fa que es facin en l'ordre correcte.

C) Tasks

1.serial.c:

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

Si, imprimeix per pantalla la successió de fibonnaci. No, s'esta executant en seqüencial, només l'està executant el thread0.

2.parallel.c:

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

No es correcta la successió de fibonnaci.

No es correcte degut a que varis threads executen la funció processwork(p).

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

Colocant un "#pragma omp single" abans del "#pragma omp task..." i així farem que només un thread executi el el processwork(p).

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?

No passaria res. No afectarà ja que ja de per si el "#pragma omp task" ja privatitzarà la variable p.

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

Degut a que sinó la variable p serà compartida pels demes threads i això provocarà un error al fer p→next.

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

Perquè es seqüencial i com que no tenim més threads no cal que privatitzem la variable perquè cap altre thread la farà servir perquè no hi han més.

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.

Un dels threads crea T1 i T2, doncs es creen 4 tasques i els threads es posen a fer els fors i el thread que ha creat T1 i T2 fa un sleep de x segons i quan acaba l'sleep "ajuda" als demes threads a acabar el programa asumint part de la feina.

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.

```
par2110@boada-1:~/lab2/overheads$ cat pi_omp_parallel_1_24.txt
All overheads expressed in microseconds
Nthr    Overhead        Overhead per thread
2       1.8634          0.9317
3       1.7101          0.5700
4       2.1385          0.5346
5       2.6946          0.5389
6       2.7768          0.4628
7       2.6700          0.3814
8       2.9549          0.3694
9       3.3682          0.3742
10      3.1799          0.3180
11      3.8737          0.3522
12      3.3237          0.2770
13      3.8935          0.2995
14      4.1724          0.2980
15      3.9468          0.2631
16      4.6981          0.2936
17      4.8853          0.2874
18      4.6618          0.2590
19      4.8999          0.2579
20      4.4526          0.2226
21      5.3195          0.2533
22      5.1437          0.2338
23      5.9062          0.2568
24      5.8251          0.2427
```

L'ordre de magnitud es en microsegons.

Observem que cada vegada que creem un thread esta augmentant el overhead del programa, però també que per cada thread creat cada vegada el overhead de creació és menor fins al punt de que sembla que es constant al voltant dels 0.240 microsegons.

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 results reported by the pi omp tasks.c code.

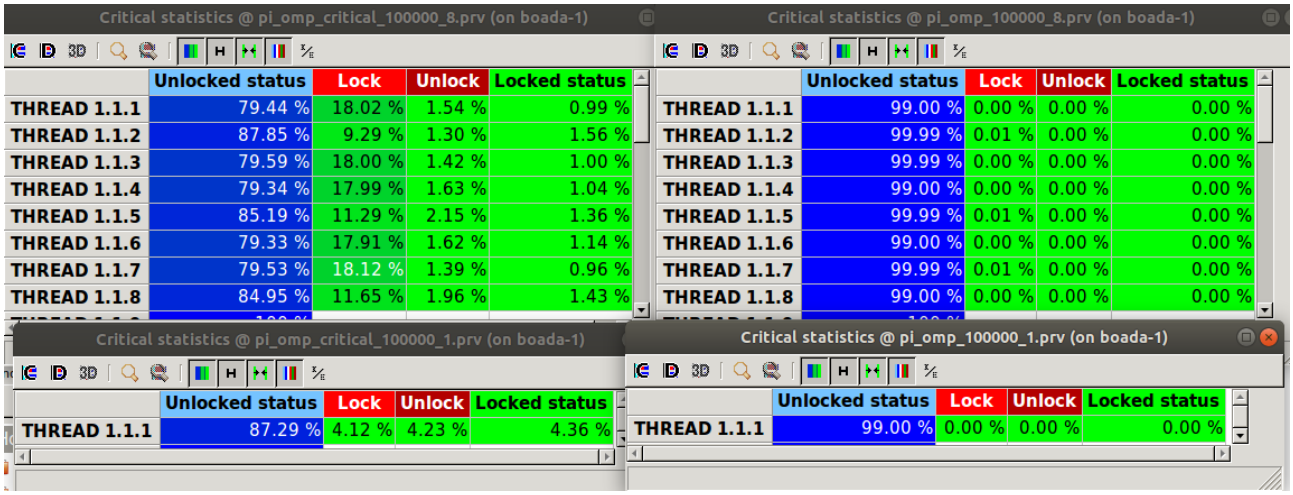
L'overhead de sincronització depen de la quantitat de tasques creades pel programa, per això observem que si tenim molts threads no afecta el overhead. Es constant.

```
All overheads expressed in microseconds
Ntasks  Overhead per task
2       3.9779
4       3.3977
6       3.4184
8       3.4529
10      3.3045
12      3.7799
14      3.9031
16      3.4122
18      3.4601
20      3.4953
22      3.4128
24      3.1620
26      3.1394
28      4.0088
30      3.7428
32      3.5470
34      3.6984
36      2.3021
38      2.2585
40      3.3191
42      3.4611
44      3.2436
46      3.1810
48      3.1301
50      2.9164
52      2.5881
54      3.6167
56      3.6085
58      3.5899
60      3.5075
62      3.1783
64      3.4409
```

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.

Seria millor fer servir task perquè taskloop serveix per especificar que 1 o mes loops s'executaran en paralel usant tasques, i com que hem de crear tasques sense cap loop es millor crear 1 unica tasca amb #pragma omp task.

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.



Temps execució: pi_omp_100000_1 = 207,023,952 μs

pi_omp_100000_8 = 273,319,789 μs

pi_omp_critical_100000_1 = 340,905,117 μs

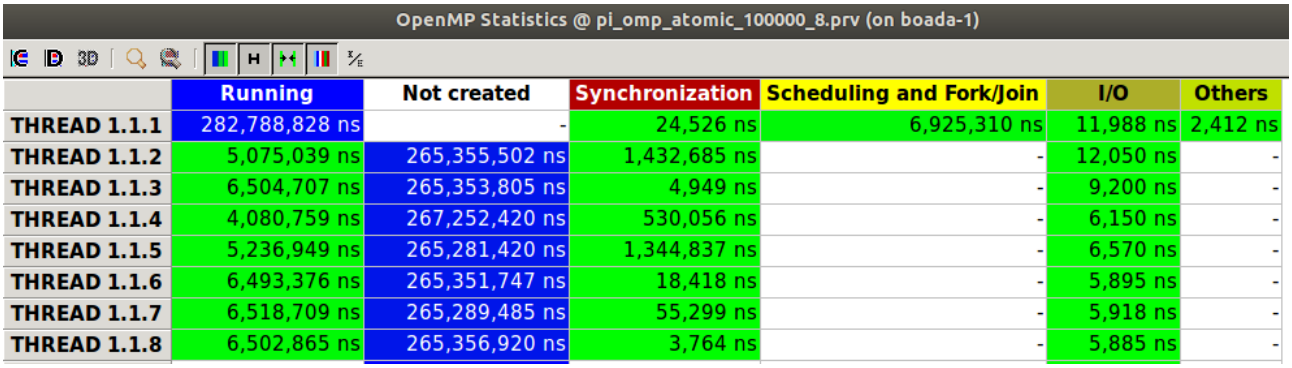
pi_omp_critical_100000_8 = 336,658,554 μs

L'orde de magnitut de del overhead es de microsegons.

L'overhead es descomposa en Unlocked Status, Lock, Unlock i Locket status.

Observem que si augmentem el nombre de threads augmentarà també el overhead que ve donat per la creació de noves tasques, la espera per accedir a les variables (en aquest cas la variable sum) i per la sincronització de les tasques.

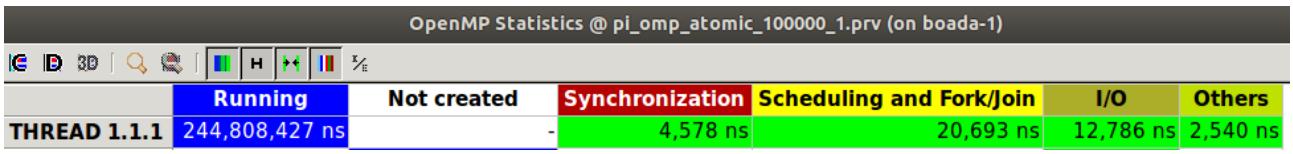
5.- Which is the order of magnitude for the overhead associated with the execution of atomic memory accesses in OpenMP? How and why does the overhead associated with atomic increase with the number of processors? Reason the answers based on the execution times reported by the pi omp.c and pi omp atomic.c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



OpenMP Statistics @ pi_omp_atomic_100000_8.prv (on boada-1)

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	282,788,828 ns	-	24,526 ns	6,925,310 ns	11,988 ns	2,412 ns
THREAD 1.1.2	5,075,039 ns	265,355,502 ns	1,432,685 ns	-	12,050 ns	-
THREAD 1.1.3	6,504,707 ns	265,353,805 ns	4,949 ns	-	9,200 ns	-
THREAD 1.1.4	4,080,759 ns	267,252,420 ns	530,056 ns	-	6,150 ns	-
THREAD 1.1.5	5,236,949 ns	265,281,420 ns	1,344,837 ns	-	6,570 ns	-
THREAD 1.1.6	6,493,376 ns	265,351,747 ns	18,418 ns	-	5,895 ns	-
THREAD 1.1.7	6,518,709 ns	265,289,485 ns	55,299 ns	-	5,918 ns	-
THREAD 1.1.8	6,502,865 ns	265,356,920 ns	3,764 ns	-	5,885 ns	-

pi_omp_atomic 8 threads



OpenMP Statistics @ pi_omp_atomic_100000_1.prv (on boada-1)

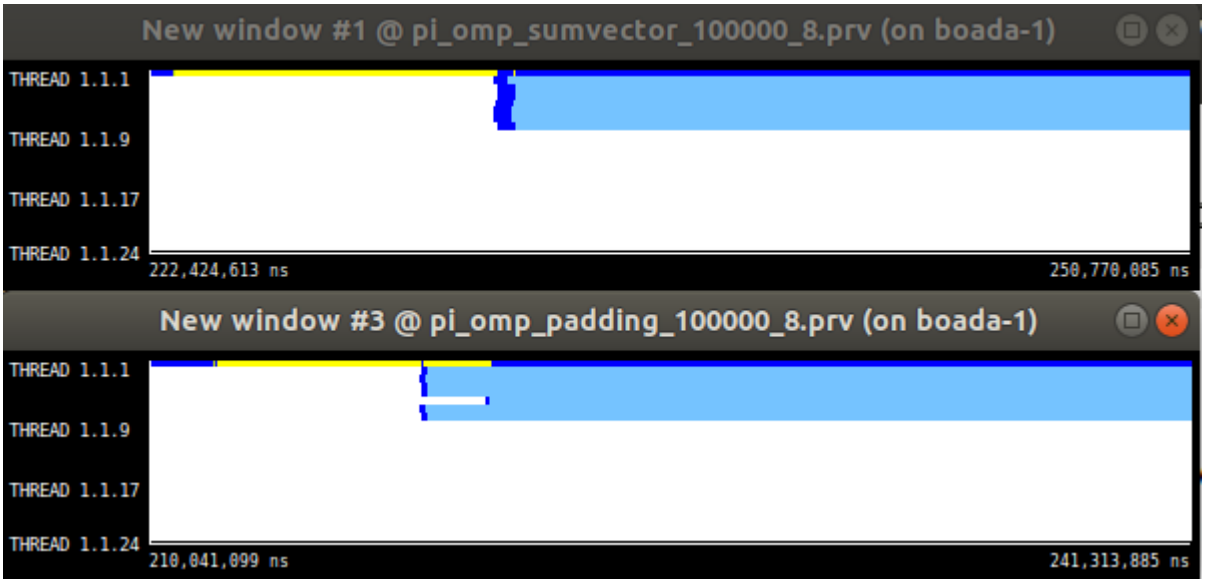
	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	244,808,427 ns	-	4,578 ns	20,693 ns	12,786 ns	2,540 ns

pi_omp_atomic 1 thread

L'ordre de magnitut són milisegons.

Observem que cada vegada que creem un thread està augmentant el overhead del programa degut a que haurà d'esperar-se a que un dels threads faci la línia de codi: "sum += 4.0/(1.0+x*x);", doncs com més threads tinguem, més threads hauran d'esperar-se. També cal destacar que augmentarà degut a la creació de nous threads i la seva sincronització.

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 increase 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



Ambdues finestres ampliades (no està a escala)

A partir de les anteriors traces obtenim el temps d'execució de cada programa.
241.313.885 + 250.770.085 = 492083970 ns
Tot aquest temps es provocat pel false sharing que es un problema que es produeix quan varis threads modifiquen varies posicions de memòria a una mateixa línia de cache, i això va provocant invalidacions de la línia de cache.
Això podríem arreglar-ho forçant que cada element es trobi a una fila diferent de la cache, i això ho podríem aconseguir transformant el vector en una matriu(per exemple).