

# Second Deliverable

Robert Almar Graupera

Erick Aramayo Monrroy

10/04/2015

---par1205---

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\_overhead.c code.

L'ordre de magnitud és en microsegons. L'overhead no es constant ja que quants més threads tenim, més augmenta l'overhead. En canvi, podem veure que el temps d'overhead per thread disminueix ja que molts threads s'executen en paral·lel.

```
All overheads expressed in microseconds
Nthr   Time    Time per thread
2       2.1383  1.0692
3       2.1204  0.7068
4       2.4740  0.6185
5       2.5288  0.5058
6       2.5145  0.4191
7       2.8098  0.4014
8       3.3921  0.4240
9       3.3537  0.3726
10      3.5664  0.3566
11      3.5711  0.3246
12      3.6254  0.3021
13      4.2265  0.3251
14      4.0399  0.2886
15      4.2594  0.2840
16      4.6745  0.2922
17      4.4223  0.2601
18      4.9242  0.2736
19      4.6558  0.2450
20      5.0953  0.2548
21      4.8060  0.2289
22      5.2223  0.2374
23      5.3186  0.2312
24      5.4614  0.2276
Number pi after 1 iterations = 0.0000000000000000
Total execution time: 0.892115s
```

Aquest temps els hem obtingut ficant a la cua d'execució l'script "submit-omp-overhead.sh" que ens retornarà un fitxer .txt del temps d'execució per diversos threads del programa "pi\_omp\_overhead.c"

2. 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 es de milisegons. Les parts que componen l'execució de l'overhead són: lock, locked status, unlock i unlock status.

A les imatges podem veure com augmenta el temps d'overhead proporcionalment al nombre de processadors.

Critical statistics @ pi\_omp\_critical\_i\_100000\_1.prv <@boada-1>

	Unlocked status	Lock	Unlock	Locked status
THREAD 1.1.1	187,225.46 us	137,163.65 us	135,055.44 us	132,662.06 us
Total	187,225.46 us	137,163.65 us	135,055.44 us	132,662.06 us
Average	187,225.46 us	137,163.65 us	135,055.44 us	132,662.06 us
Maximum	187,225.46 us	137,163.65 us	135,055.44 us	132,662.06 us
Minimum	187,225.46 us	137,163.65 us	135,055.44 us	132,662.06 us
StDev	0 us	0 us	0 us	0 us
Avg/Max	1	1	1	1

Critical statistics @ pi\_omp\_critical\_i\_100000\_8.prv #1 <@boada-1>

	Unlocked status	Lock	Unlock	Locked status
THREAD 1.1.1	172,176.29 us	182,383.95 us	21,862.31 us	19,526.35 us
THREAD 1.1.2	74,282.67 us	282,728.77 us	19,435.53 us	19,501.93 us
THREAD 1.1.3	74,322.73 us	282,493.62 us	19,531.71 us	19,600.84 us
THREAD 1.1.4	170,949.10 us	184,880.07 us	20,748.83 us	19,370.89 us
THREAD 1.1.5	174,314.22 us	181,544.18 us	20,635.65 us	19,454.85 us
THREAD 1.1.6	148,722.17 us	204,780.58 us	21,451.53 us	20,994.62 us
THREAD 1.1.7	83,882.24 us	269,190.01 us	21,391.66 us	21,484.98 us
THREAD 1.1.8	83,872.48 us	272,088.55 us	19,938.85 us	20,049.03 us
Total	982,521.90 us	1,860,089.72 us	164,996.06 us	159,983.49 us
Average	122,815.24 us	232,511.22 us	20,624.51 us	19,997.94 us
Maximum	174,314.22 us	282,728.77 us	21,862.31 us	21,484.98 us
Minimum	74,282.67 us	181,544.18 us	19,435.53 us	19,370.89 us
StDev	44,463.39 us	44,832.80 us	858.65 us	751.79 us
Avg/Max	0.70	0.82	0.94	0.93

*Raons per les quals s'incrementa l'overhead quan tenim més threads:*

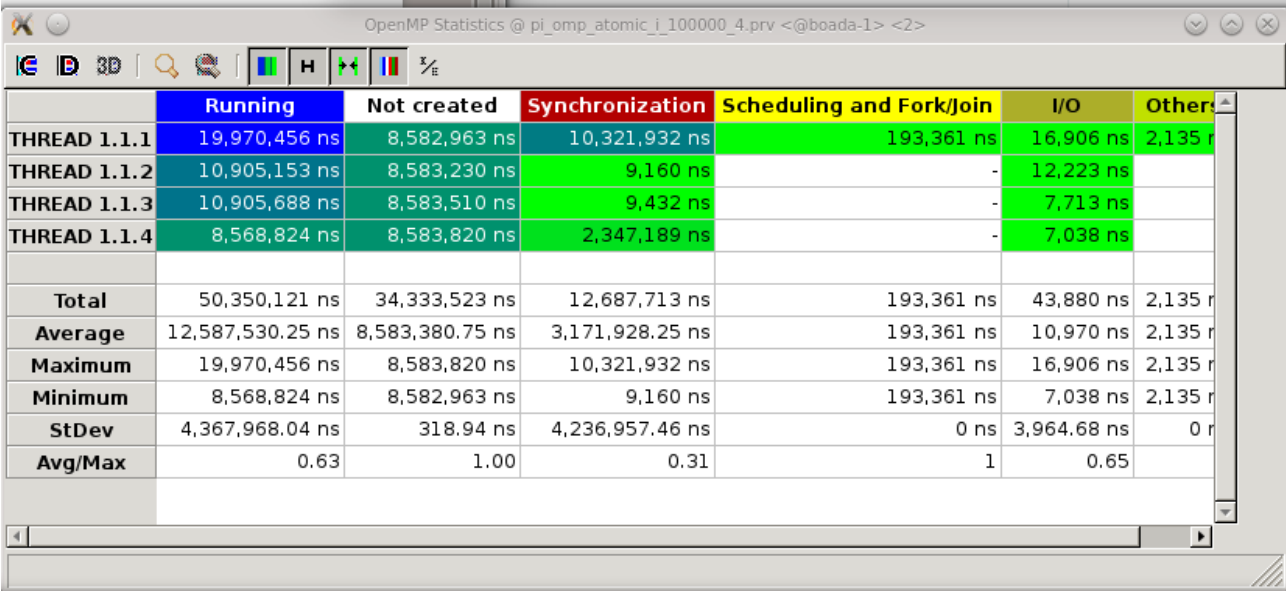
- Quan un thread s'esta executant els altres threads estan en estat de "lock", aleshores s'incrementa el temps d'overhead.
- Quan més threads tenim també incrementem el nombre de regions crítiques.
- Per cada thread que canvia d l'estat lock també s'incrementa el temps d'overhead

Les imatges les hem obtingut enviant el "programa pi\_omp\_critical" a la cua d'execució amb 1 thread i amb 8 threads, després hem obert les corresponents traces i hem aplicat la configuració "OMP\_critical\_profile.cfg".

**3. 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.**

L'ordre de magnitud es en milisegons. Veiem que com l'atòmic no bloqueja el thread el temps d'execució disminueix (la variable protegida és "sum").

Distribucio de les diferents arts del temps d'execució amb 4 threads:



	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	19,970,456 ns	8,582,963 ns	10,321,932 ns	193,361 ns	16,906 ns	2,135 r
THREAD 1.1.2	10,905,153 ns	8,583,230 ns	9,160 ns	-	12,223 ns	
THREAD 1.1.3	10,905,688 ns	8,583,510 ns	9,432 ns	-	7,713 ns	
THREAD 1.1.4	8,568,824 ns	8,583,820 ns	2,347,189 ns	-	7,038 ns	
Total	50,350,121 ns	34,333,523 ns	12,687,713 ns	193,361 ns	43,880 ns	2,135 r
Average	12,587,530.25 ns	8,583,380.75 ns	3,171,928.25 ns	193,361 ns	10,970 ns	2,135 r
Maximum	19,970,456 ns	8,583,820 ns	10,321,932 ns	193,361 ns	16,906 ns	2,135 r
Minimum	8,568,824 ns	8,582,963 ns	9,160 ns	193,361 ns	7,038 ns	2,135 r
StDev	4,367,968.04 ns	318.94 ns	4,236,957.46 ns	0 ns	3,964.68 ns	0 r
Avg/Max	0.63	1.00	0.31	1	0.65	

Distribucio de les diferents arts del temps d'execució amb 6 threads:

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	19,116,611 ns	7,549,905 ns	7,434,257 ns	370,603 ns	15,411 ns	2,140 ns
THREAD 1.1.2	7,770,087 ns	7,550,186 ns	15,583 ns	-	12,497 ns	-
THREAD 1.1.3	7,777,660 ns	7,550,528 ns	9,012 ns	-	9,392 ns	-
THREAD 1.1.4	5,873,350 ns	7,550,838 ns	1,913,365 ns	-	12,372 ns	-
THREAD 1.1.5	5,934,289 ns	7,551,178 ns	1,852,763 ns	-	5,737 ns	-
THREAD 1.1.6	7,701,092 ns	7,551,536 ns	84,728 ns	-	6,030 ns	-
Total	54,173,089 ns	45,304,171 ns	11,309,708 ns	370,603 ns	61,439 ns	2,140 ns
Average	9,028,848.17 ns	7,550,695.17 ns	1,884,951.33 ns	370,603 ns	10,239.83 ns	2,140 ns
Maximum	19,116,611 ns	7,551,536 ns	7,434,257 ns	370,603 ns	15,411 ns	2,140 ns
Minimum	5,873,350 ns	7,549,905 ns	9,012 ns	370,603 ns	5,737 ns	2,140 ns
StDev	4,586,380.42 ns	558.58 ns	2,615,693.89 ns	0 ns	3,537.86 ns	0 ns
Avg/Max	0.47	1.00	0.25	1	0.66	1

Distribucio de les diferents arts del temps d'execució amb 8 threads:

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	19,413,640 ns	8,185,152 ns	8,818,531 ns	418,601 ns	16,940 ns	2,063 ns
THREAD 1.1.2	9,074,240 ns	8,185,372 ns	6,577 ns	-	12,163 ns	-
THREAD 1.1.3	7,734,574 ns	8,185,669 ns	1,341,635 ns	-	7,633 ns	-
THREAD 1.1.4	9,056,582 ns	8,186,014 ns	24,173 ns	-	7,220 ns	-
THREAD 1.1.5	7,775,477 ns	8,186,402 ns	1,306,059 ns	-	7,023 ns	-
THREAD 1.1.6	9,039,824 ns	8,186,689 ns	41,022 ns	-	7,252 ns	-
THREAD 1.1.7	7,215,376 ns	8,187,072 ns	1,966,512 ns	-	7,777 ns	-
THREAD 1.1.8	9,073,022 ns	8,187,384 ns	46,929 ns	-	7,137 ns	-
Total	78,382,735 ns	65,489,754 ns	13,551,438 ns	418,601 ns	73,145 ns	2,063 ns
Average	9,797,841.88 ns	8,186,219.25 ns	1,693,929.75 ns	418,601 ns	9,143.12 ns	2,063 ns
Maximum	19,413,640 ns	8,187,384 ns	8,818,531 ns	418,601 ns	16,940 ns	2,063 ns
Minimum	7,215,376 ns	8,185,152 ns	6,577 ns	418,601 ns	7,023 ns	2,063 ns
StDev	3,702,241.86 ns	753.30 ns	2,788,116.79 ns	0 ns	3,351.34 ns	0 ns
Avg/Max	0.50	1.00	0.19	1	0.54	1

L'augment del temps de sincronització es degut a que a mesura que augmentem el threads, més s'hauran d'esperar.



Com abans hem obtingut els timelines i el temps d'execució del threads enviant el programa "pi\_omp\_atomic\_i" a la cua d'execució per obtenir les traces, ajustant els temps dels timelines i carregant les corresponents configuracions.

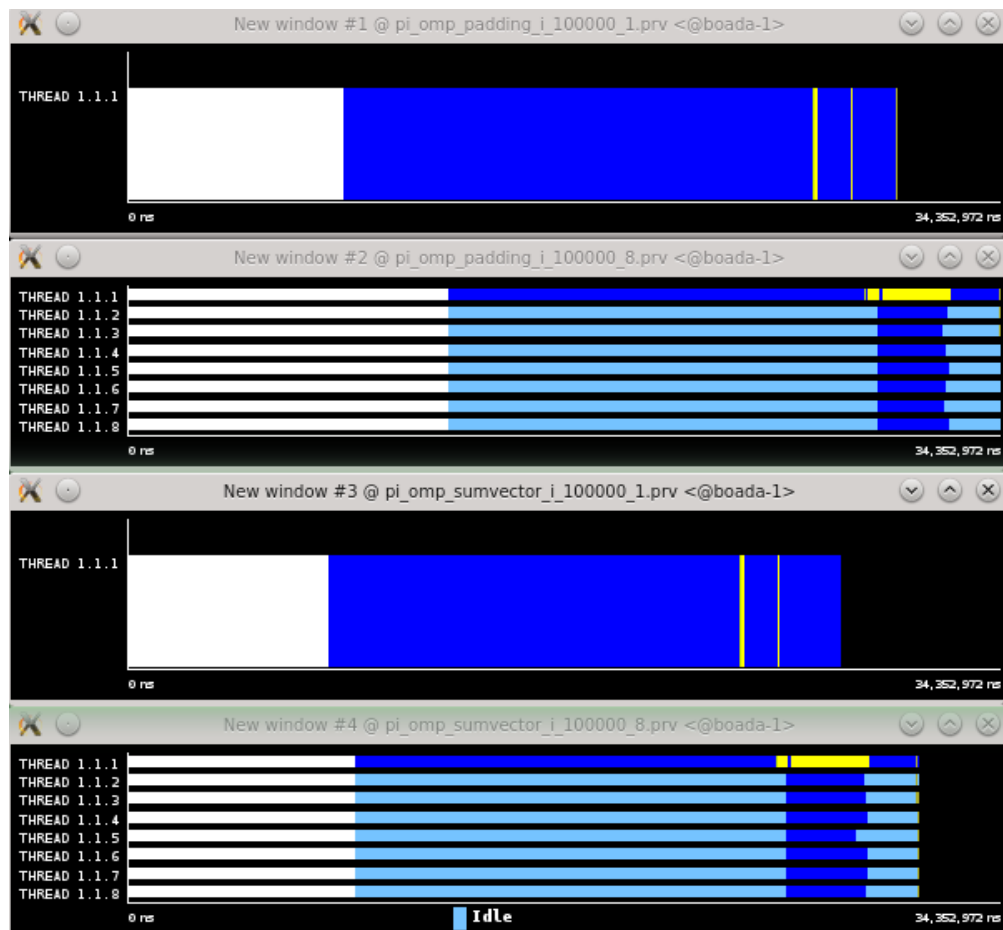
4. In the presence of false sharing (as it happens in pi omp sumvector.c), which is the additional average memory access time that you observe? What is causing this increase in the memory access time? Reason the answers based on the execution times reported by the pi omp sumvector.c and pi omp padding.c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

Com s'accedeix a la mateixa linea de cache i s'estan modificant dades de la mateixa linea es produeix aquest increment(false sharing).

Per calcular el padding i no es produeixi false sharing es calcula cuants elements hi caben en una fila de cache i es deixa aquest espai entre dos elements per a que així a cada thread només li correspongui un element.

```
double sumvector[NUMTHRDS][CACHE_SIZE/sizeof(double)];
```

Podem observar que fem servir una matriu per als calculs abans explicats.



Com abans hem obtingut el timelines enviant els programes “pi\_omp\_sumvector” i “pi\_omp\_padding” a la cua d'execució per obtenir les traces i obrint-les amb paraver.

**5. Complete the following table with the execution times of the different versions for the computation of Pi that we provide to you in this first laboratory assignment when executed with 100.000.000 iterations. The speed-up has to be computed with respect to the execution of the serial version. For each version and number of threads, how many executions have you performed?**

Hem fet la mitjana de 3 execucions

Versión	1 Procesador	8 Procesadores	Speed-up
pi seq.c	0.794842s	0.791893s	1
pi omp.c (sumlocal)	0.795380s	0.106241s	7.45
pi omp critical.c	1.833695s	21.356556s	0.03
pi omp lock.c	1.796985s	25.781719s	0.03
pi omp atomic.c	1.447889s	9.081532s	0.08
pi omp sumvector.c	0.789384s	0.580683s	1.36
pi omp padding.c	0.798572s	0.114540s	6.91

OMP\_NUM\_THREADS=1 ./nombre\_ejecutable 100000000

OMP\_NUM\_THREADS=8 ./ nombre\_ejecutable 100000000