PAR Laboratory

29/10/2014 – Q1

SECOND DELIVERABLE

group: par1107
Gabriel Carrillo
Younes Zeriahi

Parallelization overheads:

1.- Which is the order of magnitude for the overhead associated with *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.

The results obtained after executing the *submit-omp*-overhead.sh, in the *pi_omp_overhead_times.txt* file are the following:

```
All overheads expressed in microseconds
Nthr    Time    Time per thread
2       2.2746  1.1373
3       1.9408  0.6469
4       2.4296  0.6074
5       2.5665  0.5133
6       2.5969  0.4328
7       2.8748  0.4107
8       3.4449  0.4306
9       3.3394  0.3710
10      3.5315  0.3532
11      3.5555  0.3232
12      3.6340  0.3028
13      4.3363  0.3336
14      4.0780  0.2913
15      4.1044  0.2736
16      4.5237  0.2827
17      4.6151  0.2715
18      4.8671  0.2704
19      4.5367  0.2388
20      5.0281  0.2514
21      4.8314  0.2301
22      5.8846  0.2675
23      5.4224  0.2358
24      5.4928  0.2289
Number pi after 1 iterations = 0.000000000000000
Total execution time: 0.899218s
```

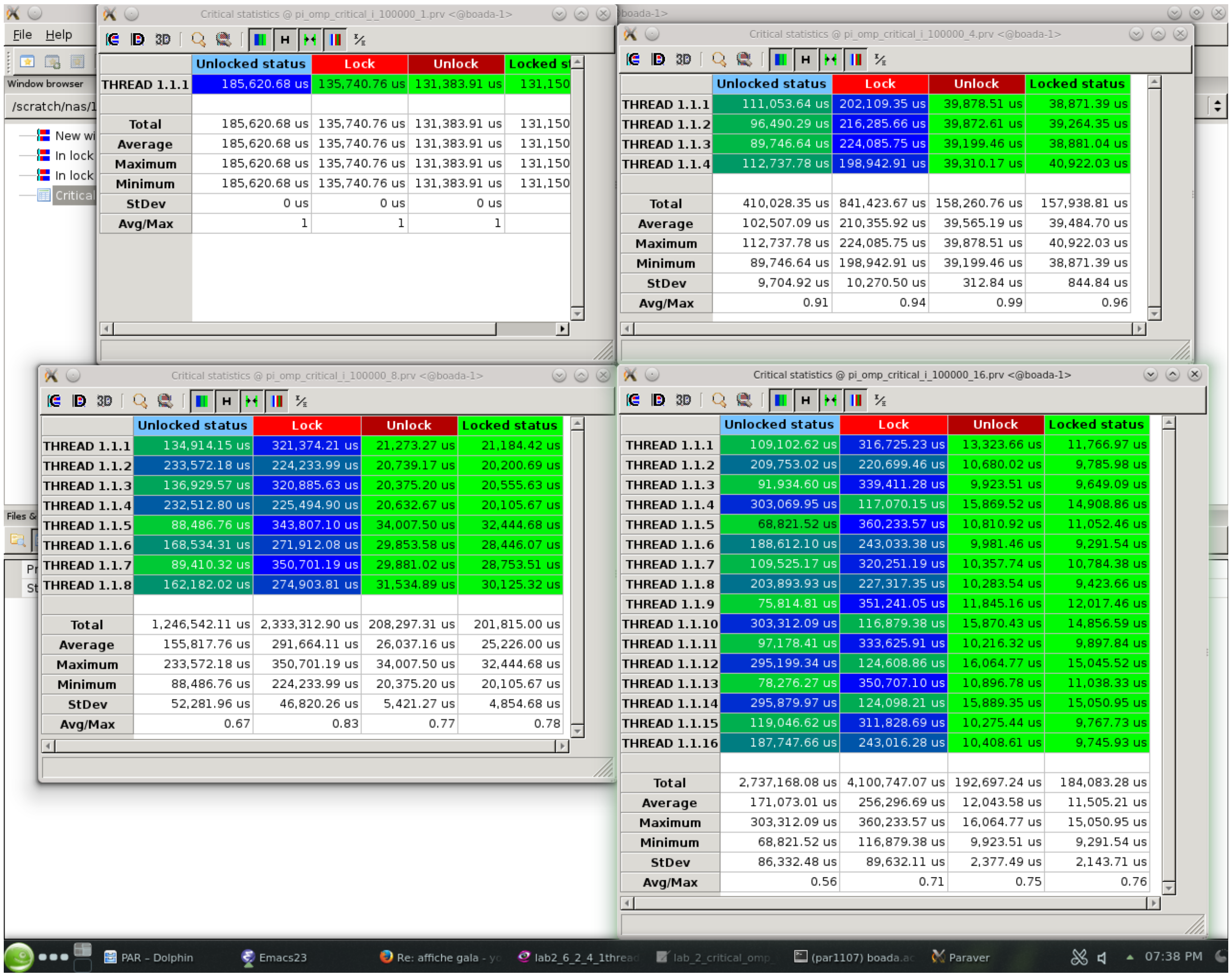
The order of magnitude for the overhead associated with *parallel* region is measured in seconds.
The order of magnitude of the total overhead is constant along the number of threads, but we notice that the total overhead slightly increases with the number of threads. This occurs because each thread adds an overhead to the total execution time.
Therefore, we can appreciate a vague decrease in the individual overhead inversely proportional to the number of threads.

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 programs and their *Paraver* execution traces.

- The order of magnitude for the overhead associated with the execution of critical regions is about several tenths of a seconds (10^(-1) seconds = 10^5 microseconds) as we can see in the following picture showing the profile of the critical sections for 1 (up left), 4 (up right), 8 (bottom left) and 16 (bottom right) threads.
- The overhead is decomposed in the three lock phases:

1. Lock: this phase corresponds to the state when the thread is waiting for the access to the critical section.
2. Lock Status: this phase corresponds to the state when the thread is in the critical section, executing this section and making this section unavailable for others threads which want to get access.
3. Unlock: this phase corresponds to the state when the thread is leaving the section and unlocking it to make it available for others threads which want to get access.

- The overhead associated with critical increases with the number of processors as shown in the following picture:



Critical statistics @ pi_omp_critical_i_100000_1.prv <@boada-1>

	Unlocked status	Lock	Unlock	Locked st
THREAD 1.1.1	185,620.68 us	135,740.76 us	131,383.91 us	131,150
Total	185,620.68 us	135,740.76 us	131,383.91 us	131,150
Average	185,620.68 us	135,740.76 us	131,383.91 us	131,150
Maximum	185,620.68 us	135,740.76 us	131,383.91 us	131,150
Minimum	185,620.68 us	135,740.76 us	131,383.91 us	131,150
StDev	0 us	0 us	0 us	
Avg/Max	1	1	1	

Critical statistics @ pi_omp_critical_i_100000_4.prv <@boada-1>

	Unlocked status	Lock	Unlock	Locked status
THREAD 1.1.1	111,053.64 us	202,109.35 us	39,878.51 us	38,871.39 us
THREAD 1.1.2	96,490.29 us	216,285.66 us	39,872.61 us	39,264.35 us
THREAD 1.1.3	89,746.64 us	224,085.75 us	39,199.46 us	38,881.04 us
THREAD 1.1.4	112,737.78 us	198,942.91 us	39,310.17 us	40,922.03 us
Total	410,028.35 us	841,423.67 us	158,260.76 us	157,938.81 us
Average	102,507.09 us	210,355.92 us	39,565.19 us	39,484.70 us
Maximum	112,737.78 us	224,085.75 us	39,878.51 us	40,922.03 us
Minimum	89,746.64 us	198,942.91 us	39,199.46 us	38,871.39 us
StDev	9,704.92 us	10,270.50 us	312.84 us	844.84 us
Avg/Max	0.91	0.94	0.99	0.96

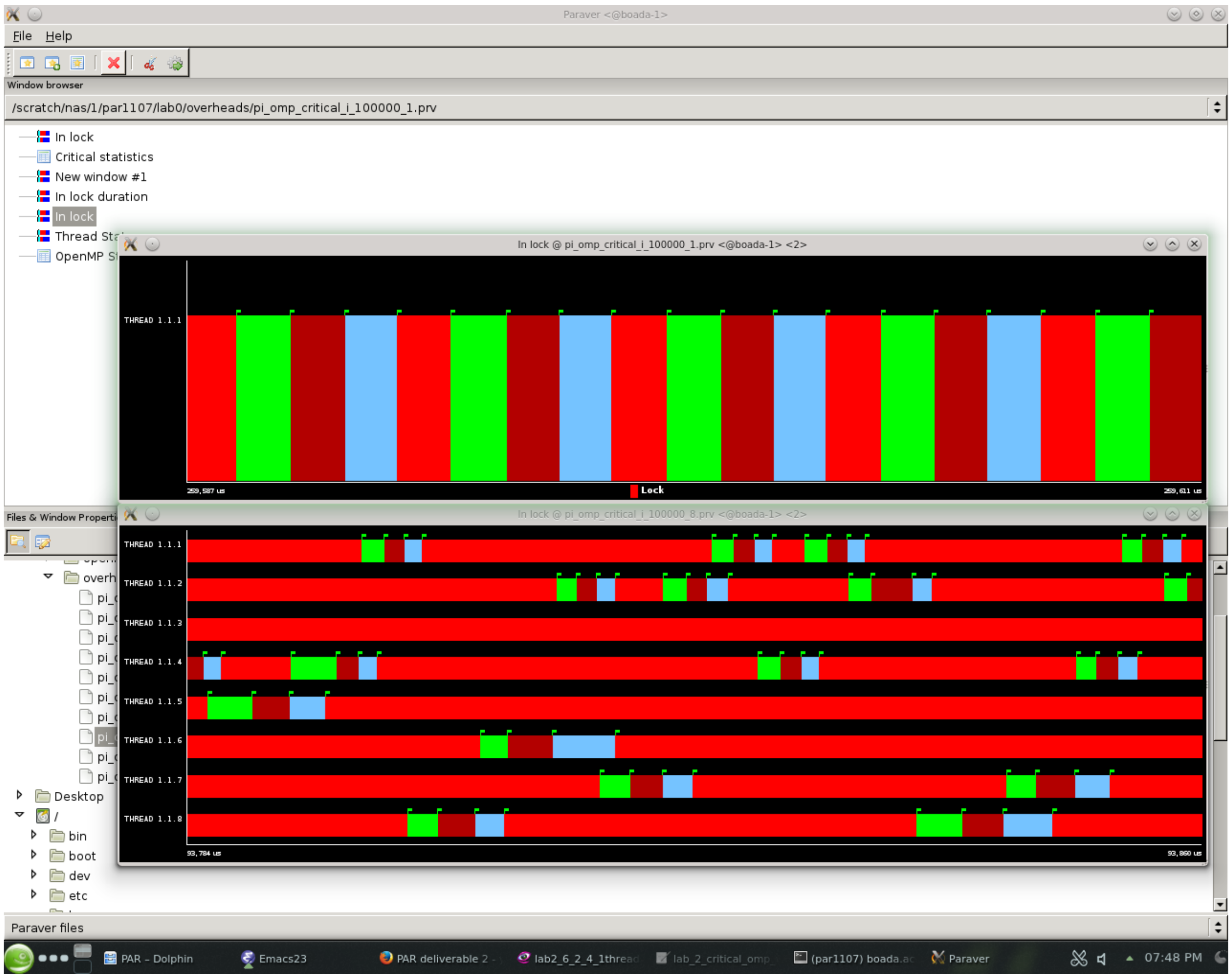
Critical statistics @ pi_omp_critical_i_100000_8.prv <@boada-1>

	Unlocked status	Lock	Unlock	Locked status
THREAD 1.1.1	134,914.15 us	321,374.21 us	21,273.27 us	21,184.42 us
THREAD 1.1.2	233,572.18 us	224,233.99 us	20,739.17 us	20,200.69 us
THREAD 1.1.3	136,929.57 us	320,885.63 us	20,375.20 us	20,555.63 us
THREAD 1.1.4	232,512.80 us	225,494.90 us	20,632.67 us	20,105.67 us
THREAD 1.1.5	88,486.76 us	343,807.10 us	34,007.50 us	32,444.68 us
THREAD 1.1.6	168,534.31 us	271,912.08 us	29,853.58 us	28,446.07 us
THREAD 1.1.7	89,410.32 us	350,701.19 us	29,881.02 us	28,753.51 us
THREAD 1.1.8	162,182.02 us	274,903.81 us	31,534.89 us	30,125.32 us
Total	1,246,542.11 us	2,333,312.90 us	208,297.31 us	201,815.00 us
Average	155,817.76 us	291,664.11 us	26,037.16 us	25,226.00 us
Maximum	233,572.18 us	350,701.19 us	34,007.50 us	32,444.68 us
Minimum	88,486.76 us	224,233.99 us	20,375.20 us	20,105.67 us
StDev	52,281.96 us	46,820.26 us	5,421.27 us	4,854.68 us
Avg/Max	0.67	0.83	0.77	0.78

Critical statistics @ pi_omp_critical_i_100000_16.prv <@boada-1>

	Unlocked status	Lock	Unlock	Locked status
THREAD 1.1.1	109,102.62 us	316,725.23 us	13,323.66 us	11,766.97 us
THREAD 1.1.2	209,753.02 us	220,699.46 us	10,680.02 us	9,785.98 us
THREAD 1.1.3	91,934.60 us	339,411.28 us	9,923.51 us	9,649.09 us
THREAD 1.1.4	303,069.95 us	117,070.15 us	15,869.52 us	14,908.86 us
THREAD 1.1.5	68,821.52 us	360,233.57 us	10,810.92 us	11,052.46 us
THREAD 1.1.6	188,612.10 us	243,033.38 us	9,981.46 us	9,291.54 us
THREAD 1.1.7	109,525.17 us	320,251.19 us	10,357.74 us	10,784.38 us
THREAD 1.1.8	203,893.93 us	227,317.35 us	10,283.54 us	9,423.66 us
THREAD 1.1.9	75,814.81 us	351,241.05 us	11,845.16 us	12,017.46 us
THREAD 1.1.10	303,312.09 us	116,879.38 us	15,870.43 us	14,856.59 us
THREAD 1.1.11	97,178.41 us	333,625.91 us	10,216.32 us	9,897.84 us
THREAD 1.1.12	295,199.34 us	124,608.86 us	16,064.77 us	15,045.52 us
THREAD 1.1.13	78,276.27 us	350,707.10 us	10,896.78 us	11,038.33 us
THREAD 1.1.14	295,879.97 us	124,098.21 us	15,889.35 us	15,050.95 us
THREAD 1.1.15	119,046.62 us	311,828.69 us	10,275.44 us	9,767.73 us
THREAD 1.1.16	187,747.66 us	243,016.28 us	10,408.61 us	9,745.93 us
Total	2,737,168.08 us	4,100,747.07 us	192,697.24 us	184,083.28 us
Average	171,073.01 us	256,296.69 us	12,043.58 us	11,505.21 us
Maximum	303,312.09 us	360,233.57 us	16,064.77 us	15,050.95 us
Minimum	68,821.52 us	116,879.38 us	9,923.51 us	9,291.54 us
StDev	86,332.48 us	89,632.11 us	2,377.49 us	2,143.71 us
Avg/Max	0.56	0.71	0.75	0.76

- The reason is because the lock/unlock statuses have the additional synchronization time between processors. The overhead associated with critical increases with the increase of lock and unlock states of each thread to ensure that at a given moment only one thread can execute the critical section. We can see this phenomenon in the following picture where the first one computes with one thread and the second one with 8 threads. We can see that the locked zones are shown in red in the pictures, and that this zones are way more present in the second one:



- One of the reasons is the time due to task creation and task termination that increases along with the number of threads. Another source of overhead is the data sharing with the variable sum. And finally, the contention, the competition for the access to this shared variable, also adds overhead associated with critical.

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.

- The order of magnitude for the overhead associated with the execution of critical regions is about several nanoseconds (10^(-9) seconds).
We executed the programs *pi_omp.c* and *pi_omp_atomic.c* with 1, 4, 8 and 16 processors and 100.000.000 iterations and got the following execution times:

	Total exec time omp	**Total exec time atomic**	**overhead**
1 processor	0.795515	1.450039	6.54524e-9
4 processors	0.205257	5.411918	52.06661e-9
8 processors	0.108060	6.285419	61.77359e-9
16 processors	0.10056	8.179054	80.78494e-9

The order of magnitude of the overhead associated to the execution of atomic memory accesses can be deduced from those four execution times knowing that the only difference between the two programs is that the second implementation adds an atomic section.
Thus, the difference between the two execution times divided by the number of iterations may give us the overhead in each case (1, 4, 8 and 16 processors).

- The overhead associated with atomic slightly increases with the number of processors, and this one adds much less overhead than the critical studied before.
This is so because the atomic instruction only works when another thread is trying to read, to operate or to write in the same memory location. This means that the more processors we have trying to read, operate or write into the same memory location, the bigger the overhead is going to be, but this overhead is going to be smaller than with the critical instruction.
In fact, the "atomic" is more permissive than the "critical". Indeed, the atomic make the bloc of instructions indivisible but it allows several threads to enter in the section and start to execute it if there is no memory conflict whereas the critical ensures that only one thread is executing the section at given time, that's why the overhead is much more important with a critical.

4.- In the presence of false sharing (as it happens in the *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 answer based on the execution times reported by the *pi_omp_sumbector.c* a n d *pi_omp_padding.c* programs (and their *Paraver* execution traces if generated). Explain how padding is done in *pi_omp_padding.c*.

- We executed the two programs *pi_omp_sumvector.c* and *pi_omp_padding.c* with 8 processors and 100 000 000 iterations and found the following execution times :
pi_omp_sumvector.c: 0.490935s
pi_omp_padding.c: 0.106353s
We can deduce the average memory access time from those two execution times knowing that the only difference between the two programs is that the second solve the false sharing problem.
Thus, the difference between the two execution times divided by the number of iterations may represent the average memory access time:

Average memory access time = (0.40935 – 0106353)/10^8 = 0.302997 * 10^(-8) = 3.02997 ns.

- This increase of memory access time is caused by the false sharing which occurs when two processors are trying to access to two distinct variables located in the same memory cache line, so it takes a longer time for each processor to get the access.
- The padding is done by creating a two dimensional array instead of a simple array. This way, we force the increase of the memory allocated for the array *sumvector* to avoid a false sharing.
We choose on purpose the size (Cache_size/sizeof(double)) of this new two dimensional array to induce the fact that each element of the original *sumvector* array perfectly fits on a cache line. Thus, it avoids the fact that more than one variable share the same cache line because it can be problematical when two distinct processors try to modify two different variables defined in the same cache line.

<u>Execution time and speed-up:</u>

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?

version	1 processor	8 processors	speed-up
pi_seq.c	0.793534s	-	1
pi_omp.c (sumlocal)	0.795515s	0.108060s	7.361789746
pi_omp_critical.c	1.839784s	16.969754s	0.108415479
pi_omp_atomic.c	1.474777s	6.285419s	0.23463463613
pi_omp_sumvector.c	0.796167s	0.490935s	1.621736075
pi_omp_padding.c	0.809432s	0.106353s	7.610805525

We have performed 5 executions for each version and number of threads. This way, we could remove the bigger and smaller outliers and calculate the average number with the three resulting values.