Práctica 2

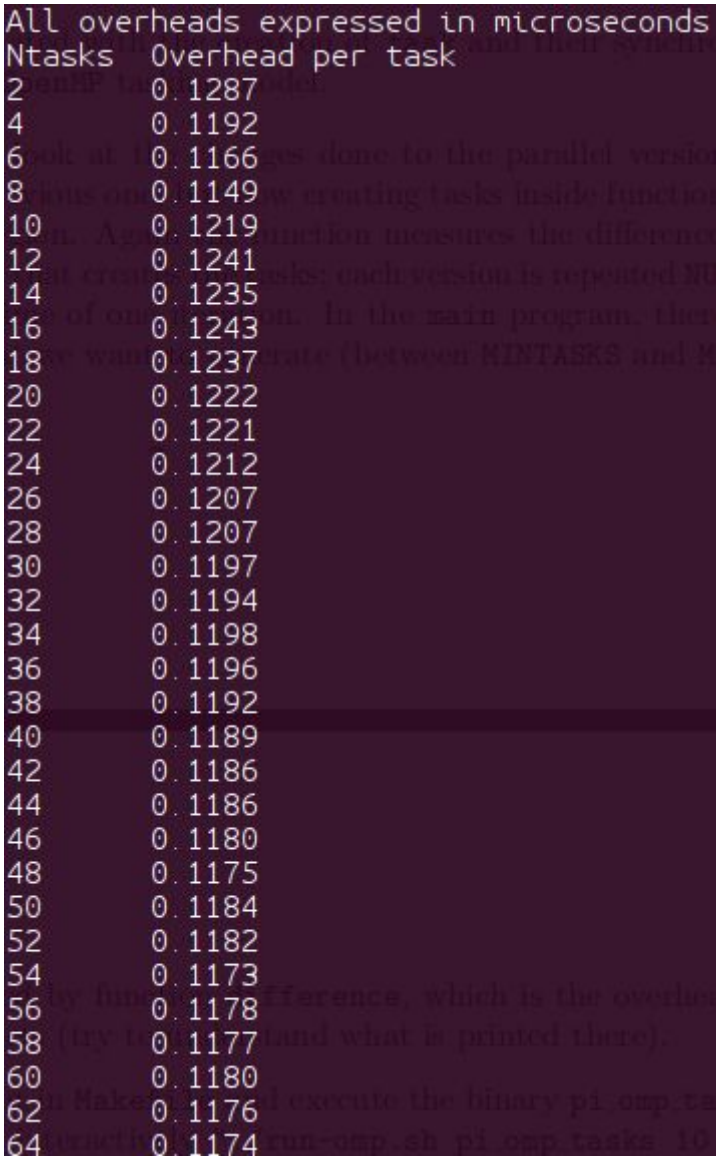
Noel Carmona García

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.

```
paco11@boada-1:~/lab2/overheads$ ./run-omp.sh pi_omp_parallel 1 24
make: 'pi_omp_parallel' is up to date.
All overheads expressed in microseconds
Nthr    Overhead        Overhead per thread
2       0.8666          0.4333
3       1.8660          0.6220
4       2.1289          0.5322
5       2.7834          0.5567
6       2.8451          0.4742
7       2.6285          0.3755
8       3.3986          0.4248
9       3.4061          0.3785
10      3.2525          0.3252
11      3.7584          0.3417
12      3.5927          0.2994
13      3.4591          0.2661
14      4.3939          0.3139
15      4.0952          0.2730
16      4.5940          0.2871
17      4.8777          0.2869
18      4.5078          0.2504
19      4.4209          0.2327
20      4.5311          0.2266
21      4.5577          0.2170
22      4.7863          0.2176
23      4.9964          0.2172
24      5.0738          0.2114
```

Tal y como vemos en la imagen el overhead no es constante. A medida que el número de threads aumenta también lo hará el tiempo de fork and join y el overhead total, sin embargo el overhead por cada thread irá bajando.

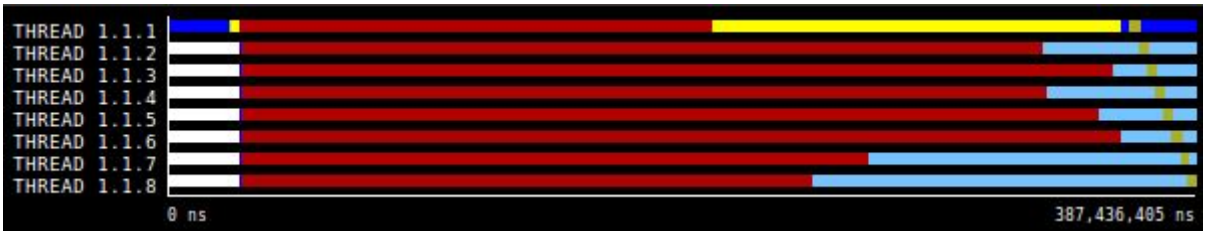
2. Which is the order of magnitude for the overhead associated with the creation of a task and its synchronization at taskwait in OpenMP? Is it constant? Reason the answer based on the results reported by the pi_omp_task.c code.



```
All overheads expressed in microseconds
Ntasks  Overhead per task
2       0.1287
4       0.1192
6       0.1166
8       0.1149
10      0.1219
12      0.1241
14      0.1235
16      0.1243
18      0.1237
20      0.1222
22      0.1221
24      0.1212
26      0.1207
28      0.1207
30      0.1197
32      0.1194
34      0.1198
36      0.1196
38      0.1192
40      0.1189
42      0.1186
44      0.1186
46      0.1180
48      0.1175
50      0.1184
52      0.1182
54      0.1173
56      0.1178
58      0.1177
60      0.1180
62      0.1176
64      0.1174
```

El overhead en mi pi_omp_task no es constante y se debe a que tenemos un taskwait que impide continuar con la ejecución del código hasta que no se termine de calcular el número pi utilizando n tareas.

3. 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 Paravel execution traces.



Hay overhead tanto al salir como al entrar en la zona de exclusión mutua, debemos tener en cuenta que los threads que no pueden acceder deben esperar, por lo tanto la bajada de rendimiento será más notable a medida que aumentamos el número de procesadores.

La bajada de rendimiento se debe a :

1. hemos definido una zona de exclusión mutua mediante #pragma omp critical al sumar sum.

2. Estamos aumentando el número de procesadores, si utilizamos un único procesador tendremos un tiempo de espera menor ya que tendremos menos hilos intentando acceder a la zona de exclusión mutua.

3.

4. Which is the order of magnitude for the overhead associated with the execution of atomic memory accesses in OpenMP? How and why the overhead associated with atomic increase with the number of processors? Reason the answers base on the execution times reported by the pi_omp.c and pi_omp_atomic.c programs.

pi_omp_atomic 100.000.000 8 threads → tiempo 6.53s

```
paco11@boada-1:~/lab2/overheads$ ./run-omp.sh pi_omp_atomic 100000000 8
make: 'pi_omp_atomic' is up to date.
Total execution time: 6.530549s
Number pi after 100000000 iterations = 3.141592653589818
51.52user 0.00system 0:06.53elapsed 788%CPU (0avgtext+0avgdata 1768maxresident)k
0inputs+0outputs (0major+166minor)pagefaults 0swaps
paco11@boada-1:~/lab2/overheads$
```

pi_omp_atomic 100.000.000 1 thread → tiempo 1.47s

```
paco11@boada-1:~/lab2/overheads$ ./run-omp.sh pi_omp_atomic 100000000 1
make: 'pi_omp_atomic' is up to date.
Total execution time: 1.471147s
Number pi after 100000000 iterations = 3.141592653590426
1.47user 0.00system 0:01.47elapsed 99%CPU (0avgtext+0avgdata 1652maxresident)k
0inputs+0outputs (0major+78minor)pagefaults 0swaps
paco11@boada-1:~/lab2/overheads$
```

pi_omp 100.000.000 8 threads → tiempo 0.18

```
paco11@boada-1:~/lab2/overheads$ ./run-omp.sh pi_omp 100000000 8
make: 'pi_omp' is up to date.
Total execution time: 0.186517s
Number pi after 100000000 iterations = 3.141592653589816
1.18user 0.00system 0:00.18elapsed 628%CPU (0avgtext+0avgdata 1852maxresident)k
0inputs+0outputs (0major+95minor)pagefaults 0swaps
paco11@boada-1:~/lab2/overheads$
```

pi_omp 100.000.000 1 thread → tiempo 0.79s

```
paco11@boada-1:~/lab2/overheads$ ./run-omp.sh pi_omp 100000000 1
make: 'pi_omp' is up to date.
Total execution time: 0.793826s
Number pi after 100000000 iterations = 3.141592653590426
0.79user 0.00system 0:00.79elapsed 99%CPU (0avgtext+0avgdata 1796maxresident)k
0inputs+0outputs (0major+81minor)pagefaults 0swaps
paco11@boada-1:~/lab2/overheads$
```

Tal y como podemos ver en las imágenes anteriores pi_omp se ejecuta más rápido pese a que utiliza zonas de exclusión mutuas para calcular sum.

5. 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

Padding

I/O
14,578 ns
11,408 ns
7,955 ns
8,686 ns
7,178 ns
7,540 ns
7,010 ns
6,975 ns
71,330 ns
8,916.25 ns

sum vector

I/O
16,098 ns
11,688 ns
8,350 ns
7,548 ns
11,859 ns
7,339 ns
6,630 ns
5,728 ns
75,240 ns
9,405 ns

El incremento del tiempo de acceso a memoria se debe a que estamos modificando el contenido de direcciones de memoria que se encuentran en la misma línea de cache por diferentes threads, esto causa que el protocolo de coherencia de la caché obligue a los threads a modificar o invalidar sus caches.

En pi_omp_sumvector utilizamos este vector:

Thrd #0	Thrd #1	Thrd #2	Thrd #3	Thrd #4	Thrd #5	Thrd #6	Thrd #7

En pi_omp_padding utilizamos esta matriz, la cual tiene una fila para cada thread y cada fila es igual de grande que una línea de caché de nuestro procesador:

Línea de caché del procesador								
Thrd #0	sumvector							
Thrd #1	sumvector							
Thrd #2	sumvector							
Thrd #3	sumvector							
Thrd #4	sumvector							
Thrd #5	sumvector							
Thrd #6	sumvector							
Thrd #7	sumvector							

Cada thread guarda la suma en su correspondiente fila y cuando las tareas finalizan se hace la suma de la posición 0 de cada fila, dicha suma será el valor de pi.

6. Write down a table (or draw a plot) showing the execution times for the different versions of the Pi computation that we provide to you in this laboratory assignment (session3) when executed with 100.000.000 iterations and the speed-up achieved with respect to the execution of the serial version pi_seq.c. For each version and number of threads, how many executions have you performed?

Versión	1 procesador	8 procesadores	speed-up
pi_seq	0.79s	-	1
pi_omp	0.79s	0.17s	4.64
pi_omp_critical	1.83s	46.24s	0.03
pi_omp_lock	2s	58.69s	0.03
pi_omp_atomic	1.47s	8.5s	0.17
pi_omp_sumvector	0.79s	0.58s	1.36
pi_omp_padding	0.78s	0.18s	4.3

He ejecutado 3 veces cada una de las versiones de 1 y 8 threads, el tiempo de ejecución ha sufrido pequeñas variaciones. Tal y como podemos ver en la tabla hay 3 casos en los que el speed-up es menor que uno, por lo que utilizar varios procesadores en esos casos no merece la pena, por otro lado tenemos pi_omp y pi_omp_padding que tienen un speed-up superior a 4. Tras ver los resultados podemos concluir que utilizar más procesadores no implica un menor tiempo de ejecución.