

Arquitecturas Avanzadas

Grupo AA-2-1

04/11/2019

Julio César Velasquez Cardenas 1397896
Sergio Prada Maeso 1459122
Juan Carlos Bermúdez Rodríguez 1455486

1. Basic parallelization

- How many times will you see the "Hello world!" message? Why?

- Vemos 8 threads, ya que son los máximos que puede utilizar la máquina:
(4 cores * 2 threads/core)

- Without changing the program, how to make it to print 5 times the "Hello World!" message? (Hint: use the OMP_NUM_THREADS environment variable).

- Añadiendo una variable de entorno llamada OMP_NUM_THREADS=5 y ejecutando el programa con ella.

- Now, do the same by changing the source code (hint: use the omp_set_num_threads () function or use a num_threads() clause at the #pragma omp parallel directive).

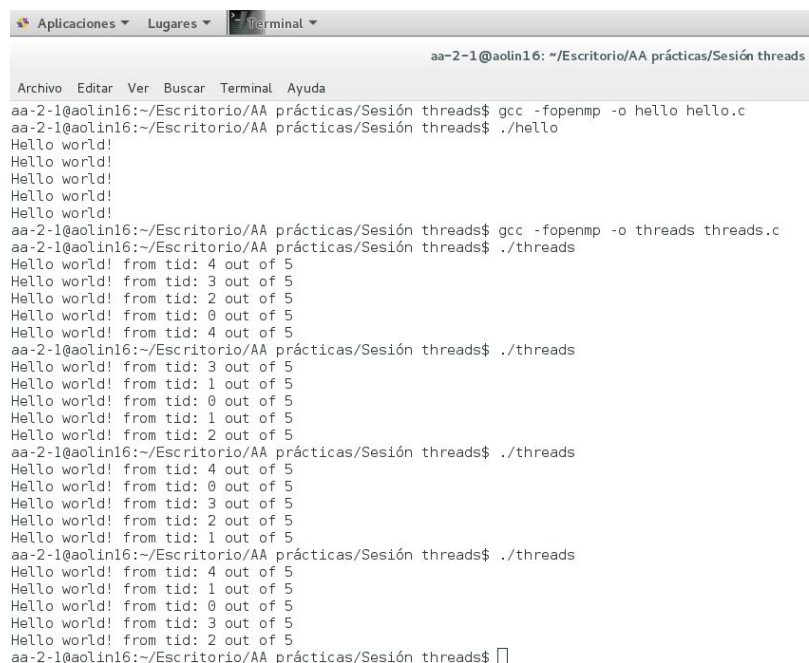
```
#include <stdio.h>
#include <omp.h>

int main ()
{
    omp_set_num_threads (5);
    #pragma omp parallel
    printf("Hello world!\n");

    return 0;
}
```

2. Thread identification

- How many messages are printed? Are they printed in any specific order (execute it several times to check it)?



```
Aplicaciones Lugares Terminal
aa-2-1@aolin16: ~/Escritorio/AA prácticas/Sesión threads
Archivo Editar Ver Buscar Terminal Ayuda
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o hello hello.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./hello
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o threads threads.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./threads
Hello world! from tid: 4 out of 5
Hello world! from tid: 3 out of 5
Hello world! from tid: 2 out of 5
Hello world! from tid: 0 out of 5
Hello world! from tid: 4 out of 5
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./threads
Hello world! from tid: 3 out of 5
Hello world! from tid: 1 out of 5
Hello world! from tid: 0 out of 5
Hello world! from tid: 1 out of 5
Hello world! from tid: 2 out of 5
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./threads
Hello world! from tid: 4 out of 5
Hello world! from tid: 0 out of 5
Hello world! from tid: 3 out of 5
Hello world! from tid: 2 out of 5
Hello world! from tid: 1 out of 5
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./threads
Hello world! from tid: 4 out of 5
Hello world! from tid: 1 out of 5
Hello world! from tid: 0 out of 5
Hello world! from tid: 3 out of 5
Hello world! from tid: 2 out of 5
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

- 5 Messages sin orden específico.

3. Data sharing

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

```
aa-2-1@aolin16: ~/Escritorio/AA prácticas/Sesión threads$ ./data_sharing
Within first parallel (shared) x is: 2
Within first parallel (shared) x is: 1
Within first parallel (shared) x is: 4
Within first parallel (shared) x is: 1
Within first parallel (shared) x is: 3
Within first parallel (shared) x is: 3
Within first parallel (shared) x is: 1
Within first parallel (shared) x is: 1
After first parallel (shared) x is: 4
Within second parallel (private) x is: 1
Within second parallel (private) x is: 32768
Within second parallel (private) x is: 32768
Within second parallel (private) x is: 32768
Within second parallel (private) x is: 32768
Within second parallel (private) x is: 32768
Within second parallel (private) x is: 32768
After second parallel (private) x is: 5
Within third parallel (first private) x is: 72
Within third parallel (first private) x is: 72
Within third parallel (first private) x is: 72
Within third parallel (first private) x is: 72
Within third parallel (first private) x is: 72
Within third parallel (first private) x is: 72
Within third parallel (first private) x is: 72
After third parallel (first private) x is: 71
aa-2-1@aolin16: ~/Escritorio/AA prácticas/Sesión threads$
```

- Para shared, todos los Threads tienen la variable “X” compartida y la modifican al mismo tiempo (provocando así errores en el acceso a la variable que se observa con el hecho que muchos threads leen y guardan valores erróneos de la variable.

- Add the necessary changes in the first parallel block to ensure that the value after it is always 8.

```
#include <stdio.h>
#include <omp.h>

int main ()
{
    omp_set_num_threads(8);

    int x=0;
    #pragma omp parallel shared(x)
    {
        #pragma omp critical
        x++;
        printf("Within first parallel (shared) x is: %d\n",x);
    }
    printf("After first parallel (shared) x is: %d\n",x);

    return 0;
}
```

- Is there any potential data race in the program?

- En este caso, hay un data race en el acceso a la variable x para incrementarla entre los threads del programa.

4. Parallel sections and data sharing

- In the first case, variables are shared by all threads. What are the potential race conditions that affect the `section_count` variable? Try out different timing scenarios (using calls to `sleep()`) and check what values are printed in this program?

```
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o sections sections.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 0
section_count 2 from thread: 1
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o sections sections.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 1
section_count 2 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o sections sections.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 1
section_count 2 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o sections sections.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 2 from thread: 0
section_count 1 from thread: 1
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o sections sections.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 7 from thread: 1
section_count 8 from thread: 0
FINAL value section_count 8 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 7 from thread: 1
section_count 8 from thread: 0
FINAL value section_count 8 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 8 from thread: 0
section_count 7 from thread: 1
FINAL value section_count 8 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 8 from thread: 1
section_count 8 from thread: 0
FINAL value section_count 8 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 8 from thread: 1
```

- La race condition en este caso es en que orden incrementan los threads la variable `section_count`.

- When `section_count` is declared as a private variable, what are the differences when `firstprivate`, `private` or `lastprivate` clauses are used? (modify the program accordingly to try out each case). Do values of `section_count` depend on the order of execution of threads now? Why?

Firstprivate()

```
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o sections sections.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 6 from thread: 1
section_count 7 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 7 from thread: 1
section_count 6 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 6 from thread: 1
section_count 7 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 6 from thread: 1
section_count 7 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 6 from thread: 1
section_count 7 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 6 from thread: 1
section_count 7 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

Private()

```
^[[Aaa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o sections sections.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 2 from thread: 1
section_count 1 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 1
section_count 2 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 1
section_count 2 from thread: 0
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 2 from thread: 0
section_count 1 from thread: 1
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 2 from thread: 0
section_count 1 from thread: 1
FINAL value section_count 5 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```


Lastprivate()

```
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o sections sections.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 1
section_count 2 from thread; 0
FINAL value section_count 1 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 0
section_count 2 from thread; 1
FINAL value section_count 1 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 1
section_count 2 from thread; 0
FINAL value section_count 1 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 1
section_count 2 from thread; 0
FINAL value section_count 1 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 1
section_count 2 from thread; 0
FINAL value section_count 1 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./sections
section_count 1 from thread: 1
section_count 2 from thread; 0
FINAL value section_count 1 from thread: 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

5. Data races

- Is the program always executing correctly?

- No

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

- Hemos añadido “#pragma omp critical” antes de hacer la operación de “x++”.Haciendo que slo un thread pueda estar en esa sección a la vez (exclusión mutua)

```
#include <stdio.h>
#include <omp.h>
#define N 1 << 10
#define NUM_THREADS 8

int main()
{
    int i, x=0;

    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel private(i)
    {
        #pragma omp critical
        {
            int id=omp_get_thread_num();
            for (i=id; i < N; i+=NUM_THREADS) {
                #pragma omp atomic
                x++;
            }
        }
    }
    printf ("N = %d \n", N);
    if (x==N) printf("Congratulations!, program executed correctly (x = %d)\n", x);
    else printf("Sorry, something went wrong, value of x = %d\n", x);

    return 0;
}
```

- La otra directiva es “#pragma omp atomic” en el mismo lugar que el anterior. Hace que las operaciones a las que hace referencia (en este caso x++) sean atómicas, es decir, no se puedan ver interrumpidas en medio de su ejecución: la operación de incrementar requiere de una lectura, modificación y guardado del resultado. Los threads podrían interrumpirse entre sí entre cualquiera de estos pasos: un thread podría estar leyendo la variable mientras otro la modifica, y daría lugar a resultados incorrectos. Con esto, nos aseguramos que la operación (o operaciones), no puedan ser interrumpidas, desde su inicio hasta su final.

```
#include <stdio.h>
#include <omp.h>
#define N 1 << 10
#define NUM_THREADS 8

int main()
{
    int i, x=0;

    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel private(i)
    {
        int id=omp_get_thread_num();
        for (i=id; i < N; i+=NUM_THREADS) {
            #pragma omp critical
            x++;
        }
    }
    printf ("N = %d \n", N);
    if (x==N) printf("Congratulations!, program executed correctly (x = %d)\n", x);
    else printf("Sorry, something went wrong, value of x = %d\n", x);

    return 0;
}
```

6. Barriers

- Can you predict the sequence of printf in this program? Do threads exit from the barrier in any specific order?

```
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o barrier barrier.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./barrier
(0) going to sleep for 2 seconds ...
(3) going to sleep for 11 seconds ...
(2) going to sleep for 8 seconds ...
(1) going to sleep for 5 seconds ...
(0) wakes up and enters barrier ...
(1) wakes up and enters barrier ...
(2) wakes up and enters barrier ...
(3) wakes up and enters barrier ...
(3) We are all awake!
(1) We are all awake!
(2) We are all awake!
(0) We are all awake!
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

- Entran en la barrera dependiendo de cuánto tiempo estaban en sleep (que va asociado al id que tenían asignado), pero no salen en un orden preestablecido, solo esperan a que todos lleguen a la barrera.

7. Basic loops and worksharing

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

```
(2) wakes up and enters barrier ...
(3) wakes up and enters barrier ...
(3) We are all awake!
(1) We are all awake!
(2) We are all awake!
(0) We are all awake!
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o for for.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./for
Going to distribute iterations in first loop ...
(3) gets iteration 12
(3) gets iteration 13
(3) gets iteration 14
(3) gets iteration 15
Going to distribute iterations in first loop ...
(0) gets iteration 0
(0) gets iteration 1
(0) gets iteration 2
(0) gets iteration 3
Going to distribute iterations in first loop ...
(5) gets iteration 20
(5) gets iteration 21
(5) gets iteration 22
(5) gets iteration 23
Going to distribute iterations in first loop ...
(6) gets iteration 24
(6) gets iteration 25
(6) gets iteration 26
(6) gets iteration 27
Going to distribute iterations in first loop ...
(7) gets iteration 28
(7) gets iteration 29
(7) gets iteration 30
(7) gets iteration 31
Going to distribute iterations in first loop ...
(1) gets iteration 4
(1) gets iteration 5
(1) gets iteration 6
(1) gets iteration 7
Going to distribute iterations in first loop ...
(4) gets iteration 16
(4) gets iteration 17
(4) gets iteration 18
(4) gets iteration 19
Going to distribute iterations in first loop ...
(2) gets iteration 8
(2) gets iteration 9
(2) gets iteration 10
(2) gets iteration 11
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

- Si no hay cláusula de schedule, por defecto se reparte el trabajo entre los threads se asigna el trabajo en un formato round-robin, haciendo que cada thread haga $\text{iter_total}/\text{num_threads}$ iteraciones.

- Add a directive so that the first "printf" is executed only once by the first thread that finds it.

```

for.c X
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/time.h>
4  #include <omp.h>      /* OpenMP */
5  #define N 32
6
7
8
9  int main()
10 {
11     int i;
12
13     omp_set_num_threads(8);
14     #pragma omp parallel
15     {
16         #pragma omp single
17         {
18             printf("Going to distribute iterations in first loop ...\n");
19         }
20         #pragma omp for
21         for (i=0; i < N; i++) {
22             int id=omp_get_thread_num();
23             printf("(%d) gets iteration %d\n",id,i);
24         }
25     }
26
27     return 0;
28 }
29

```

8. Worksharing

```

aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o hellofor hello_for.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./hellofor
Hello world! from tid: 3 out of 4
Hello world! from tid: 3 out of 4
Hello world! from tid: 3 out of 4
Hello world! from tid: 3 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 0 out of 4
Hello world! from tid: 0 out of 4
Hello world! from tid: 0 out of 4
Hello world! from tid: 0 out of 4
Second Hello world! from tid: 4 out of 6
Second Hello world! from tid: 2 out of 6
Second Hello world! from tid: 1 out of 6
Second Hello world! from tid: 5 out of 6
Second Hello world! from tid: 0 out of 6
Second Hello world! from tid: 3 out of 6
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$

```

- Modify the program so that the second loop has 6 iterations and each iteration is executed by a different thread (your solution has to be general for any number of iterations/threads)

```

for.c ✕    hello_for.c ✕
1  #include <stdio.h>
2  #include <omp.h>
3
4
5  int main ()
6  {int i, tid, nthreads;
7
8      #pragma omp parallel num_threads(4) private (i)
9      {
10         nthreads = omp_get_num_threads();
11         for (i=0; i<nthreads; i++){
12             tid = omp_get_thread_num();
13             printf("Hello world! from tid: %d out of %d \n", tid, nthreads);
14         }
15     }
16     omp_set_num_threads(6);
17     nthreads = omp_get_max_threads();
18     #pragma omp parallel
19     #pragma omp for schedule(static,1)
20     for (i=0; i< nthreads; i++){
21         tid = omp_get_thread_num();
22         printf("Second Hello world! from tid: %d out of %d of iter %d \n", tid, nthreads,i);
23     }
24     return 0;
25 }
26

```

```

aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./hellofor
Hello world! from tid: 3 out of 4
Hello world! from tid: 3 out of 4
Hello world! from tid: 3 out of 4
Hello world! from tid: 3 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 0 out of 4
Hello world! from tid: 0 out of 4
Hello world! from tid: 0 out of 4
Hello world! from tid: 0 out of 4
Second Hello world! from tid: 4 out of 6 of iter 4
Second Hello world! from tid: 0 out of 6 of iter 0
Second Hello world! from tid: 1 out of 6 of iter 1
Second Hello world! from tid: 2 out of 6 of iter 2
Second Hello world! from tid: 5 out of 6 of iter 5
Second Hello world! from tid: 3 out of 6 of iter 3
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./hellofor
Hello world! from tid: 0 out of 4
Hello world! from tid: 3 out of 4
Hello world! from tid: 3 out of 4
Hello world! from tid: 3 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 2 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 1 out of 4
Hello world! from tid: 0 out of 4
Hello world! from tid: 0 out of 4
Hello world! from tid: 0 out of 4
Hello world! from tid: 0 out of 4
Second Hello world! from tid: 5 out of 6 of iter 5
Second Hello world! from tid: 2 out of 6 of iter 2
Second Hello world! from tid: 0 out of 6 of iter 0
Second Hello world! from tid: 1 out of 6 of iter 1
Second Hello world! from tid: 4 out of 6 of iter 4
Second Hello world! from tid: 3 out of 6 of iter 3
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$

```

- Modify the original program in such a way that only one parallel region is used but you get the same output (regard that you have to guarantee that all output messages from the first loop are printed before any output from the second loop). What are the main changes that are required to guarantee the order of messages?

```
hello_for.c X
1  #include <stdio.h>
2  #include <omp.h>
3
4  int main ()
5  {int i, tid, nthreads;
6
7      #pragma omp parallel num_threads(4) private (i)
8      {
9          nthreads = omp_get_num_threads();
10         for (i=0; i<nthreads; i++){
11             tid = omp_get_thread_num();
12             printf("Hello world! from tid: %d out of %d \n", tid, nthreads);
13         }
14         #pragma omp barrier
15
16         omp_set_num_threads(6);
17         nthreads = omp_get_max_threads();
18
19
20
21         #pragma omp for
22         for (i=0; i<nthreads; i++){
23             tid = omp_get_thread_num();
24             printf("Second Hello world! from tid: %d out of %d \n", tid, nthreads);
25         }
26     }
27     return 0;
28 }
29
30
```

- Colocamos una barrera entre los dos pragmas omp, para asegurarnos de que los threads no pasan al pragma omp for hasta que todos hayan finalizado el pragma omp parallel

9. Worksharing schedule

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

```

Loop 1: (1) gets iteration 8
Loop 1: (1) gets iteration 9
Loop 1: (1) gets iteration 10
Loop 1: (1) gets iteration 11
Loop 1: (1) gets iteration 12
Loop 1: (1) gets iteration 13
Loop 1: (1) gets iteration 14
Loop 1: (1) gets iteration 15
Loop 1: (0) gets iteration 0
Loop 1: (0) gets iteration 1
Loop 1: (0) gets iteration 2
Loop 1: (0) gets iteration 3
Loop 1: (0) gets iteration 4
Loop 1: (0) gets iteration 5
Loop 1: (0) gets iteration 6
Loop 1: (0) gets iteration 7
Loop 1: (2) gets iteration 16
Loop 1: (2) gets iteration 17
Loop 1: (2) gets iteration 18
Loop 1: (2) gets iteration 19
Loop 1: (2) gets iteration 20
Loop 1: (2) gets iteration 21
Loop 1: (2) gets iteration 22
Loop 1: (2) gets iteration 23
Loop 2: (1) gets iteration 2
Loop 2: (1) gets iteration 3
Loop 2: (1) gets iteration 8
Loop 2: (1) gets iteration 9
Loop 2: (1) gets iteration 14
Loop 2: (1) gets iteration 15
Loop 2: (1) gets iteration 20
Loop 2: (1) gets iteration 21
Loop 2: (2) gets iteration 4
Loop 2: (2) gets iteration 5
Loop 2: (2) gets iteration 10
Loop 2: (2) gets iteration 11
Loop 2: (2) gets iteration 16
Loop 2: (2) gets iteration 17
Loop 2: (2) gets iteration 22
Loop 2: (2) gets iteration 23
Loop 2: (0) gets iteration 0
Loop 2: (0) gets iteration 1
Loop 2: (0) gets iteration 6
Loop 2: (0) gets iteration 7
Loop 2: (0) gets iteration 12
Loop 2: (0) gets iteration 13
Loop 2: (0) gets iteration 18
Loop 2: (0) gets iteration 19
Loop 3: (1) gets iteration 0
Loop 3: (1) gets iteration 1
Loop 3: (1) gets iteration 6
Loop 3: (1) gets iteration 7
Loop 3: (1) gets iteration 8
Loop 3: (1) gets iteration 9
Loop 3: (1) gets iteration 10
Loop 3: (1) gets iteration 11
Loop 3: (1) gets iteration 12
Loop 3: (1) gets iteration 13
Loop 3: (1) gets iteration 14
Loop 3: (1) gets iteration 15
Loop 3: (1) gets iteration 16
Loop 3: (1) gets iteration 17
Loop 3: (1) gets iteration 18
Loop 3: (1) gets iteration 19
Loop 3: (2) gets iteration 4
Loop 3: (2) gets iteration 5
Loop 3: (0) gets iteration 2
Loop 3: (0) gets iteration 3
Loop 4: (1) gets iteration 0
Loop 4: (0) gets iteration 8
Loop 4: (2) gets iteration 14
Loop 4: (0) gets iteration 9
Loop 4: (1) gets iteration 1
Loop 4: (2) gets iteration 15
Loop 4: (0) gets iteration 10
Loop 4: (1) gets iteration 2
Loop 4: (2) gets iteration 16
Loop 4: (1) gets iteration 3
Loop 4: (0) gets iteration 11
Loop 4: (2) gets iteration 17
Loop 4: (0) gets iteration 12
Loop 4: (1) gets iteration 4
Loop 4: (2) gets iteration 18
Loop 4: (0) gets iteration 13
Loop 4: (1) gets iteration 5
Loop 4: (2) gets iteration 19
Loop 4: (0) gets iteration 20
Loop 4: (1) gets iteration 6
Loop 4: (2) gets iteration 22
Loop 4: (0) gets iteration 21
Loop 4: (1) gets iteration 7
Loop 4: (2) gets iteration 23
aa-2-1@aoln16:~/Escritorio/AA prácticas/Sesión threads$

```

Loop 1	Thread 0	Iteraciones: 0-7
	Thread 1	Iteraciones: 8-15
	Thread 2	Iteraciones: 16-23
Loop 2	Thread 0	Iteraciones: 0-1 / 6-7 / 12-13 / 18-19
	Thread 1	Iteraciones: 2-3 / 8-9 / 14-15 / 20-21
	Thread 2	Iteraciones: 4-5 / 10-11 / 16-17 / 22-23
Loop 3	Thread 0	Iteraciones: 2-3
	Thread 1	Iteraciones: 0-1 / 6-7 / 8-9 / 10-23
	Thread 2	Iteraciones: 4-5
Loop 4	Thread 0	Iteraciones: 8-13 / 20-21
	Thread 1	Iteraciones: 0-7
	Thread 2	Iteraciones: 14-19 / 22-23

10. Nowait

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

```
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o nowait nowait.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./nowait
Loop 1: (1) gets iteration 2
Loop 1: (1) gets iteration 3
Loop 2: (1) gets iteration 2
Loop 2: (1) gets iteration 3
Loop 1: (3) gets iteration 6
Loop 1: (3) gets iteration 7
Loop 2: (3) gets iteration 6
Loop 2: (3) gets iteration 7
Loop 1: (2) gets iteration 4
Loop 1: (2) gets iteration 5
Loop 2: (2) gets iteration 4
Loop 2: (2) gets iteration 5
Loop 1: (0) gets iteration 0
Loop 1: (0) gets iteration 1
Loop 2: (0) gets iteration 0
Loop 2: (0) gets iteration 1
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

- Los Threads ejecutan las iteraciones del siguiente Loop antes que los demás hayan ejecutado todas sus acciones antes.

- If the nowait clause is removed in the second for pragma, will you observe any difference?

```
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o nowait nowait.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./nowait
Loop 1: (1) gets iteration 2
Loop 1: (1) gets iteration 3
Loop 2: (1) gets iteration 2
Loop 2: (1) gets iteration 3
Loop 1: (3) gets iteration 6
Loop 1: (3) gets iteration 7
Loop 2: (3) gets iteration 6
Loop 2: (3) gets iteration 7
Loop 1: (2) gets iteration 4
Loop 1: (2) gets iteration 5
Loop 2: (2) gets iteration 4
Loop 2: (2) gets iteration 5
Loop 1: (0) gets iteration 0
Loop 1: (0) gets iteration 1
Loop 2: (0) gets iteration 0
Loop 2: (0) gets iteration 1
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o nowait nowait.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./nowait
Loop 1: (0) gets iteration 0
Loop 1: (0) gets iteration 1
Loop 2: (0) gets iteration 0
Loop 2: (0) gets iteration 1
Loop 1: (2) gets iteration 4
Loop 1: (2) gets iteration 5
Loop 2: (2) gets iteration 4
Loop 2: (2) gets iteration 5
Loop 1: (1) gets iteration 2
Loop 1: (1) gets iteration 3
Loop 2: (1) gets iteration 2
Loop 2: (1) gets iteration 3
Loop 1: (3) gets iteration 6
Loop 1: (3) gets iteration 7
Loop 2: (3) gets iteration 6
Loop 2: (3) gets iteration 7
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

- No. El segundo Loop sigue haciendo las mismas iteraciones con los mismos threads que hicieron las iteraciones del Loop anterior.

11. Collapse

- Which iterations of the loops are executed by each thread when the collapse clause is used?

```
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o collapse collapse.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./collapse
(2) Iter (1 4)
(2) Iter (1 5)
(2) Iter (2 0)
(2) Iter (2 1)
(2) Iter (2 2)
(7) Iter (5 2)
(7) Iter (5 3)
(7) Iter (5 4)
(7) Iter (5 5)
(0) Iter (0 0)
(0) Iter (0 1)
(0) Iter (0 2)
(0) Iter (0 3)
(0) Iter (0 4)
(4) Iter (3 2)
(4) Iter (3 3)
(4) Iter (3 4)
(4) Iter (3 5)
(5) Iter (4 0)
(5) Iter (4 1)
(5) Iter (4 2)
(5) Iter (4 3)
(3) Iter (2 3)
(3) Iter (2 4)
(3) Iter (2 5)
(3) Iter (3 0)
(3) Iter (3 1)
(1) Iter (0 5)
(1) Iter (1 0)
(1) Iter (1 1)
(1) Iter (1 2)
(1) Iter (1 3)
(6) Iter (4 4)
(6) Iter (4 5)
(6) Iter (5 0)
(6) Iter (5 1)
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

- Is the execution correct if we remove the collapse clause? Add the appropriate clause to make it correct.

schedule.c	collapse.c
<pre>1 #include <stdio.h> 2 #include <stdlib.h> 3 #include <sys/time.h> 4 #include <omp.h> /* OpenMP */ 5 #define N 6 6 7 8 int main() 9 { 10 int i,j; 11 12 omp_set_num_threads(8); 13 #pragma omp parallel for private(i,j) 14 for (i=0; i < N; i++) { 15 for (j=0; j < N; j++) { 16 int id=omp_get_thread_num(); 17 printf("(%) Iter (%d %d)\n",id,i,j); 18 } 19 } 20 21 return 0; 22 } 23</pre>	<pre>aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads\$ gcc -fopenmp -o collapse collapse.c aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads\$./collapse (2) Iter (2 0) (2) Iter (2 1) (2) Iter (2 2) (2) Iter (2 3) (2) Iter (2 4) (2) Iter (2 5) (5) Iter (5 0) (0) Iter (0 0) (4) Iter (4 0) (3) Iter (3 0) (1) Iter (1 0) aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads\$</pre>

12. Ordered

- Can you explain the order in which printf appear?

```
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o ordered ordered.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./ordered
Before ordered - (3) gets iteration 0
Inside ordered - (3) gets iteration 0
Before ordered - (3) gets iteration 8
Before ordered - (6) gets iteration 2
Before ordered - (2) gets iteration 3
Before ordered - (5) gets iteration 5
Before ordered - (0) gets iteration 6
Before ordered - (4) gets iteration 4
Before ordered - (1) gets iteration 1
Inside ordered - (1) gets iteration 1
Before ordered - (1) gets iteration 9
Inside ordered - (6) gets iteration 2
Inside ordered - (2) gets iteration 3
Before ordered - (2) gets iteration 11
Before ordered - (7) gets iteration 7
Inside ordered - (4) gets iteration 4
Before ordered - (4) gets iteration 12
Inside ordered - (5) gets iteration 5
Before ordered - (5) gets iteration 13
Inside ordered - (0) gets iteration 6
Before ordered - (0) gets iteration 14
Inside ordered - (7) gets iteration 7
Before ordered - (7) gets iteration 15
Inside ordered - (3) gets iteration 8
Inside ordered - (1) gets iteration 9
Before ordered - (6) gets iteration 10
Inside ordered - (6) gets iteration 10
Inside ordered - (2) gets iteration 11
Inside ordered - (4) gets iteration 12
Inside ordered - (5) gets iteration 13
Inside ordered - (0) gets iteration 14
Inside ordered - (7) gets iteration 15
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

- No, aparecen de forma desordenada.

- 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?

```
schedule.c % collapse.c % ordered.c %
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/time.h>
4  #include <omp.h> /* OpenMP */
5  #define N 16
6
7  int main()
8  {
9      int i;
10
11      omp_set_num_threads(8);
12      #pragma omp parallel
13      {
14          #pragma omp for schedule(dynamic,2) ordered
15          for (i=0; i < N; i++) {
16              int id=omp_get_thread_num();
17              printf("Before ordered - (%d) gets iteration %d\n",id,i);
18              #pragma omp ordered
19              printf("Inside ordered - (%d) gets iteration %d\n",id,i);
20          }
21      }
22
23      return 0;
24  }
25
```

- Con la cláusula `schedule(dynamic,2)`, asignamos el trabajo en fracciones de 2 iteraciones, y con `dynamic` provocamos que cualquier thread que termine con sus iteraciones automáticamente haga las 2 siguientes disponibles que no este haciendo otro thread ya, a diferencia de `static`, en el que los threads se asignan las iteraciones en este orden: T1 (0,1) T2(2,3) T3(4,5) T1(6,7) T2(7,8)...

13. Doacross

- In which order are the "Outside" and "Inside" messages printed? What would happen at the contents of arrays b and c if the depend clause is removed?

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

```
aa-2-l@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o doacross doacross.c
aa-2-l@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./doacross
Outside from 0 executing 1
Inside from 0 executing 1
Outside from 0 executing 9
Outside from 7 executing 5
Outside from 3 executing 8
Outside from 1 executing 4
Outside from 5 executing 3
Inside from 5 executing 3
Outside from 5 executing 10
Outside from 6 executing 7
Outside from 4 executing 2
Inside from 4 executing 2
Outside from 4 executing 11
Inside from 7 executing 5
Outside from 7 executing 12
Inside from 6 executing 7
Inside from 0 executing 9
Outside from 0 executing 14
Outside from 6 executing 13
Inside from 1 executing 4
Outside from 1 executing 15
Outside from 2 executing 6
Inside from 2 executing 6
Inside from 3 executing 8
Inside from 5 executing 10
Inside from 7 executing 12
Inside from 0 executing 14
Inside from 4 executing 11
Inside from 6 executing 13
Inside from 1 executing 15
Computing iteration 1 1
Computing iteration 1 2
Computing iteration 2 1
Computing iteration 1 3
Computing iteration 2 2
Computing iteration 3 1
Computing iteration 1 4
Computing iteration 2 3
Computing iteration 3 2
Computing iteration 4 1
Computing iteration 2 4
Computing iteration 3 3
Computing iteration 4 2
Computing iteration 3 4
Computing iteration 4 3
Computing iteration 4 4
aa-2-l@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

- Que el array de b y c actualizará en paralelo sus valores con valores de array incorrectos ya que utiliza posiciones antiguas del array cuyo thread encargado de actualizarlas no ha actualizado todavía.

- What would happen if you remove the invocation of sleep(1). Execute several times to answer in the general case. Can you explain the order in which printf appear?

- No se predicen los prints. Con el Sleep(1); Aseguramos que se impriman en el siguiente orden:
1 1 - 1 2 - 1 3 ...
Si le quitamos el sleep(1); Hacemos que se imprima todo seguido y sin orden alguno.

14. Tasking

- Is the code printing what you expect? How many tasks are created and what computation is associated to each task? Is the code executing tasks in parallel?

```
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o fibonacci fibonacci.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./fibonacci
Starting computation of Fibonacci for numbers in linked list
Finished computation of Fibonacci for numbers in linked list
0: 1 computed by thread 0
1: 1 computed by thread 0
2: 2 computed by thread 0
3: 3 computed by thread 0
4: 5 computed by thread 0
5: 8 computed by thread 0
6: 13 computed by thread 0
7: 21 computed by thread 0
8: 34 computed by thread 0
9: 55 computed by thread 0
10: 89 computed by thread 0
11: 144 computed by thread 0
12: 233 computed by thread 0
13: 377 computed by thread 0
14: 610 computed by thread 0
15: 987 computed by thread 0
16: 1597 computed by thread 0
17: 2584 computed by thread 0
18: 4181 computed by thread 0
19: 6765 computed by thread 0
20: 10946 computed by thread 0
21: 17711 computed by thread 0
22: 28657 computed by thread 0
23: 46368 computed by thread 0
24: 75025 computed by thread 0
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$
```

- Todo lo esta haciendo un solo thread, y cada tarea es una fracción del cálculo de fibonacci, pero no es en paralelo, ya que lo está haciendo todo el thread 0 solo.

- Insert the necessary pragmas to execute the code in parallel (Hint: you have to include a directive to identify which part of the code has to be executed in parallel; think also about whether variables have to be shared or private to each thread; and which thread has to be responsible for task creation).

```

aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ gcc -fopenmp -o fibonacci fibonacci.c
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ ./fibonacci
Starting computation of Fibonacci for numbers in linked list
Finished computation of Fibonacci for numbers in linked list
0: 1 computed by thread 0
1: 1 computed by thread 1
2: 2 computed by thread 2
3: 3 computed by thread 0
4: 5 computed by thread 2
5: 8 computed by thread 0
6: 13 computed by thread 2
7: 21 computed by thread 2
8: 34 computed by thread 0
9: 55 computed by thread 1
10: 89 computed by thread 2
11: 144 computed by thread 0
12: 233 computed by thread 1
13: 377 computed by thread 2
14: 610 computed by thread 0
15: 987 computed by thread 1
16: 1597 computed by thread 2
17: 2584 computed by thread 0
18: 4181 computed by thread 3
19: 6765 computed by thread 2
20: 10946 computed by thread 1
21: 17711 computed by thread 0
22: 28657 computed by thread 3
23: 46368 computed by thread 2
24: 75025 computed by thread 1
aa-2-1@aolin16:~/Escritorio/AA prácticas/Sesión threads$ █

```

```

41  █   for (i=0; i<nelems-1; i++) {
42      p2 = malloc(sizeof(struct node));
43      p1->next = p2;
44      p2->data = i+1;
45      p2->fibdata = 0;
46      p2->threadnum = 0;
47      p1 = p2;
48  }
49  p1->next = NULL;
50  return head;
51  }
52
53  █   int main(int argc, char *argv[]) {
54      struct node *p, *temp, *head;
55
56      printf("Starting computation of Fibonacci for numbers in linked list \n");
57
58      p = init_list(N);
59      head = p;
60
61      #pragma omp parallel num_threads(4)
62      {
63          #pragma omp single
64          while (p != NULL) {
65              #pragma omp task
66              processwork(p);
67              p = p->next;
68          }
69      }
70
71      printf("Finished computation of Fibonacci for numbers in linked list \n");
72      p = head;
73      while (p != NULL) {
74          printf("%d: %d computed by thread %d \n", p->data, p->fibdata, p->threadnum);
75          temp = p->next;
76          free (p);
77          p = temp;
78      }
79      free (p);
80
81      return 0;
82  }
83

```


15. Taskloop

- 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.

```
aa-2-1@aolin-login:~/Escritorio/AA prácticas/Sesión threads$ ./taskloop
I am thread 2 and going to create T1 and T2
I am still thread 2 after creating T1 and T2, ready to enter in the taskwait
Thread 1 going to sleep for 5 seconds
Thread 2 finished the execution of task creating T3 and T4
Thread 3 executing loop body (1, 0)
Thread 0 going to sleep for 10 seconds
Thread 3 executing loop body (2, 0)
Thread 3 executing loop body (2, 1)
Thread 3 executing loop body (3, 0)
Thread 3 executing loop body (3, 1)
Thread 1 weaking up after a 5 seconds siesta, willing to work ...
I am still thread 2, but now after exiting from the taskwait
Thread 3 executing loop body (3, 2)
Thread 3 executing loop body (4, 0)
Thread 3 executing loop body (4, 1)
Thread 3 executing loop body (4, 2)
Thread 3 executing loop body (4, 3)
Thread 0 weaking up after a 10 seconds siesta, willing to work ...
Thread 3 executing loop body (5, 0)
Thread 3 executing loop body (5, 1)
Thread 3 executing loop body (5, 2)
Thread 3 executing loop body (5, 3)
Thread 3 executing loop body (5, 4)
Thread 3 executing loop body (6, 0)
Thread 3 executing loop body (6, 1)
Thread 3 executing loop body (6, 2)
Thread 3 executing loop body (6, 3)
Thread 3 executing loop body (6, 4)
Thread 3 executing loop body (6, 5)
Thread 3 executing loop body (7, 0)
Thread 3 executing loop body (7, 1)
Thread 3 executing loop body (7, 2)
Thread 3 executing loop body (7, 3)
Thread 3 executing loop body (7, 4)
Thread 3 executing loop body (7, 5)
Thread 3 executing loop body (7, 6)
Thread 3 executing loop body (8, 0)
Thread 3 executing loop body (8, 1)
Thread 3 executing loop body (8, 2)
Thread 3 executing loop body (8, 3)
```

```
Thread 3 executing loop body (8, 4)
Thread 3 executing loop body (8, 5)
Thread 3 executing loop body (8, 6)
Thread 3 executing loop body (8, 7)
Thread 3 executing loop body (9, 0)
Thread 3 executing loop body (9, 1)
Thread 3 executing loop body (9, 2)
Thread 3 executing loop body (9, 3)
Thread 3 executing loop body (9, 4)
Thread 3 executing loop body (9, 5)
Thread 3 executing loop body (9, 6)
Thread 3 executing loop body (9, 7)
Thread 3 executing loop body (9, 8)
Thread 3 finished the creation of all tasks in taskloop TL
```