CI-0117 Laboratorio 9 Paralela y Concurrente
Rodrigo Vilchez Ulloa | B78292

---

1. Programa `omp_hello.c`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 9$ ./a.out 4
Hello from thread 0 of 4
Hello from thread 3 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 9$ ./a.out 4
Hello from thread 3 of 4
Hello from thread 0 of 4
Hello from thread 2 of 4
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 9$
```

La salida varía pues, al ejecutarlo una vez, puede que un hilo termine antes que el resto y logre imprimir su mensaje en pantalla primero y al ejecutarlo una segunda vez, un hilo distinto puede ser que termine primero en comparación con la ejecución anterior.

2. Programa `omp_fibo_c`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 10 20
The first n Fibonacci numbers:
0       1
1       1
2       2
3       3
4       38
5       39
6       77
7       116
8       193
9       309
10      502
11      811
12      7235443401905431948
13      7235443401905457912
14      -3975857269898661756
15      3259586132006796156
16      7954894494947082388
17      7954894494947082504
18      -2536955083815386724
19      -2677976203439374040
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$
```

Efectivamente el programa produce una salida errónea, esto se debe a que, primero, la asignación de iteraciones para un trabajador las realiza el sistema y no el programador, esto genera que puede que un trabajador trate de utilizar un valor de `fibo[i]` que debe ser calculado por otro trabajador pero este simplemente aún no lo ha calculado todavía, entonces el trabajador que utiliza ese valor termina calculando su resultado con datos "basura".

3. Programa `omp_private.c`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 9$ ./a.out 4
Thread 0 > before initialization, x = 0
Thread 0 > after initialization, x = 2
Thread 3 > before initialization, x = 0
Thread 3 > after initialization, x = 8
Thread 2 > before initialization, x = 0
Thread 2 > after initialization, x = 6
Thread 1 > before initialization, x = 0
Thread 1 > after initialization, x = 4
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 9$
```

Al hacer uso de la claúsula `private`, todos los hilos tendrán una copia privada de todas las variables declaradas dentro de los paréntesis, por tanto su uso dentro del bloque paralelo, por parte de un hilo, no va a afectar el resultado que pueda mostrar otro hilo al hacer uso de la misma variable.

4. Programa `omp_trap1.c`

Prueba con `# pragma omp critical`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 8
Enter a, b, and n
2
200
16
With n = 16 trapezoids, our estimate
of the integral from 2.000000 to 200.000000 = 2.67171764062500e+06
```

Prueba sin `# pragma omp critical`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 8
Enter a, b, and n
2
200
16
With n = 16 trapezoids, our estimate
of the integral from 2.000000 to 200.000000 = 1.80994719726562e+06
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
```

Los resultados en ambas pruebas para la integral son distintos pues claramente hay una condición de carrera al intentar escribir en la variable global que almacena el resultado final de la integral, y puede que varios hilos modifiquen la variable al mismo tiempo y que no se tome en cuenta algún resultado calculado por un hilo. Con `# pragma omp critical`, se define una sección crítica donde solo un hilo puede acceder a ese bloque y se evita la condición de carrera.

5. Programa `omp_trap2.c`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 8
Enter a, b, and n
2
200
16
With n = 16 trapezoids, our estimate
of the integral from 2.000000 to 200.000000 = 2.67171764062500e+06
```

El resultado es el mismo, se cambia un poco la implementación del problema pero la sección crítica de sumar el resultado final local al resultado final global sigue estando protegida de condiciones de carrera.

6. Programa `omp_trap2a.c`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 8
Enter a, b, and n
2
200
16
With n = 16 trapezoids, our estimate
of the integral from 2.000000 to 200.000000 = 2.67171764062500e+06
```

El resultado es el mismo, se cambia un poco la implementación del problema pero la sección crítica de sumar el resultado final local al resultado final global sigue estando protegida de condiciones de carrera.

7. Programa `omp_trap2b.c`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 8
Enter a, b, and n
2
200
16
With n = 16 trapezoids, our estimate
of the integral from 2.000000 to 200.000000 = 2.67171764062500e+06
```

El resultado es el mismo, se utiliza `reduction clause`, útil cuando se aplica reiteradamente una misma operación sobre una misma variable, en este caso del tipo `+=`, lo que hace OpenMP en este caso es que crea una variable privada para cada hilo y almacena el resultado de ese hilo e internamente crea una sección crítica donde estas variables privadas se suman en la variable compartida.

8. Programa `omp_trap3.c`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 8
Enter a, b, and n
2
200
16
With n = 16 trapezoids, our estimate
of the integral from 2.000000 to 200.000000 = 2.67171764062500e+06
```

El resultado es el mismo, se cambia la implementación del programa utilizando `# pragma omp parallel for` que indica que existe una operación que debe ser tratada bajo un `reduction clause`.

9. Programa `omp_pi.c`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ gcc omp_pi.c -fopenmp
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 10 1000
With n = 1000 terms and 10 threads,
   Our estimate of pi = 3.14059265383979
                pi = 3.14159265358979
```

Para calcular el resultado, se indica con `# pragma omp parallel for` que el ciclo debe realizarse en paralelo, de igual forma, con el `reduction clause` se evitan condiciones de carrera al momento de sumar un resultado local en la variable global que almacena el resultado final de la aproximación de pi. El último detalle es que el ciclo utiliza la variable `factor`, la cual ha sido declarada fuera del bloque en paralelo, por lo que se trata de una variable compartido, pero por

la naturaleza de la sumatoria, donde se debe ir alternando ese factor entre negativo y positivo, es posible que un hilo pueda modificar ese valor antes de que otro hilo la vaya a utilizar, lo que causaría resultados incorrectos. Es por esto que se indica que esa variable será privada, de esta forma, cada hilo crea una copia de la variable y no afectará el resultado en otros hilos.

10. Programa `bubble.c`

Correr el programa.

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 50 g
Before sort:
83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29 82
 30 62 23 67 35 29 2 22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70


After sort:
2 11 11 15 15 19 21 21 22 23 24 26 26 27 29 29 29 30 35 35 36 37 40 42 49
56 58 59 62 62 63 67 67 67 68 69 70 72 73 77 82 83 84 86 86 90 92 93 93 98
```

Correr el programa y tomar el tiempo.

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 50 g
Before sort:
83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29 82
 30 62 23 67 35 29 2 22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70


Tiempo: 0.000019
```

Correr el programa con Valgrind.

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ valgrind ./a.out 50 g
==25629== Memcheck, a memory error detector
==25629== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==25629== Using Valgrind-3.16.0 and LibVEX; rerun with -h for copyright in
fo
==25629== Command: ./a.out 50 g
==25629==
Before sort:
83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29 82
 30 62 23 67 35 29 2 22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70


Tiempo: 0.001121
After sort:
2 11 11 15 15 19 21 21 22 23 24 26 26 27 29 29 29 30 35 35 36 37 40 42 49
56 58 59 62 62 63 67 67 67 68 69 70 72 73 77 82 83 84 86 86 90 92 93 93 98


==25629==
==25629== HEAP SUMMARY:
==25629==     in use at exit: 0 bytes in 0 blocks
==25629==   total heap usage: 2 allocs, 2 frees, 1,224 bytes allocated
==25629==
==25629== All heap blocks were freed -- no leaks are possible
==25629==
==25629== For lists of detected and suppressed errors, rerun with: -s
==25629== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

No se muestra que haya alguna fuga de memoria pues todos los bytes asignados fueron liberados.

11. Programa `omp_odd_even1.c`

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Sema
na 9$ ./a.out 8 50 g
Elapsed time = 0.006433 seconds
```

El ordenamiento parece verse afectado por el uso de hilos pues `bubble.c` es considerablemente más rápido.

12. Programa `omp_sin_sum.c`

Correr programa con 1 hilo y 10000 términos.

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 9
$ ./a.out 1 10000
Result = 1.88892327977909e+00
Check = 1.88892327977974e+00
With n = 10000 terms, the error is 6.45261621912141e-13
Elapsed time = 1.065039e+00 seconds
```

Correr programa con 2 hilos y 10000 términos.

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Laboratorios/Semana 9
$ ./a.out 2 10000
Result = 1.88892327977889e+00
Check = 1.88892327977913e+00
With n = 10000 terms, the error is 2.45581333047085e-13
Elapsed time = 5.672073e-01 seconds
```

El speedup es de 1.8776891623221355, el programa con dos hilos es ligeramente más rápido.

Correr programa con `OMP_SCHEDULE="dynamic, static, auto"`

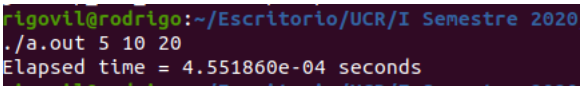
```
$ OMP_SCHEDULE="dynamic"
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117
$ ./a.out 2 10000
Result = 1.88892327977970e+00
Check = 1.88892327977913e+00
With n = 10000 terms, the error is 5.71542813077031e-13
Elapsed time = 5.650158e-01 seconds
$ OMP_SCHEDULE="auto"
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-011
$ ./a.out 2 10000
Result = 1.88892327977904e+00
Check = 1.88892327977913e+00
With n = 10000 terms, the error is 9.28146448586631e-14
Elapsed time = 5.686576e-01 seconds
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-011
$ ./a.out 2 10000
Result = 1.88892327977906e+00
Check = 1.88892327977913e+00
With n = 10000 terms, the error is 6.86117829218347e-14
Elapsed time = 5.685360e-01 seconds
```

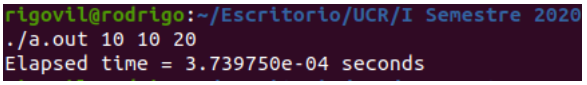
No parece haber diferencias significativas entre los tres casos, sin embargo, el caso dinámico parece indicar que es ligeramente más rápido.

13. Programa `omp_msgps.c`

Lo que sucede en el programa es una comunicación mediante el paso de mensajes entre hilos, cada hilo tiene una cola de mensajes el cual es capaz de recibir una capacidad máxima de mensajes, los mensajes se envían a un destinatario aleatorio. Con `barrier` se logra que los hilos creen y asignen memoria a las colas respectivas y que ninguno comience a enviar y recibir mensajes hasta que todos los hilos hayan logrado asignar ese espacio. Con `atomic` se logra que se acceda a una sección de la memoria en específico de manera atómica, es decir, solo un hilo a la vez, asumiendo la función de un lock normal.

14. Programa `omp_mat_vect.c`





Se hizo la prueba para una matriz $10 * 20$ con cinco y diez hilos. La solución con diez hilos parece ser más rápida al momento de hacer el cálculo.