Diego Lemus - 21469

Hoja de Trabajo 1

Computación Paralela y Distribuida

Repositorio

Acceso: https://github.com/TheDeloz-v2/CPD-HT1

Ejercicio 1

```
root@b1c3eea0fe7c:/usr/src/app# ./hello_omp
Hello from thread 8 of 10!
Hello from thread 1 of 10!
Hello from thread 7 of 10!
Hello from thread 4 of 10!
Hello from thread 9 of 10!
Hello from thread 5 of 10!
Hello from thread 3 of 10!
Hello from thread 2 of 10!
Hello from thread 0 of 10!
Hello from thread 6 of 10!
root@b1c3eea0fe7c:/usr/src/app# ./hello_omp 4
Hello from thread 0 of 4!
Hello from thread 2 of 4!
Hello from thread 1 of 4!
Hello from thread 3 of 4!
root@b1c3eea0fe7c:/usr/src/app#
```

¿Por qué al ejecutar su código los mensajes no están desplegados en orden?

Porque cada thread se ejecuta de manera concurrente e independiente. OpenMP divide el trabajo entre varios threads que pueden ejecutarse al mismo tiempo, lo que significa que no hay garantía de que los threads completen su tarea en un orden predecible.

Ejercicio 2

```
root@b1c3eea0fe7c:/usr/src/app# ./hbd_omp 14
Saludos del hilo 2
Feliz cumpleaños número 14!
Feliz cumpleaños número 14!
Feliz cumpleaños número 14!
Saludos del hilo 4
Saludos del hilo 6
Feliz cumpleaños número 14!
Feliz cumpleaños número 14!
Saludos del hilo 10
Saludos del hilo 8
Feliz cumpleaños número 14!
Saludos del hilo 12
Feliz cumpleaños número 14!
Saludos del hilo 0
root@b1c3eea0fe7c:/usr/src/app#
```

Ejercicio 3

```
root@b1c3eea0fe7c:/usr/src/app# g++ -o riemann1 riemann.c
root@b1c3eea0fe7c:/usr/src/app# ./riemann1 2 10
Con n = 1.000000e+07
En el intervalo [2.000000, 10.000000]
La aproximacion es = 330.666667
root@b1c3eea0fe7c:/usr/src/app#
```

```
root@b1c3eea0fe7c:/usr/src/app# g++ -o riemann2 riemann.c
root@b1c3eea0fe7c:/usr/src/app# ./riemann2 3 7
Con n = 1.000000e+07
En el intervalo [3.000000, 7.000000]
La aproximacion es = 1160.000000
root@b1c3eea0fe7c:/usr/src/app#
```

```
root@b1c3eea0fe7c:/usr/src/app# g++ -o riemann3 riemann.c
root@b1c3eea0fe7c:/usr/src/app# ./riemann3 0 1
Con n = 1.000000e+07
En el intervalo [0.000000, 1.000000]
La aproximacion es = 0.459698
root@b1c3eea0fe7c:/usr/src/app#
```

Ejercicio 4

```
root@b1c3eea0fe7c:/usr/src/app# ./riemann_omp2-1 2 10 4
Thread 2: [6.000000, 8.000000]
Thread 0: [2.000000, 4.000000]
Thread 1: [4.000000, 6.000000]
Thread 3: [8.000000, 10.000000]
Con n = 1000000 y 4 threads
En el intervalo [2.000000, 10.000000]
La aproximacion es 330.666667
root@b1c3eea0fe7c:/usr/src/app#
```

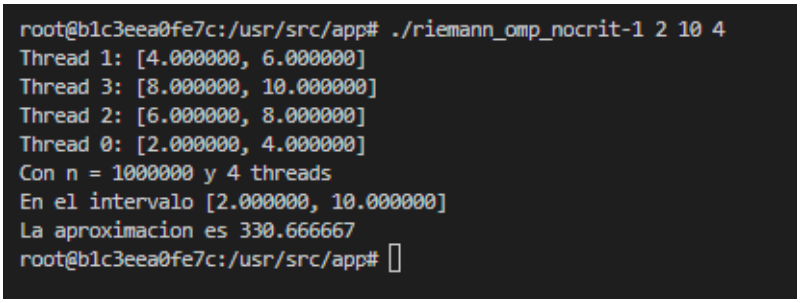
```
root@b1c3eea0fe7c:/usr/src/app# ./riemann_omp2-2 3 7 4
Thread 3: [6.000000, 7.000000]
Thread 0: [3.000000, 4.000000]
Thread 1: [4.000000, 5.000000]
Thread 2: [5.000000, 6.000000]
Con n = 1000000 y 4 threads
En el intervalo [3.000000, 7.000000]
La aproximacion es 1160.000000
root@b1c3eea0fe7c:/usr/src/app#
```

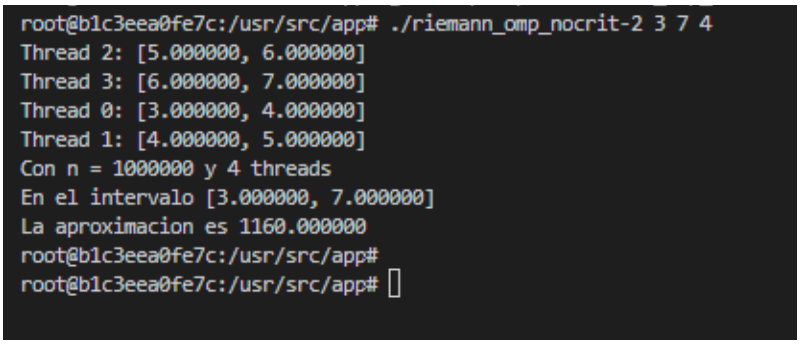
```
root@b1c3eea0fe7c:/usr/src/app# ./riemann_omp2-2 0 1 6
Thread 5: [0.833330, 0.999996]
Thread 2: [0.333332, 0.499998]
Thread 1: [0.166666, 0.333332]
Thread 4: [0.666664, 0.833330]
Thread 0: [0.000000, 0.166666]
Thread 3: [0.499998, 0.666664]
Con n = 1000000 y 6 threads
En el intervalo [0.000000, 1.000000]
La aproximacion es 0.499992
root@b1c3eea0fe7c:/usr/src/app#
```

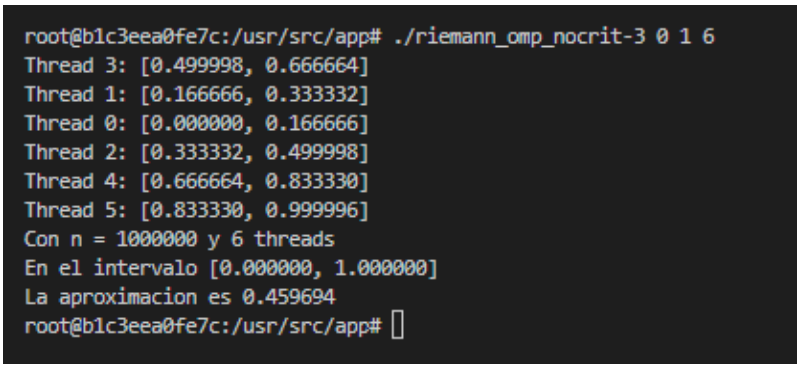
¿Por qué es necesario el uso de la directiva #pragma omp critical?

Se utiliza para proteger una sección crítica del código, es decir, una sección donde múltiples threads podrían acceder y modificar recursos compartidos simultáneamente, lo que podría llevar a resultados incorrectos o indeterminados; en este caso, por el manejo de la variable de sum que se lee antes de realizar la operación.

Ejercicio 5







¿Por qué es necesario el uso de la directiva #pragma omp critical?

Al usar un arreglo global en lugar de una única variable global protegida por una sección crítica, se evita el problema de las race condition de una manera diferente. En lugar de proteger la escritura a la variable sum, cada thread escribe su resultado local en un índice único de un arreglo global. De esta forma, no hay necesidad de sincronización adicional, ya que cada thread tiene su propia posición en el arreglo, y no existe el riesgo de que dos threads intenten escribir en la misma posición al mismo tiempo.