Práctica 5

**Optimización y Sincronización en OpenMP**

Jordi Blasco Lozano

Computación de alto rendimiento

Grado en Inteligencia Artificial

## Indice:

[Indice: 2](#_Toc190980674)

[1. INSTALACIÓN Y CONFIGURACIÓN 2](#_Toc190980675)

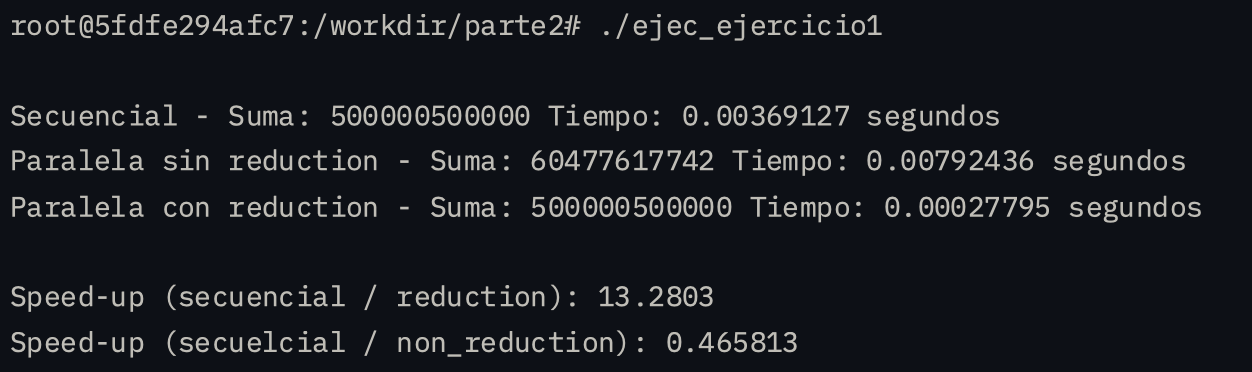
[2. PROGRAMACIÓN CON OPENMP 3](#_Toc190980676)

[CÓDIGO COMENTADO EJERCICIO FINAL 6](#_Toc190980677)

## Desarrollo práctico guiado

## Ejercicio 1:

El objetivo de este ejercicio será comprender como podemos aplicar la cláusula ‘reduction’, que función desempeña en nuestro código y que ventajas obtenemos.

Nuestro código ejecuta 3 bucles con 1000000 iteraciones, 1 de los bucles se ejecuta en modo secuencial, otro de los bucles se ejecuta en paralelo sin hacer uso de la cláusula ‘reduction’ y el ultimo bucle se ejecuta en paralelo también, pero haciendo uso de esta cláusula. Dentro de cada bucle se hace una suma de +i por cada iteración, el tiempo que tarda en ejecutarse cada tipo de bucle se guarda también. Al final calculamos el speed-up entre el secuencial y los bucles paralelos. He obtenido estos resultados:  


Como podemos observar la versión paralela sin reduction nos devuelve una suma invalida, esto se debe a las condiciones de carrera. Las condiciones de carrera ocurren cuando varios hilos intentan modificar una variable compartida de manera desincronizada.

Para paliar esta situación usamos la cláusula ‘reduction’. En nuestro código, en la versión paralela sin reduction, la variable 'suma' es compartida por todos los hilos sin sincronización esto provoca condiciones de carrera y un resultado final erróneo. Por otra parte, con reduction cada hilo mantiene una copia local de la variable 'suma' y al finalizar se combinan, garantizando un resultado correcto sin la sobreescritura simultánea.

En nuestra ejecución hemos obtenido que el bucle paralelo con reduction se ejecuta 13.28 veces más rápido que el bucle secuencial, esto supone una gran mejora de rendimiento. Al tener 16 hilos esta mejora es bastante considerable.