Arquitectura de Computadores

Grado de Ingeniería Informática 2018-19

Práctica de programación paralela con OpenMP

Grupo 3

CARLOS GARCIA CAÑIBANO	100363813
JUAN JOSE GARZON LUJAN	100363861
ZHENFENG ZHU	100363798

Índice de contenidos

1. Introducción

Este documento explica detalladamente la solución que hemos desarrollado para el problema que nos presenta esta práctica.

En primer lugar hablaremos de nuestras implementaciones y optimizaciones realizadas de la versión secuencial y de la versión paralela para resolver el problema propuesto. Luego, incluiremos una gráfica de distintas evaluaciones comparando con el programa base inicial, la versión secuencial y la versión paralela, junto con una explicación convincente para los resultados evaluados. Después incluiremos las pruebas realizadas que aseguran la correcta ejecución de las dos versiones. Por último concluimos con nuestras opiniones personales sobre esta práctica.

Para simplificar el problema hemos decidido desarrollar la solución en dos ficheros distintos para las dos versiones de la práctica. El fichero *nasteroids-xxx.cpp*, donde *xxx* podría ser *seq* o *par*, es el fichero que se encarga de implementar el problema propuesto, mientras el fichero *math_functions.cpp* se encarga de implementar las funciones matemáticas que necesitamos para resolver el problema para ambas versiones.

Por tanto, para la parte secuencial tenemos los ficheros *nasteroids-seq.cpp* y *math_functions.cpp*, mientras para la parte paralela tenemos los ficheros *nasteroids-par.cpp* y *math_functions.cpp*.

2. Versión secuencial

a. Implementación

El problema a resolver consiste en una simulación del movimiento de un conjunto de asteroides en intervalos de tiempo de longitud Δt, teniendo en cuenta la atracción gravitatoria que producen unos sobre otros.

Para la resolución del problema, hemos seguido el siguiente orden:

En primer lugar, procedemos a comprobar los parámetros de entrada. En el caso de que alguno de los parámetros no sean correctos o no estén presentes se procederá a terminar el programa con código de error -1 y un mensaje de error como indica en el enunciado.

Una vez comprobado los parámetros de entrada, procedemos a generar las masas y las coordenadas en los ejes x e y para los asteroides y las planetas, definidos como vectores de clases, mediante la función de generación de parámetros de simulación que nos facilitó el enunciado.

La clase que hemos definida para los asteroides y planetas se llama *SpaceObject*, y que contiene los siguientes parámetros:

- Coordenada x
- Coordenada y
- Velocidad en eje x
- Velocidad en eje y
- Velocidad en eje x de la iteración anterior
- Velocidad en eje y de la iteración anterior
- Masa
- Número de colisiones con otros asteroides

Después de generar los parámetros iniciales, procedemos a generar un fichero de salida llamado *init_conf.txt* que contiene la configuración inicial del programa.

Una vez generado el fichero de salida, procedemos a implementar el movimiento de los asteroides. Para ello, implementamos un bucle que repite *n* veces, siendo *n* el número de iteraciones introducido en los parámetros de entrada.

Dentro del bucle de iteraciones, calculamos primero las fuerzas que recibe un asteroide frente a todos los demás mediante la fórmula proporcionada en el enunciado, y guardamos esas fuerzas de cada eje en dos vectores distintos, en el índice que corresponde el asteroide. El asteroide objetivo se le guarda las mismas fuerzas, pero cambiando de signo. Después haremos lo mismo con el siguiente asteroide hasta el final.

Tras conseguir las fuerzas de cada asteroide con los demás, continuamos a calcular fuerzas para cada asteroide con las planetas, y sumar esas fuerzas calculadas en los dos vectores de fuerza en el índice correspondiente para cada asteroide.

Después de obtener las fuerzas finales de la iteración para todos los asteroides, pasamos cada asteroide (clase de asteroide) con sus vectores de fuerzas a la función para actualizar sus coordenadas y velocidades.

Luego procedemos a evaluar si la nueva posición de un asteroide choca con el borde. En caso afirmativo se le asigna una posición que equidista a 2 unidades del borde y se le cambia de signo a la velocidad en sentido al borde.

Una vez cambiadas las coordenadas para los asteroides, comprobamos si hay colisiones entre ellos. En caso de que haya colisión, si la distancia entre dos asteroides es menor o igual que 2, sus velocidades pasan a ser las velocidades anteriores de la actualización, y esas velocidades se intercambian entre los dos asteroides.

Luego se repite el bucle de iteraciones.

Por último, generamos un fichero de salida llamado out.txt que contiene los resultados finales de coordenadas, velocidades y masa para cada asteroide.

3. Versión paralela

a. Implementación

b. Optimización

En términos relativos a la **implementación** de la versión paralela del código, hemos priorizado acerca de la cercanía entre ambas codificaciones y el aprovechamiento del máximo rendimiento en aquellas zonas donde la carga de trabajo del programa es mucho más elevada. Ahora pasaremos a describir las diferentes zonas del código para clarificar el por qué de su paralelización (también en caso contrario):

- La **creación de asteroides y planetas** no ha sido paralelizada porque el motor de generación de números aleatorios proporcionado es totalmente dependiente del orden de ejecución para obtener los valores. No tendría sentido que para ejecuciones diferentes con la misma semilla, los datos iniciales fuesen diferentes.

- El primer **bucle de control de las iteraciones** no ha sido paralelizado puesto que los cálculos sucesivos llevados a cabo en las iteraciones posteriores dependen

totalmente del resultado de sus anteriores. Las iteraciones deben ser ejecutadas de manera ordenada para no afectar negativamente al resultado final.

- En el caso del bucle anidado que se encarga del **cálculo de las fuerzas entre todos los asteroides** tuvimos que utilizar una estructura alternativa a los vectores unidimensionales para poder respetar el orden de suma y evitar las carreras de datos. Por este motivo decidimos usar una matriz bidimensional que exclusivamente manejase una sola posición, dentro de sí misma, que representase la fuerza de un asteroide respecto de otro de forma unívoca. De esta manera se soluciona el problema y, aunque la estructura de datos sea más pesada que la original, se puede paralelizar completamente sin el uso de secciones críticas u otros mecanismos. Esta zona es importante puesto que representa gran parte de la carga de trabajo del programa y por este motivo hemos estimado oportuno refactorizar el código para intentar conseguir un buen rendimiento. (Después todos los resultados son sumados en otro bucle paralelo adicional…)

- Para el **cálculo de fuerzas entre asteroides y planetas** el proceso de paralelización ha sido similar al del punto anteriormente descrito. Son las mismas consideraciones y la misma solución.

- Los bucles de **cambio de posición, manejo de choques en los bordes y recolocación en caso de choque** han sido totalmente paralelizados. El orden de manipulación de las propiedades de los asteroides no es relevante puesto que las nuevas posiciones o velocidades (en el caso exclusivo de los bordes…) son independientes unas de otras.

- El bucle de **choque entre asteroides** ha representado un problema a la hora de ser paralelizado puesto que el intercambio de las velocidades entre asteroides tiene en cuenta el orden de aplicación. Por este motivo, ha preservado el comportamiento original. La evaluación de los choques entre pares de asteroides ha impedido hacerlo.

- El último **bucle de recolocación tras choque** presenta las mismas consideraciones que su homólogo ejecutado antes de las colisiones. Al ser una operación independiente entre cuerpos, la paralelización resulta trivial.

Una vez descritos los detalles de la implementación y aquellas variaciones en términos de paralelización que hemos considerado importantes, explicaremos brevemente **algunas optimizaciones básicas** que hemos realizado para intentar mejorar el rendimiento del programa.

- Para paralelizar los bucles anidados hemos utilizado la cláusula *collapse*. Por lo que hemos podido entender y aplicar, fusiona ambas iteraciones anidadas de los bucles y gestiona mucho mejor la repartición de cómputo entre hilos.

- En los cálculos de fuerzas entre todos los distintos cuerpos, hemos utilizado la matriz anteriormente descrita y un bucle adicional de suma total. Aunque para una versión secuencial del código el rendimiento sería inferior, consideramos que para el caso paralelo es una optimización, puesto que no utilizamos secciones críticas u otro tipo de mecanismos cuyo uso impactaría negativamente en el rendimiento del programa.

4. Evaluación de rendimiento

Los parámetros relevantes de la máquina en la que hemos procedido a la ejecución:

- Arquitectura: x86_64
- Modos de operación de las CPU: 32-bit, 64-bit
- Orden de los bytes: Little Endian
- Modelo del procesador: Intel Core i5-4460 3.20 GHz
- Número de cores: 4
- Tamaño de memoria principal: 16334776 KiB
- Jerarquía de la memoria caché:
    - L1: 32K
    - L2: 256K
    - L3: 6144K

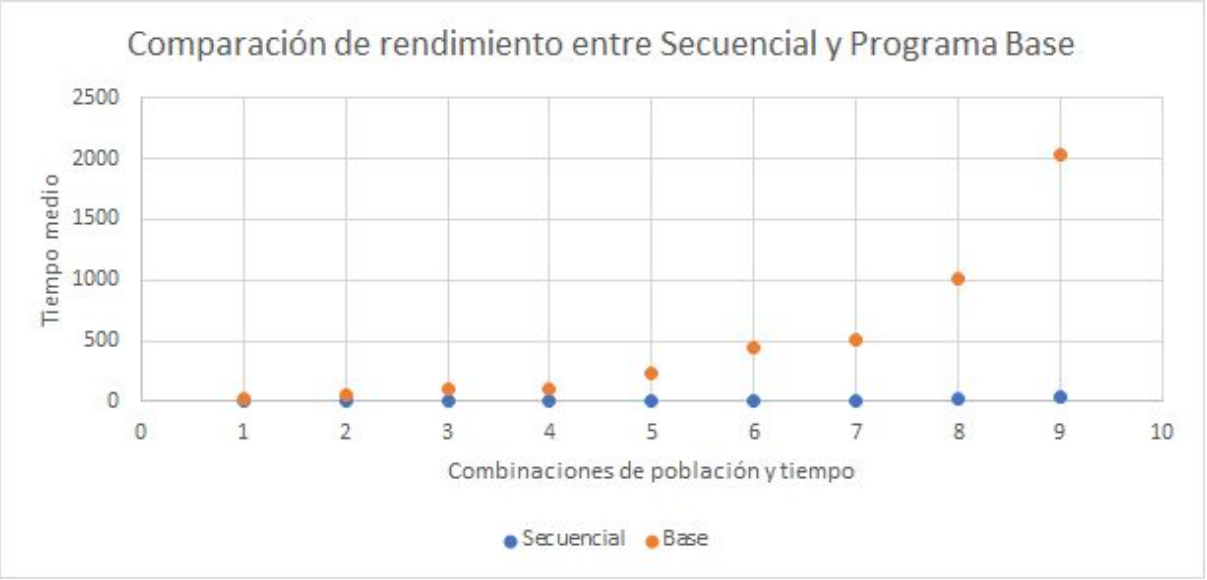
Software de sistema

- Versión de S.O. : Debian GNU/Linux 9 (stretch)
- Versión del compilador: Debian 6.3.0-18+deb9u1 (6.3.0 20170516)

Tabla de tiempo medio de ejecución

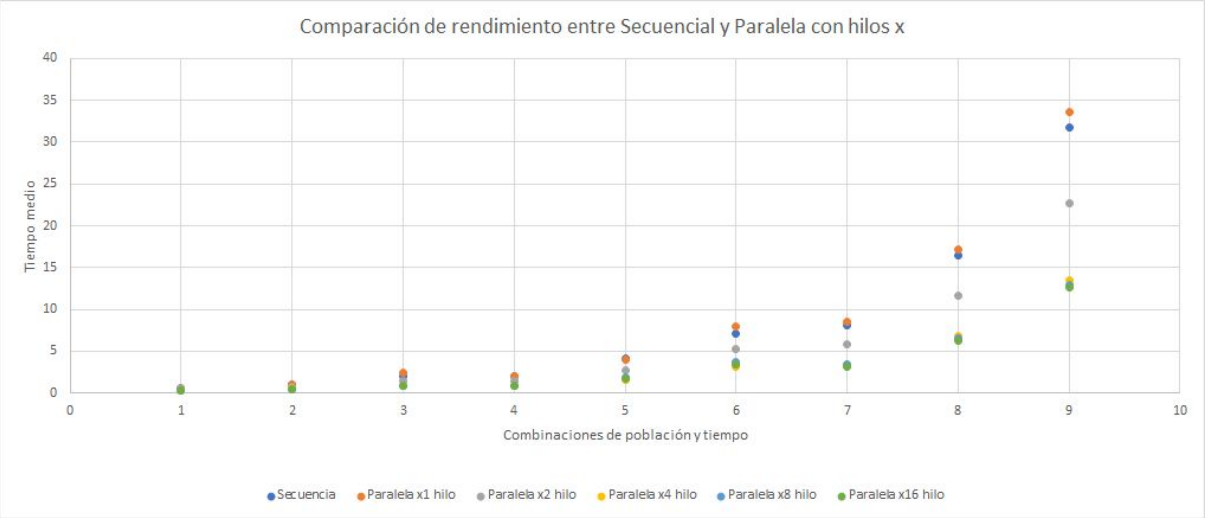
Combinaciones de población y tiempo	Secuencial	Paralela x1 hilo	Paralela x2 hilo	Paralela x4 hilo	Paralela x8 hilo	Paralela x16 hilo	Base
1. 250 50 250 10	0.5303628	0.54536	0.55029	0.26988	0.27045	0.27206	26.831
2. 250 100 250 10	1.0231705	1.02181	0.75975	0.52055	0.47607	0.4519	59.403
3. 250 200 250 10	2.0093262	2.40199	1.40168	0.82651	0.8912	0.88212	97.733
4. 500 50 500 10	2.0512038	1.99945	1.37909	0.87093	0.93514	0.90222	110.732
5. 500 100 500 10	4.0736755	3.93318	2.76188	1.60026	1.83034	1.72825	225.511s
6. 500 200 500 10	7.125310453	7.91492	5.32036	3.14629	3.6869	3.42857	445.087
7. 1000 50 1000 10	8.077210576	8.52104	5.75956	3.4093	3.35782	3.18867	510.748
8. 1000 100 1000 10	16.4461687	17.21101	11.59135	6.87468	6.59714	6.32471	1018.236
9. 1000 200 1000 10	31.75333	33.55808	22.76888	13.44794	12.89266	12.59868	2035.324

Las combinaciones de población y tiempo están formados por 9 casos, tal como indica en la primera fila. Los primeros 3 casos son para 250 asteroides y planetas, los 3 siguientes son para 500 asteroides y planetas, y los 3 últimos son para 1000 asteroides y planetas. Para cada uno de esos tri-casos, el número de iteraciones varían de 50, 100 y 200.

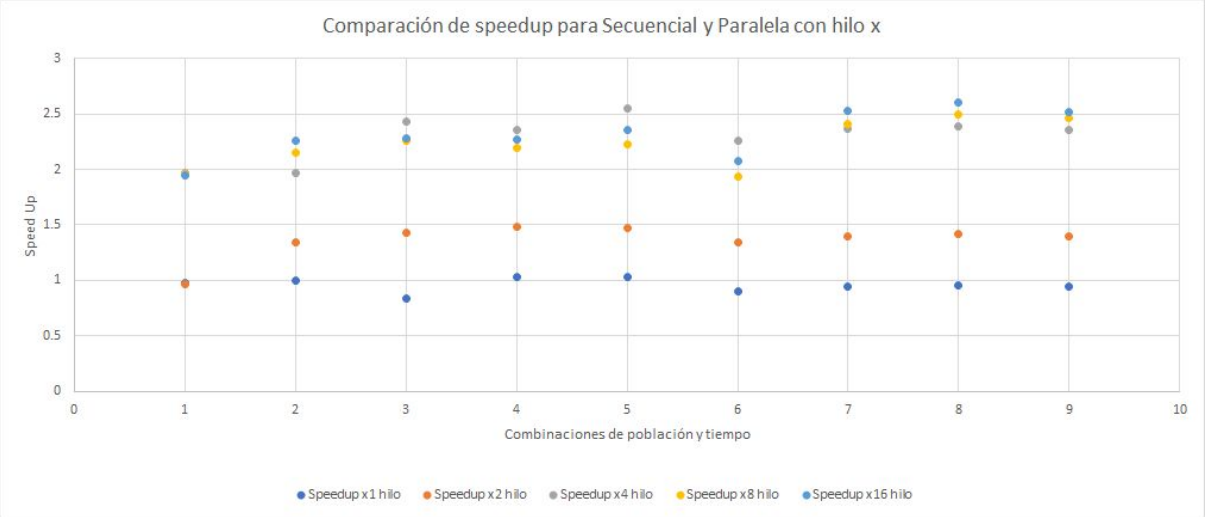


En esta gráfica observamos que el tiempo medio de ejecución de nuestra versión secuencial es menor que el tiempo del programa Base, y a medida que aumentamos el número de asteroides, planetas y el número de iteraciones esa diferencia se hace cada vez mayor. Aunque también hay que tener en cuenta que el programa Base imprime un fichero adicional de fuerzas para cada iteración.

En esta gráfica nos muestra el speed up entre unos 50 y 60.

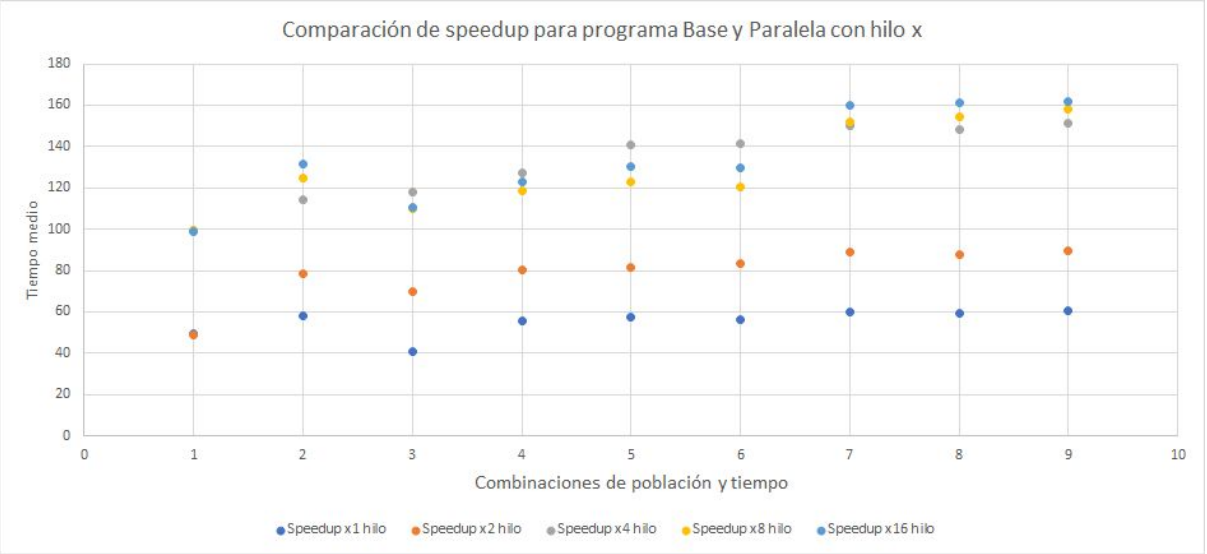


En esta gráfica de comparación de rendimiento entre Secuencial y Paralela con hilos x, observamos que la versión secuencial y la versión paralela con 1 hilo tienen casi los mismos tiempos de ejecución, ya que ambas versiones ejecutan secuencialmente. Mientras para la versión paralela con 4, 8 y 16 hilos tienen casi el mismo tiempo de ejecución porque el procesador tiene 4 núcleos, por tanto puede ejecutar paralelamente 4 hilos.



En esta gráfica comparamos el speed up para la versión secuencial con la versión paralela para los hilos 1,2,4,8 y 16. Mediante la gráfica observamos que el speed up es casi idéntica a 1 para el caso de un hilo, porque en ese caso la ejecución es totalmente secuencial. Para los hilos de 2 y 4, empiezan a tener subidas de speed up, porque el programa aprovecha los hilos para la ejecución en paralela. Sin embargo, para los hilos 4,8 y 16 no ha habido muchas diferencias en speed up. Eso es debido a que el procesador de la máquina tiene

solamente 4 núcleos, por lo que puede lanzar como máximo 4 hilos para la ejecución en paralela. Más de 4 hilos no conseguirá mejoras de speed up.



En cuanto al speedup para el programa Base y nuestra versión paralela con hilos x, nos muestra desde unos 40 hasta unos 160.

5. Pruebas realizadas

Clase utilizada	Entrada	Resultado
nasteroids-seq.cpp	**3 3 3 10**	No hay cambios respecto a de los profes. Correcto funcionamiento. No ha habido choques entre asteroides ni con borde.
nasteroids-seq.cpp	**-2 1 0 0**	nasteroids-seq: Wrong arguments. Correct use: nasteroid-seq num_asteroids num_iterations num_planets seed Al introducir mal los parámetros salta el error.
nasteroids-seq.cpp	**0 0 0 0**	No hay cambios respecto a de los profes. Correcto funcionamiento. No ha habido choques entre asteroides ni con borde.

nasteroids-seq.cpp	**100 50 100 100**	No hay cambios respecto a de los profes. Correcto funcionamiento. No ha habido choques entre asteroides ni con borde.
nasteroids-seq.cpp	**150 50 100 100**	Hay **2** líneas diferentes respecto al base. Asteroide 20: **Base:** 198.519 198.027 0.945 -0.055 932.255 **Secuencial:** 198.519 198.027 0.949 -0.045 932.255 **Asteroide 124:** **Base:** 199.573 197.073 0.949 -0.045 1029.227 **Secuencial:** 199.573 197.073 0.945 -0.055 1029.227 Ha habido **147** choques entre asteroides y ninguno con bordes.
nasteroids-seq.cpp	**250 50 250 10**	Hay **17** líneas diferentes respecto al base. **Asteroide 15:** **Base:** 177.807 12.73 40.260 0.034 994.017 **Secuencial**: 177.807 12.734 0.260 0.033 994.017 Todos los errores son en el orden de la 3 cifra decimal. Hay **4** choques con borde y **396** entre asteroides**.**
nasteroids-seq.cpp	**500 100 250 100**	Hay **214** líneas diferentes. **Asteroide 2:** **Base:** 97.285 194.241 1.185 0.079 997.580 **Secuencial: 9**7.284 194.240 1.185 0.079 997.580 **Asteroide 20:** **Base:** 196.130 198.391 -0.467 -0.092 932.255 **Secuencial:** 211.427 197.423 4.597 -0.191 932.255 Hay **3995** choques entre asteroides y **191**

		con bordes. La mayoría de los errores son en la tercera cifra decimal. Habiendo excepciones como en el asteroide 20.
nasteroids-seq.cpp	**750 200 250 50**	Hay **742** líneas diferentes. **Asteroide 1:** **Base:** 127.359 187.149 1.227 0.061 994.219 **Secuencial:** 127.359 187.148 1.227 0.061 994.219 **Base:** 183.003 119.769 0.809 -0.091 999.829 **Secuencial:** 182.870 119.780 0.778 -0.092 999.829 Hay **21584** choques entre asteroides y **2397** con bordes. Hay muchos errores de los tipos de solo en la última cifra decimal o en décimas.

Se puede ver que funciona en todos los casos pero no de la misma manera que el base. Se aprecia que cuando no hay ningún tipo de rebote sale idéntico al base pero a medida que crece la población y las iteraciones empieza a haber diferencias. Creemos que es debido a la hora de gestionar bien los choques y los rebotes con los bordes se puede ver que al principio, sale un error muy pequeño en la tercera cifra decimal pero ,al final ese error se propaga y se hace más grande hasta en décimas como se ve en la última prueba. El error obtenido en la penúltima prueba que sale muy cambiado nos parece ser de que el asteroide al chocar con borde y chocar con asteroide, el orden de los cambios no está igual ejecutado respecto al base y ahí la diferencia.

Clase utilizada	Entrada	Resultado
nasteroids-par.cpp	**5 5 98**	Correcto funcionamiento, mismo resultado que el secuencial
		nasteroids-seq: Wrong arguments. Correct use:

nasteroids-par.cpp	**-2 1 0 0**	nasteroid-seq num_asteroids num_iterations num_planets seed Al introducir mal los parámetros salta el error.
nasteroids-par.cpp	**0 0 0 0**	Correcto funcionamiento, mismo resultado que el secuencial
nasteroids-par.cpp	**100 40 50 100**	Correcto funcionamiento, mismo resultado que el secuencial
nasteroids-par.cpp	**300 100 200 60**	Correcto funcionamiento, mismo resultado que el secuencial
nasteroids-par.cpp	**600 150 400 58**	Correcto funcionamiento, mismo resultado que el secuencial
nasteroids-par.cpp	**400 200 600 67**	Correcto funcionamiento, mismo resultado que el secuencial

Funciona en todos los escenarios y siempre igual que la secuencial , eso significa que la versión paralela está bien hecha. Esta versión paralela tendrá los mismos fallos respecto al base que tiene el secuencial debido que es igual a él.

6. Conclusiones

Después de haber terminado la realización de esta práctica, podemos concluir que se nota una gran mejora de tiempo de la versión paralela a la secuencial a medida que el número de hilos crece y las operaciones a realizar es mayor. Las principales dificultades que nos hemos encontrado a la hora de trabajar ha sido la inexperiencia al trabajar con openmp y no saber bien que se podía paralelizar y cómo se debía paralelizar. El principal problema era evitar la carrera de datos que hacía que la versión paralela saliera a la secuencial , para ello tuvimos que rediseñar nuestro código y añadir pequeños elementos. Finalmente, después de muchas horas de trabajo creemos que hemos conseguido el objetivo principal de la práctica que era aprender utilizar openmp.