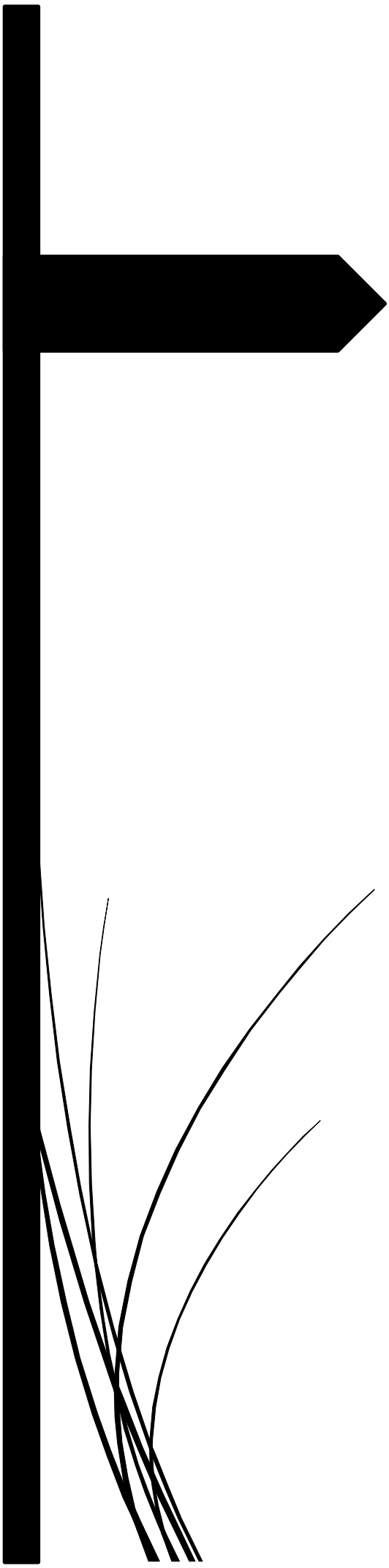


Practica de programación paralela con OpenMP

Arquitectura de computadores

Gonzalo Fernández
Daniel Romero Ureña
Alejandro Díaz García
Francisco José Rueda Zaragoza

ÍNDICE

INTRODUCCIÓN

En este documento se detalla el proceso de desarrollo de la práctica, así como su implementación en el lenguaje C++ y la comprobación de su correcto funcionamiento mediante casos de prueba.

La memoria se divide en tres grandes bloques:

- Versión secuencial del código: en esta sección se justificará el diseño del problema planteado por la práctica, así como su implementación de forma secuencial en el lenguaje C++ y sus casos de prueba para verificar el correcto funcionamiento del código.
- Versión secuencial del código: en este apartado, se justificarán las diferentes decisiones de optimización tomadas en la paralelización del código en su versión secuencial haciendo uso de la herramienta Open MP. Además, se analizarán las diferentes ventajas obtenidas gracias a la paralelización del código.
- Conclusiones sobre la práctica: en este bloque se realizará una breve conclusión y valoración sobre la práctica realizada.

VERSIÓN SECUENCIAL DEL CÓDIGO

Introducción

El problema que se plantea trata sobre una simulación donde hay dos tipos de objetos: planetas y asteroides, estos están situados en una especie de tablero el cual tiene una altura y anchura de 200. Los asteroides tendrán la capacidad de moverse en todas direcciones, pudiendo a llegar a colisionar entre sí, con un planeta o con los límites del espacio definidos; mientras que los planetas serán objetos fijos que estarán situados en cualquier lugar de las cuatro fronteras que delimitan el tablero.

Todos estos objetos tendrán un total de 4 atributos: su posición representada en los ejes X e Y, su masa y su velocidad actual, este último únicamente estará presente en los asteroides. Los atributos masa y posición se generarán de manera aleatoria, mientras que la velocidad se obtendrá mediante unas fórmulas proporcionadas en el enunciado de la práctica.

Explicación del código

El código se estructura principalmente en cuatro grandes secciones: clase planeta, clase asteroide, funciones y main. Todo esto sin contar la cabecera del código, la cual alberga las diferentes librerías utilizadas, así como las constantes que utilizaremos en las diferentes fórmulas.

Clase planeta

En esta parte del código se define el objeto planeta, el cual está compuesto de un total de tres atributos, los cuales son: su posición en los ejes cartesianos y su masa. Todos estos

atributos son de tipo doble y serán generados de forma aleatoria mediante el método default_random_engine.

Clase asteroide

En este fragmento de código, se define el objeto asteroide, que al igual que el objeto planeta cuenta con los atributos posición y masa, los cuales también de forma aleatoria mediante el método default_random_engine. Sin embargo, en este caso debido a su movimiento, los asteroides tendrán un atributo velocidad para el eje X y el eje Y.

Funciones

En esta sección del código se encuentran las diferentes funciones que se usarán a lo largo del código, siendo las siguientes:

- Distancia planeta-asteroide: Esta función es la encargada de calcular la distancia entre un planeta y un determinado asteroide, aplicando la fórmula especificada en el enunciado de la práctica, en este caso se ha prescindido del uso de la raíz cuadrada de la fórmula, debido a la simplificación en su uso en el cálculo del resto de fórmulas que hacen uso de la misma. Esta función tendrá como parámetro de entrada un objeto planeta y un objeto asteroide.
- Distancia asteroide-asteroide: Esta función es una versión alternativa de la función anterior, pero en este caso se tendrá como parámetro de entrada dos objetos asteroide.
- Ángulo planeta-asteroide: Su objetivo es el de calcular la pendiente entre un objeto planeta y un objeto asteroide, teniendo a estos como parámetros de entrada, para ello se implementará la fórmula de cálculo de pendiente proporcionada. Dependiendo del resultado obtenido mediante la fórmula se asignará a la pendiente un valor de 1, si el resultado obtenido ha sido mayor a 1, y una pendiente de -1 si el resultado obtenido ha sido menor que 1. Tras esto, se obtendrá el ángulo realizando la arcotangente de la pendiente obtenida.
- Ángulo planeta-asteroide: esta función tiene el mismo objetivo y funcionamiento que la función ángulo planeta-asteroide, cambiando únicamente los parámetros de entrada, los cuales son dos objetos asteroides.
- Fuerza x planeta-asteroide: es la encargada de calcular la fuerza gravitatoria existente entre un objeto planeta y un objeto asteroide en el eje X, para la obtención de dicho valor se hace uso de la fórmula especificada en el enunciado y de las funciones distancia planeta-asteroide y ángulo planeta-asteroide, explicadas anteriormente, para calcular la distancia y el ángulo de estos dos objetos, haciendo uso de estos valores en la fórmula que calcula la fuerza.
- Fuerza y planeta-asteroide: esta función tiene un objetivo y funcionamiento similar a la anterior, únicamente cambiando el hecho de que la fuerza calculada corresponde al eje Y en lugar del eje X, por lo que se hará uso del seno del ángulo en lugar del coseno utilizado para el eje X en la fórmula de la fuerza.

- Fuerza x asteroide-asteroide: su objetivo y funcionamiento es similar al de la función fuerza x planeta-asteroide, teniendo como únicamente los parámetros de entrada, los cuales pasan a ser dos objetos asteroide.
- Fuerza y asteroide-asteroide: esta función tiene el mismo objetivo y funcionamiento que la función fuerza x asteroide-asteroide, teniendo como única diferencia que esta fuerza será la fuerza en el eje Y en lugar del eje X, por lo que se utilizará el seno del ángulo en lugar del coseno en la fórmula de la fuerza.
- Actualizar: tiene el cometido de actualizar la posición de un determinado asteroide en función de la fuerza que tiene en los diferentes ejes de coordenadas, para ello recibirá como parámetros de entrada el objeto asteroide y sus fuerzas en el eje X e Y. Para obtener la nueva posición, se obtendrá la aceleración actual del asteroide en los ejes de coordenadas, para posteriormente hacer uso de esta en el cálculo de la velocidad en el eje X e Y.
  Finalmente se calculará la nueva posición mediante la fórmula del movimiento rectilíneo uniformemente acelerado, la cual hace uso de los valores de la aceleración y la velocidad previamente obtenidos.

Main

Al comienzo de este bloque de código se encuentran programados los diferentes casos de error posibles en la introducción de valores por pantalla, entre los cuales se encuentra el uso de valores negativos y el uso de un número de valores distinto a 5, entre otros.
A continuación, inicializamos los valores introducidos por pantalla, siendo el primer valor el número de planetas, el segundo el número de iteraciones, el tercero el número de planetas y el cuarto la semilla.

Seguidamente se encuentra la función generadores de números aleatorios especificada en el enunciado, teniendo una manera de generar los números aleatorios para las posiciones de los diferentes ejes de coordenadas y la masa de cada objeto. Bajo esta sección de código se encuentran un total de tres bucles for, siendo el primero el que asigna a los diferentes asteroides sus posiciones y masa haciendo uso de las funciones aleatorios previamente establecidas, además de ir añadiendo los asteroides dentro del vector. El segundo bucle for se encargará de crear los diferentes planetas junto con sus posiciones y masas, añadiendo estos al vector planetas, además de esto, el bucle generará cuatro planetas por cada iteración del bucle, de este modo se reducirá el número de iteraciones y mejorará el rendimiento, esto es debido a que por cada interacción generamos los planetas de cada pared del mapa. Por último, en el tercer bucle se acedera en los casos que el número de planetas no sea múltiplo de cuatro y por tanto con se deben generar 1, 2 o 3 planetas más que se suman a los generados en el bucle anterior.

Tras esta sección de código se sitúan los mensajes que saldrán por pantalla tras la ejecución del programa, mostrando los principales datos del problema, como el número de cuerpos, los

valores de las constantes utilizadas y los archivos de salida tras la ejecución. Seguidamente se genera el fichero init_conf.txt el cual contendrá la configuración inicial del problema.

La siguiente sección del main tiene como objetivo el correcto desarrollo de las interacciones del problema haciendo uso de un total de 4 bucles for, tres de ellos en el interior del bucle principal destinado a la sucesión de interacciones, para estas interacciones se hará uso de un vector copia de asteroides, el cual almacenará los valores de los diferentes asteroides al final de la iteración anterior.

- Primer bucle for: este primer bucle tiene como objetivo almacenar el resto de bucles y encargarse de que estos se ejecuten tantas veces como iteraciones se han introducido en el problema.
- Segundo bucle for: tendrá un número de iteraciones igual al número de asteroides del problema y contendrá a los dos bucles for restantes, además de inicializar la variable de fuerza x y fuerza y en cada una de sus iteraciones, actualiza la posición y velocidad de cada uno de los asteroides en el vector original y comprueba sus respectivos rebotes con los límites del mapa.
- Tercer bucle for: este se encuentra introducido en el segundo bucle y será el encargado de recorrer todo el vector asteroides y comprobar para un determinado asteroide la distancia que tiene con el resto, obteniendo de este modo la fuerza total que estos ejercen sobre él. Además de esto, se comprobará si la distancia de los asteroides es inferior a 5, provocando una colisión entre ambos.
- Cuarto bucle for: este bucle se sitúa a la altura del tercero y tiene como principal objetivo el obtener la fuerza total en los ejes de coordenadas de un determinado asteroide, esto lo hace sumando el valor de las fuerzas obtenidas en el anterior bucle con la fuerza que ejerce cada uno de los planetas con el asteroide en cuestión.

Finalmente se generará el fichero de salida out.txt haciendo uso de un bucle for, el cual imprimirá el estado final de todos y cada uno de los objetos planetas y asteroides junto con los valores finales de sus atributos.

Conclusión

Tras la implementación del código secuencial y las pruebas realizadas, podemos concluir que las soluciones esperadas se acercan a las nuestras variando en el valor decimal excepto en los casos más extremos de poblaciones de 1000 y 200 iteraciones, donde la diferencia en la posición X en casos puntuales supera en más de 5 la solución esperada. Creemos que esto puede ser debido a fallos en la gestión de colisión entre asteroides.

VERSIÓN DEL CÓDIGO CON PARALELISMO

Introducción

En esta versión del código se procede a paralelizar fragmentos del mismo mediante la herramienta Open MP, con la cual se aumentará el rendimiento del programa, reduciendo así su tiempo de ejecución.

Paralelismo implementado

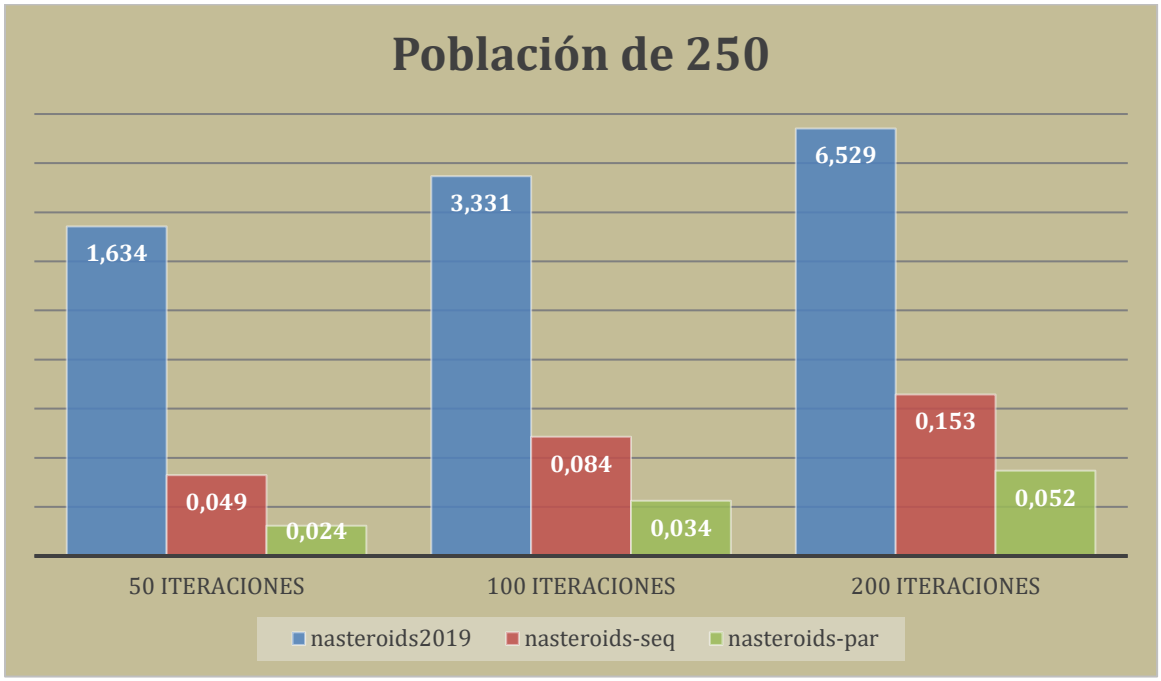
La estructura del código de la parte paralelizada es la misma que en la versión secuencial, salvo la implementación de los comandos reconocidos por OpenMP para poder paralelizar ciertas secciones del código, todo ello con el fin de obtener un mejor rendimiento.
Hemos aplicado paralelismo siempre a partes dentro del main, concretamente dentro del bucle de iteraciones. Hemos utilizado la sentencia `#pragma omp parallel for shared(as_copia, ps, as)` en el primer bucle for para poder paralelizar el recorrido de todos los asteroides del mapa (en este bucle se encuentran todas las operaciones a realizar de fuerzas, distancias...). Para lograr la paralelización de este bucle fue necesario poner como variables compartidas as_copia, ps y as, que se trata de vectores a los que cada hilo debería tener acceso actualizado para evitar así que un hilo tenga una versión desactualizada de la variable con la que va a trabajar en común con otros hilos. Después dentro del bucle en el que hemos implementado esa directiva pragma hemos aplicado paralelismo a los condicionales que hemos utilizado para evaluar los choques con los límites del mapa usando así la siguiente directiva `#pragma omp parallel`. Hemos puesto como variable compartida en este caso el vector de asteroides as para así realizar las comprobaciones pertinentes desde cada hilo.

Impacto de la paralelización

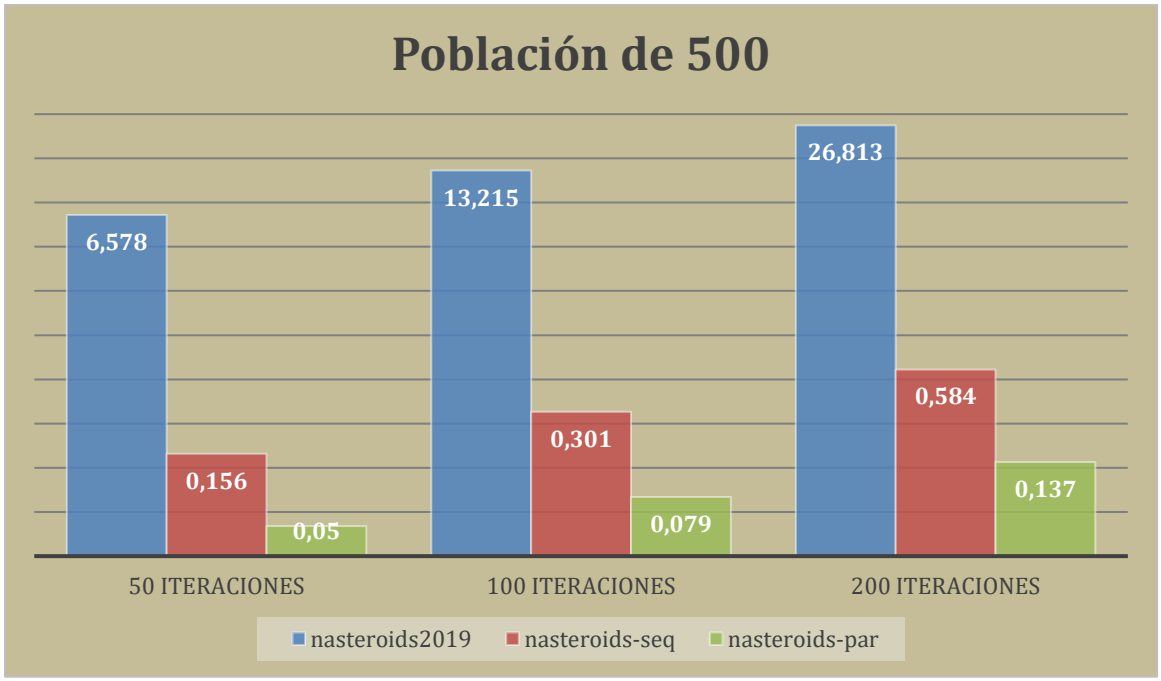
En este apartado se muestra de forma gráfica la diferencia en tiempo de ejecución del código secuencial y el código paralelo. La herramienta usada para la obtener estos tiempos ha sido el comando time, el cual hemos usado al ejecutar ambos códigos mostrando los valores de tiempo real, user y sys de cada ejecución. Para esta comparación únicamente hemos tenido en cuenta el tiempo real, ya que es el tiempo de ejecución del programa desde que se ejecuta hasta que finaliza. Además, la proporción planetas-asteroides utilizada para la evaluación del rendimiento es de un 75% planetas y 25 % asteroides.

Estas pruebas han sido realizadas en un portátil con un procesador Intel Core i5-8250U con un total de 4 núcleos, siendo capaz de lanzar hasta un total de 8 hilos: https://ark.intel.com/content/www/es/es/ark/products/124967/intel-core-i5-8250u-processor-6m-cache-up-to-3-40-ghz.html.
Para la realización de las diferentes pruebas, se han cerrado las aplicaciones en ejecución, conectado el portátil a la corriente mediante el cargador y se ha activado el modo avión.

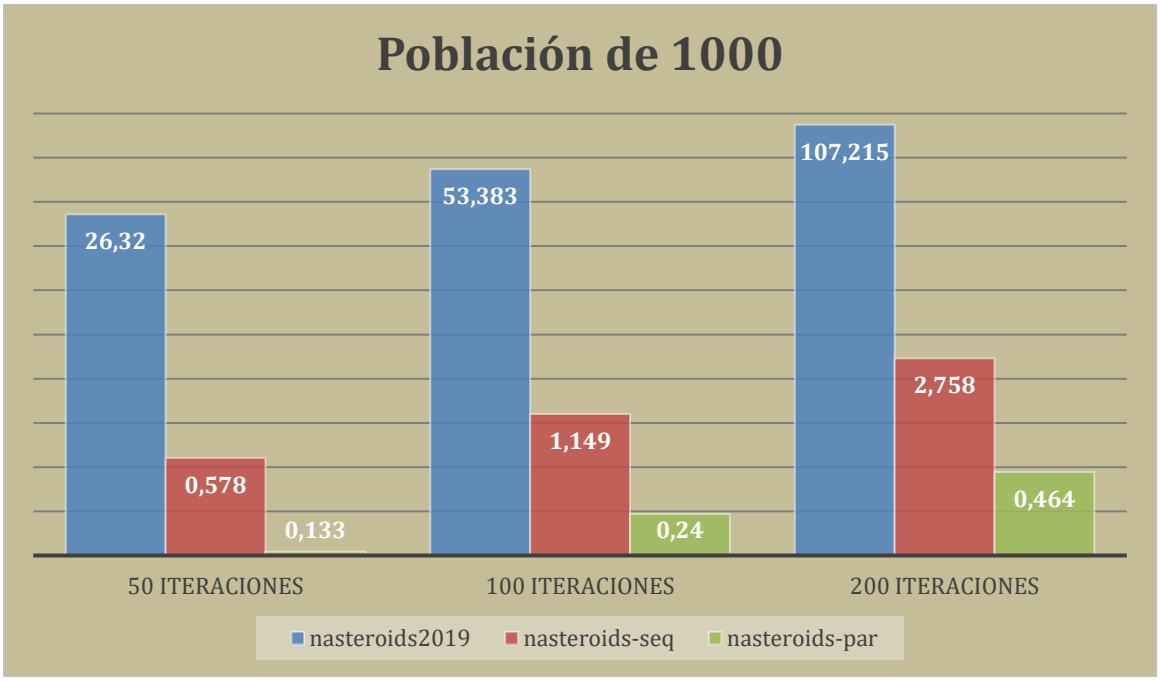
Población de 250 (62 asteroides y 188 planetas)



Población de 500 (125 asteroides y 375 planetas)



Población de 1000 (250 asteroides y 750 planetas)



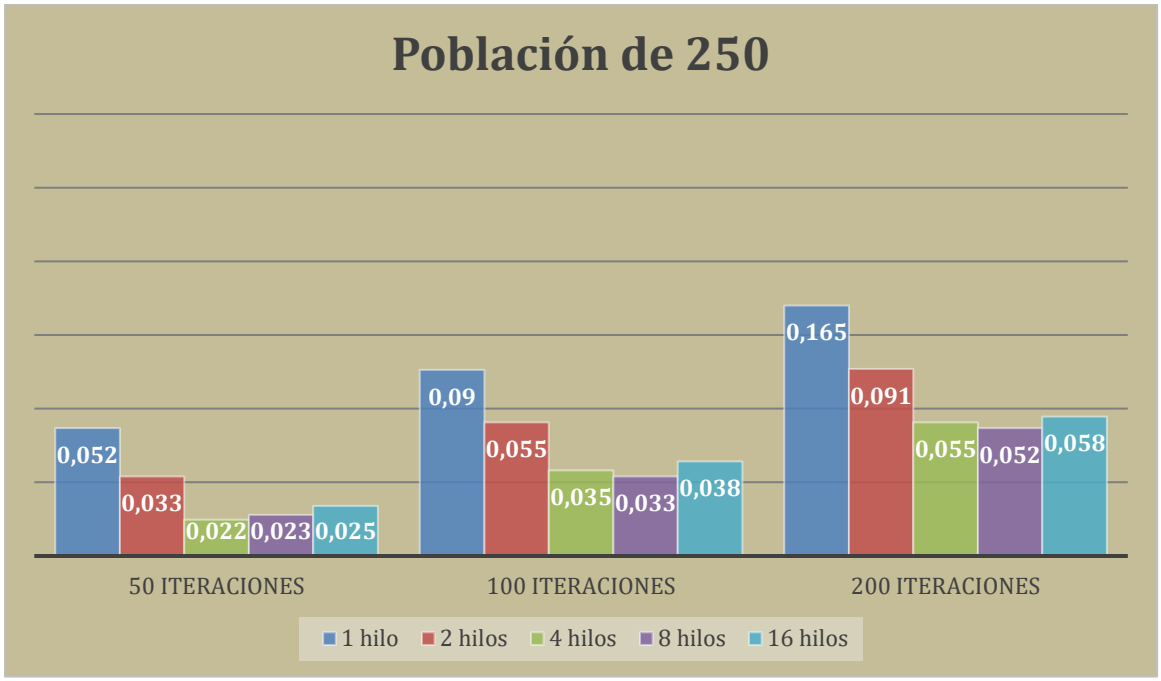
Como conclusión podemos observar que el ejecutable proporcionado para la práctica es con diferencia el más lento de los tres, esto es debido en gran parte a la generación del archivo step_by_step.txt, el cual aumenta en gran medida el tiempo de ejecución. Entre la versión paralela del código y la versión secuencial se puede apreciar una reducción de aproximadamente la mitad en los casos de poblaciones e iteraciones más bajas y hasta una reducción de casi una sexta parte del secuencial en los casos con poblaciones más altas y con mayor número de iteraciones.

Esta gran diferencia de tiempo es debida a que cuanto mayor sean las iteraciones la paralelización reduce el tiempo mediante la gestión de hilos de manera más significativa, ya que la paralelización del bucle de iteraciones reduce en gran medida el tiempo de ejecución al distribuir las iteraciones totales del bucle entre los diferentes hilos. En el caso de los condicionales if dentro del bucle de iteraciones, antes de la paralelización se realizaban individualmente cuatro condicionales if de forma individual y ahora en cada iteración los hilos se distribuyen los diferentes if, ejecutándolos de una sola vez.

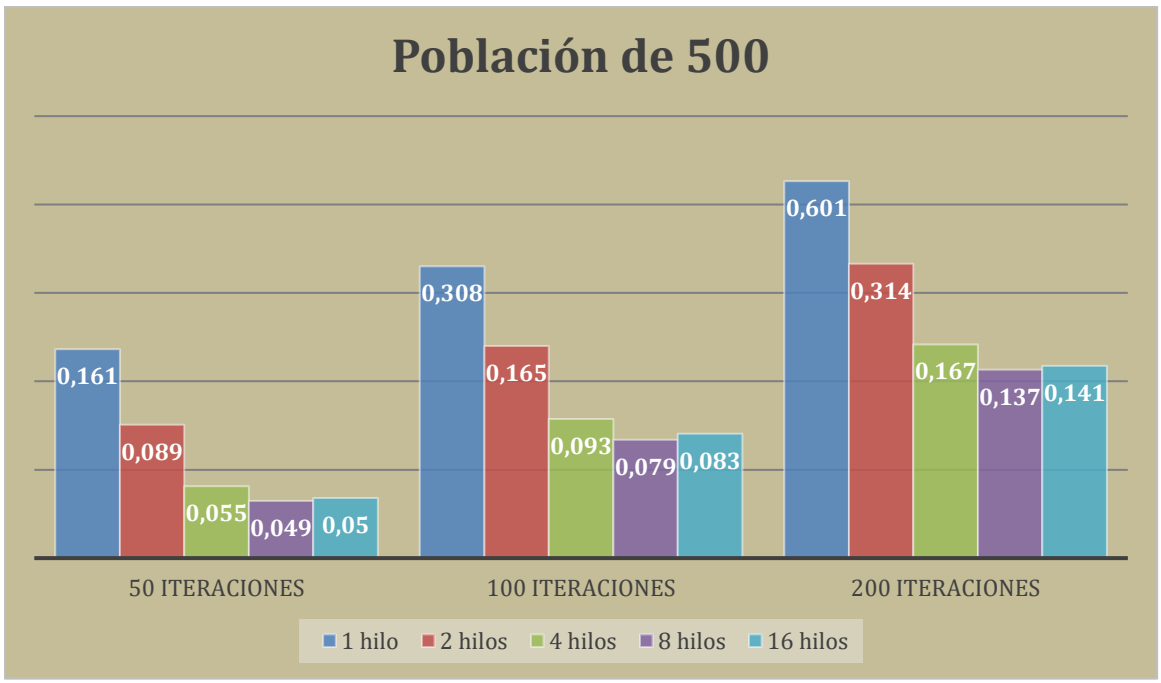
Rendimiento con el uso de hilos

En este apartado se mostrará de forma gráfica el rendimiento obtenido mediante el uso de diferentes cantidades de hilos en el código paralelizado, agrupándose en un total de tres graficas con los diferentes tipos de poblaciones, siguiendo los mismos patrones y escalas que en el apartado anterior.

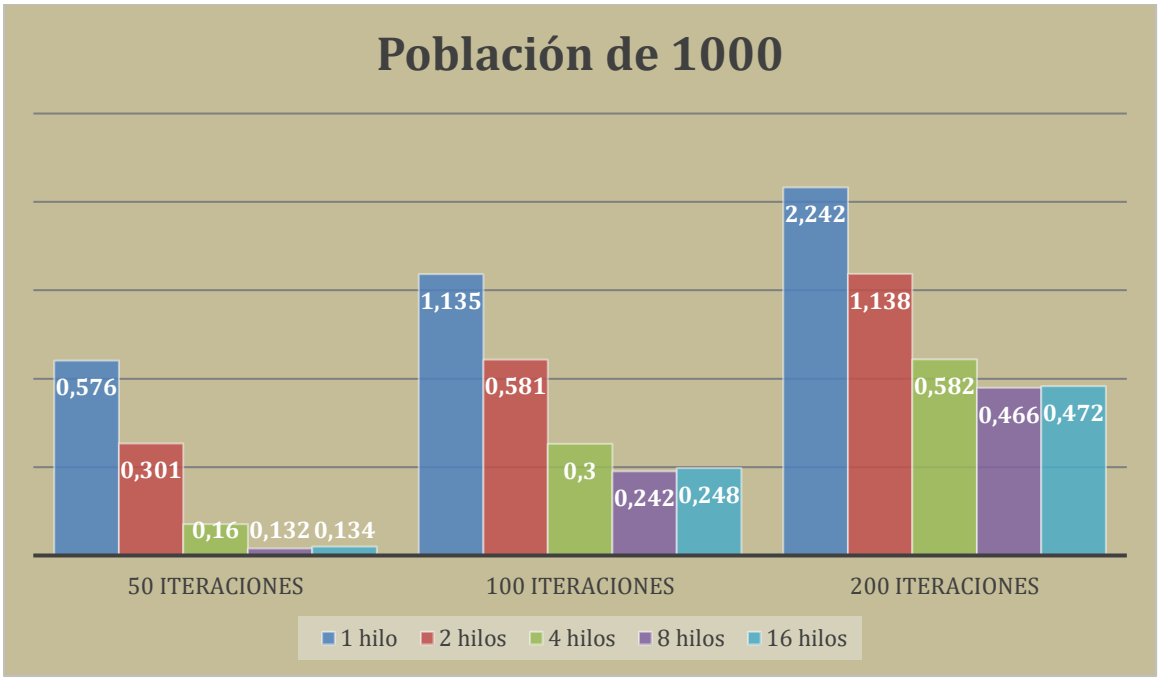
Población de 250 (62 asteroides y 188 planetas)



Población de 500 (125 asteroides y 375 planetas)



Población de 1000 (250 asteroides y 750 planetas)



Podemos concluir principalmente que los tiempos de uso con 1 hilo son bastante grande en comparación con el resto y además para el caso de 2 hilos se ve una gran optimización en comparación con el caso de 1 hilo ya que casi se reduce a la mitad, la diferencia de tiempo entre 2 y 4 hilos de nuevo se reduce a casi la mitad con 4 que con 2 sin embargo la diferencia entre 4 y 8 hilos no es tan grande. Por último, para el caso de 16 hilos se observa que lejos de reducir los tiempos lo que ocurre es que aumentan esto se debe a que el ordenador utilizado para realizar las pruebas la máxima cantidad de hilos que es capaz de lanzar es de 8 lo que supone que no puede lanzar los 16 hilos y al intentarlo lejos de optimizar aumenta un poco el tiempo de ejecución al tratar de lanzar eso 8 hilos extra que no es capaz.

Speed up

Tras analizar todos los speed up entre la versión secuencial y la paralelizada con los diferentes tamaños de población y cantidad de iteraciones estos son los resultados obtenidos:

	50 Iteraciones	100 Iteraciones	200 Iteraciones
Población de 250	2,042	2,471	2,942
Población de 500	3,810	3,810	4,263
Población de 1000	4,787	4,788	5,944

Se puede observar que el speed up aumenta en gran medida cuantas más iteraciones y más cuerpos haya, ya que las secciones de código que son paralelizables se ejecutan más veces y por ende el tiempo de ejecución destinado es mucho menor. Aunque el speed up aumenta más en los casos donde el número de cuerpos es el que aumenta, en lugar de las iteraciones, ya que una de las secciones más paralelizadas es el bucle que se ejecuta tantas veces como asteroides haya.

Impacto de la planificación

Para ver el verdadero impacto que tiene el tipo de planificación en la paralelización de los bucles que hemos realizado hemos hecho pruebas con el caso de 250 asteroides, 200 iteraciones, 750 planetas y 2000 de semilla. Para cada modo de planificación hemos hecho combinaciones omitiendo su tamaño de código a repartir a cada hilo (chunk-size), poniendo el chunk-size a 4 y a 8. También realizaremos las pruebas con 4 y con 8 hilos de ejecución especificando en openmp. Investigando hemos visto a simple vista que si no se especifica el tipo de planificación a realizar por defecto siempre va a trabajar con la de tipo “static”, pero a pesar de ello deducimos comprobarlo poniendo cada tipo de planificación con cada combinación de chunk-size especificada anteriormente.
Con 4 hilos: para schedule(static), es decir sin especificar el chunk-size obtenemos los valores a los que nos encontrábamos inicialmente sin poner el schedule, con schedule(static,4) y schedule(static,8) los tiempos empeoran dado que al omitir el parámetro chunk-size coge el más óptimo openmp. Al usar schedule(guided) los resultados en el tiempo de ejecución mejoran respecto a static, luego al pasar como chunk-size 4 y 8 empeoran o se mantienen iguales los resultados. Por último, aplicando la planificación schedule(dynamic) logramos obtener el tiempo más reducido, omitiendo el chunk-size.
Usando 8 hilos obtenemos los mismos resultados sobre la ejecución (proporcionales en cuanto al tiempo de ejecución) y confirmamos así que la planificación más óptima es la “dynamic” ya que consigue obtener el tiempo más reducido de todas las pruebas que hemos realizado.

CASOS DE PRUEBA GENERALES

Estas pruebas, al igual que los tiempos del apartado de impacto de la paralelización han sido realizadas en un portátil con un procesador Intel Core i5-8250U con un total de 4 núcleos, siendo capaz de lanzar un total de 8 hilos. Para la realización de las diferentes pruebas, se han cerrado las aplicaciones en ejecución, conectado el portátil a la corriente mediante el cargador y se ha activado el modo avión.

Nombre	**Entrada por consola**	**Resultado esperado**	**Resultado obtenido**
Uso de números negativos	./nasteroids-seq -2 -5 -3 -2000	nasteroids-seq: Wrong arguments. Correct use:	nasteroids-seq: Wrong arguments. Correct use:

		nasteroids-seq num_asteroides num_iteraciones num_planetas semilla: Success	nasteroids-seq num_asteroides num_iteraciones num_planetas semilla: Success
	./nasteroids-par -2 -5 -3 -2000	nasteroids-par: Wrong arguments. Correct use: nasteroids-par num_asteroides num_iteraciones num_planetas semilla: Success	nasteroids-par: Wrong arguments. Correct use: nasteroids-par num_asteroides num_iteraciones num_planetas semilla: Success
Uso de valores double	./nasteroids-seq 2.0 5.0 3.0 2000.0 ./nasteroids-par 2.0 5.0 3.0 2000.0	Resultado con los valores truncados (convertido a int).	Resultado con los valores truncados (convertido a int).
Uso de valores char	./nasteroids-seq a b c d	nasteroids-seq: Wrong arguments. Correct use: nasteroids-seq num_asteroides num_iteraciones num_planetas semilla: Success	El programa convierte los parámetros introducidos de char a entero, teniendo como resultado un fichero out.txt vacío.
	./nasteroids-par a b c d	nasteroids-par: Wrong arguments. Correct use: nasteroids-par num_asteroides num_iteraciones num_planetas semilla: Success	
Omisión de parámetros	./nasteroids-seq	nasteroids-seq: Wrong arguments. Correct use:	nasteroids-seq: Wrong arguments. Correct use:

		nasteroids-seq num_asteroides num_iteraciones num_planetas semilla: Success	nasteroids-seq num_asteroides num_iteraciones num_planetas semilla: Success
	./nasteroids-par	nasteroids-par: Wrong arguments. Correct use: nasteroids-par num_asteroides num_iteraciones num_planetas semilla: Success	nasteroids-par: Wrong arguments. Correct use: nasteroids-par num_asteroides num_iteraciones num_planetas semilla: Success
Parámetros con valor 0	./nasteroids-seq 0 0 0 2000 ./nasteroids-par 0 0 0 2000	Ejecución correcta con ficheros de salida vacíos.	Ejecución correcta con ficheros de salida vacíos.
Semilla con valor 0	./nasteroids-seq 250 200 750 2000	nasteroids-seq: Wrong arguments. Correct use: nasteroids-seq num_asteroides num_iteraciones num_planetas semilla: Success	nasteroids-seq: Wrong arguments. Correct use: nasteroids-seq num_asteroides num_iteraciones num_planetas semilla: Success
	./nasteroids-par 250 200 750 2000	nasteroids-par: Wrong arguments. Correct use: nasteroids-seq num_asteroides num_iteraciones num_planetas semilla: Success	nasteroids-par: Wrong arguments. Correct use: nasteroids-seq num_asteroides num_iteraciones num_planetas semilla: Success
Mismos parámetros de asteroides planetas e interacciones pero semilla	./nasteroids-seq 10 20 10 2000 ./nasteroids-par 10 20 10 2000	Los out.txt de cada configuración son diferentes al cambiar la semilla.	Los ficheros resultantes confirmamos que son diferentes para cualquier cambio de

diferente para confirmar que el random está bien	se compara con: ./nasteroids-seq 10 20 10 1500 ./nasteroids-par 10 20 10 1500		semilla.
Planetas no múltiplos de cuatro para ver que se siga el orden (según como hemos hecho nuestra implementación)	./nasteroids-seq 10 20 7 2000 ./nasteroids-par 10 20 7 2000	Deben generarse correctamente todos los planetas	Comprobamos que tanto la generación de los plantas como la solución final son correctos.
Caso con baja población y bajas iteraciones	./nasteroids-seq 62 50 188 2000 ./nasteroids-par 62 50 188 2000	El out.txt coincide con el producido por el ejecutable proporcionado.	Hay pequeñas variaciones en los decimales de la posición X en cada uno de los asteroides.
Caso extremo con alta población y altas iteraciones.	./nasteroids-seq 250 200 750 2000 ./nasteroids-par 250 200 750 2000	El out.txt coincide con el producido por el ejecutable proporcionado.	En algunos casos muy puntuales el valor de la posición X varía en más de 5 la solución esperada, esto puede ser debido a un cálculo erróneo en los choques entre asteroides.

CONCLUSIONES SOBRE LA PRÁCTICA

En términos generales consideramos que esta práctica ha sido problemática, ya que hay varias ambigüedades en el enunciado como el tema de las colisiones entre más de dos asteroides, además de no especificar una manera de evaluar la mejora de rendimiento. En cuanto al desarrollo de las pruebas, se han presentado dificultades por la limitación del número de núcleos.