Arquitectura de computadores:
Práctica de programación paralela con OpenMP

Daniel Zubieta Pascual – 100346138
Alfonso Sánchez Domingo – 100346065
Ignacio González Díaz-Tendero – 100346133
Sergio Cruzado Muñoz – 100346032

Contenido

1. Introducción

Para esta práctica hemos tenido que implementar el problema de los n-asteroides en C++. Para ello, hemos tenido que crear dos versiones del mismo programa: una con una implementación secuencial, y otra con una implementación paralela usando OpenMP.

Para simular este escenario, debemos considerar las fuerzas de atracción que hay en un espacio bidimensional, las fuerzas generadas por los asteroides y los planetas, que se ubican en los márgenes de nuestro espacio. Otro factor clave es, sin duda, que el espacio es cerrado, implicando que ningún asteroide pueda abandonar la zona de acción. El número de asteroides, planetas e iteraciones, junto a una semilla generadora de números aleatorios, son introducidos por parámetros al ejecutar el programa. Como se ha mencionado antes sobre la semilla, dos ejecuciones con los mismos parámetros, pero con distinta semilla arrojarán resultados diferentes.

El objetivo de la práctica es resolver el problema propuesto de dos maneras distintas y evaluar el rendimiento del código implementado, compararlo y sacar nuestras propias conclusiones.

2. Versión secuencial

2.1 Constantes utilizadas

Las constantes que se han utilizado en la implementación son las siguientes (todos los valores son especificados en el enunciado de la práctica):

- double Gravity: La gravedad que se ejercen entre los cuerpos, y tiene un valor de 6.674e-5.
- double intTime: Intervalo de tiempo, con un valor de 0.1.
- double dmin: Distancia mínima entre cuerpos, tiene un valor de 5.0.
- double width y double height: Representan la anchura y Altura del espacio bidimensional en el cual se simula el escenario, ambos poseen un valor de 200.0.
- int m: Media para el cálculo de la distribución normal de las masas, su valor es de 1000.
- int stdev: Desviación estándar para el cálculo de la distribución normal de las masas, su valor es de 50.

2.2 Cálculos

2.2.1 Cálculos de las fuerzas ejercidas por otros cuerpos

En esta función utilizaremos la tercera ley de Newton para calcular la fuerza entre los cuerpos. Para ello, la función calcula la contribución de todos los demás elementos a su movimiento.

Para calcular la fuerza que un asteroide ejerce sobre otro primero debemos calcular la distancia que existe entre ellos, y se realiza con la siguiente fórmula:

$$dist(X,Y) = \sqrt{(distX)^2(distY)^2};\quad being\quad distX = x_a - x_b;\quad distY = y_a - y_b$$

2.2.2 Cálculo de las fuerzas ejercidas entre dos cuerpos

Se utiliza la constante gravitacional G, la masa de los dos cuerpos, y la distancia entre los mismos, en la siguiente fórmula:

$$force = \frac{G \cdot Ma \cdot Mb}{dist(X,Y)^2}$$

El valor máximo permitido para la fuerza es de 200, por lo que cualquier valor superior a este será truncado a dicho valor.

2.2.3 Cálculo de la pendiente

La pendiente la necesitamos para calcular el ángulo que hay entre dos cuerpos, y para eso utilizamos la siguiente fórmula:

$$slope = \frac{diffy}{diffx};\quad being\quad diffy = y_a - y_b;\quad diffx = x_a - x_b$$

Si en algún momento la pendiente nos arroja un valor inferior a -1, este valor se truncará a -1. Del mismo modo si el valor de la pendiente es superior a 1, se truncará a 1.

2.3 Función main

Lo primero que hace el programa en la función principal es comprobar que los argumentos introducidos son correctos y están dentro de los límites que se especifican en el enunciado. Una vez comprobados que sean correctos, los guardamos en variables para su posterior uso.

Tras esto lo que hace el programa es inicializar un reloj, cuyo único objetivo es medir el tiempo real que lleva el procesamiento de la función main, cosa que nos será de gran ayuda para poder evaluar el rendimiento del programa.

Después, se inicializan los asteroides y los planetas, junto a sus correspondientes parámetros.

Una vez hecho todo esto, se da paso al bucle for principal que se ejecutará el número de iteraciones introducidas por parámetro. En el mismo lo que se hace es calcular las fuerzas entre los cuerpos y los respectivos choques entre los mismos.

Cuando el programa termina el bucle for principal, se arrojan los resultados en un fichero de salida out.txt.

Finalmente, se mide el tiempo final de la función main, se resta a este valor el tiempo inicial y se muestra por pantalla el resultado.

3. Versión paralela

En esta parte se explicará como se ha paralelizado la parte secuencial de nuestro código con OpenMP. La parte del código que se ha decidido paralelizar ha sido aquella en la que se realizan los cálculos entre asteroides y entre asteroides y planetas.

OpenMP nos permite paralelizar ciertas secciones de código. Para aquellas porciones de código más pesadas, es decir, donde se pierde más tiempo haciendo cálculos, con las directivas que permite OpenMP podemos hacer que dichas porciones de código se ejecuten por varios procesadores, logrando así una ejecución más rápida del programa.

En este caso específico, se ha decidido paralelizar solamente los bucles relativos al cálculo de fuerzas y a la actualización de posición y velocidad, debido a que el resto de bucles contienen secciones críticas que son necesarias ejecutar de forma secuencial.

Para realizar las pruebas se han implementado los 3 tipos de planificaciones posibles:

- Estática (static), donde se reparte un número fijo de iteraciones contiguas entre los diferentes hilos.
- Dinámica (dynamic), donde se reparten las iteraciones entre los hilos a medida que se van ejecutando.
- Guiada (guided), donde se van repartiendo secciones de iteraciones contiguas restando a medida que los hilos completan las iteraciones que tenían asignadas.

4. Evaluación del rendimiento

Tanto la correcta implementación del código secuencial como la del código paralelo, suponen, una parte importante de la práctica, pero, también es fundamental realizar un estudio del rendimiento de la ejecución de ambos programas.

En este apartado se evaluará el rendimiento del programa secuencial, con una serie de combinaciones de elementos. Sin embargo, para evaluar la versión paralela, se ha decidido hacer la mayor parte de las pruebas, es decir, con todos los hilos posibles (1,2,4,8 y16) con una planificación dinámica, debido a que, a nuestro parecer, es la planificación que podría obtener mayor rendimiento. Posteriormente se han utilizado las planificaciones estáticas y guiadas para hacer la comparación, con 4 y 8 hilos.

Para realizar todas las pruebas se ha utilizado una computadora de las aulas de informática con las siguientes especificaciones:

4.1 Evaluación de la versión secuencial

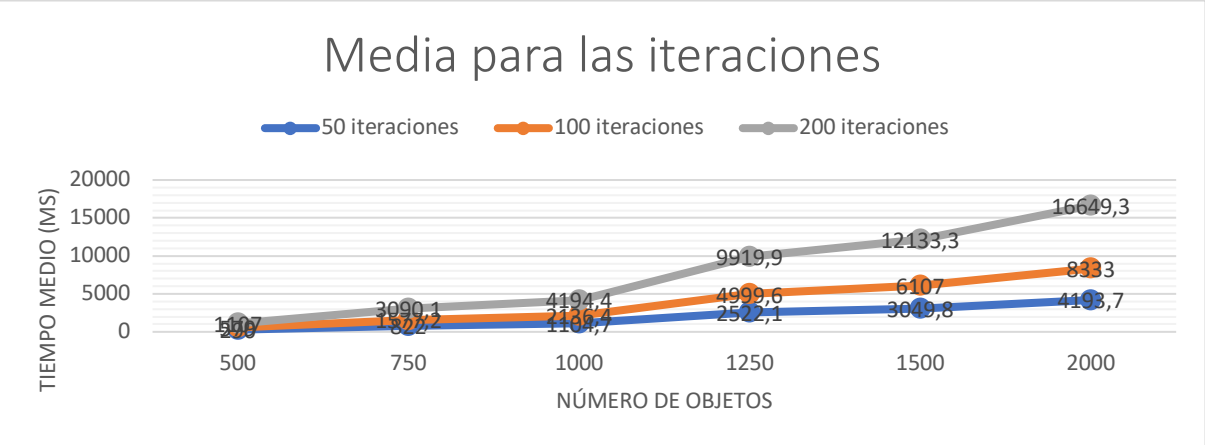
En este apartado se procederá a evaluar el rendimiento del programa en su versión secuencial, y poder extrapolar posteriormente, las posibles mejoras que supongan la ejecución del programa en su versión paralela.

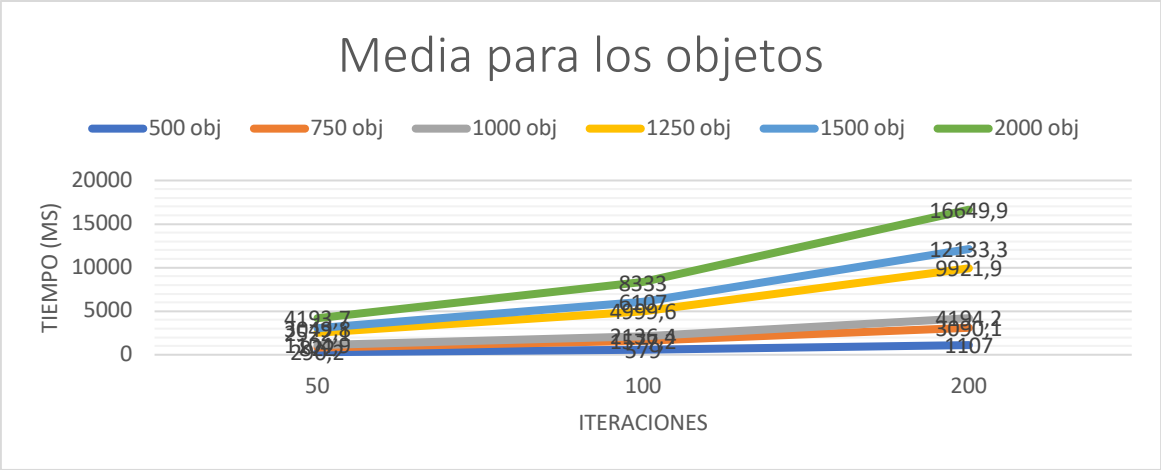
Se utilizarán varias poblaciones distintas, sin embargo, en todas las pruebas se ha utilizado la misma semilla: 2000. *Nota: Todos los tiempos están representados en milisegundos (ms).*

Prueba	Objetos	Iteraciones	Asteroides	Planetas	Semilla	Media
1	500	50	250	250	2000	290.2
2	500	100	250	250	2000	579
3	500	200	250	250	2000	1107
4	750	50	500	250	2000	822
5	750	100	500	250	2000	1577.2
6	750	200	500	250	2000	3090.1
7	1000	50	500	500	2000	1104.9
8	1000	100	500	500	2000	2136.4
9	1000	200	500	500	2000	4194.2
10	1250	50	1000	250	2000	2522.1
11	1250	100	1000	250	2000	4999.6
12	1250	200	1000	250	2000	9921.9
13	1500	50	1000	500	2000	3049.8
14	1500	100	1000	500	2000	6107
15	1500	200	1000	500	2000	12133.3
16	2000	50	1000	1000	2000	4193.7
17	2000	100	1000	1000	2000	8333
18	2000	200	1000	1000	2000	16649.3

Prueba	1	2	3	4	5	6	7	8	9	10
1	284	305	318	277	275	282	302	298	280	281
2	540	607	555	602	578	567	597	580	574	590
3	1157	1110	1111	1094	1108	1098	1099	1101	1108	1084
4	809	834	845	828	796	822	814	818	830	824
5	1551	1608	1612	1548	1602	1576	1555	1587	1560	1573
6	3092	3088	3116	3072	3075	3080	3102	3096	3088	3092
7	1111	1087	1052	1131	1135	1105	1098	1099	1110	1121
8	2192	2092	2177	2129	2095	2134	2142	2139	2123	2141
9	4160	4213	4234	4198	4195	4190	4188	4176	4192	4196
10	2532	2506	2580	2505	2510	2515	2523	2511	2517	2522
11	4946	4956	5011	5029	5012	5005	5010	5012	5013	5002
12	9864	9952	9921	9982	9893	9934	9923	9901	9929	9920
13	3085	3061	3054	3039	3067	3041	3054	3033	3021	3043
14	6078	6120	6111	6143	6097	6098	6101	6107	6111	6104
15	12118	12161	12130	12123	12167	12120	12130	12134	12108	12142
16	4226	4241	4191	4210	4165	4178	4170	4192	4187	4177
17	8363	8381	8291	8351	8321	8332	8322	8318	8322	8329
18	16683	16596	16827	16691	16594	16678	16594	16623	16609	16598

Para cada una de las pruebas se han realizado 10 ejecuciones, de las cuales haremos la media y utilizaremos dicho valor como el representativo de la prueba.





Como se puede observar, a medida que aumentamos el número de objetos y el número de iteraciones, el tiempo de ejecución aumenta progresivamente. Este hecho está dentro de lo esperado, ya que el programa está la mayor parte del tiempo de ejecución recorriendo bucles.

4.2 Evaluación de la versión paralela

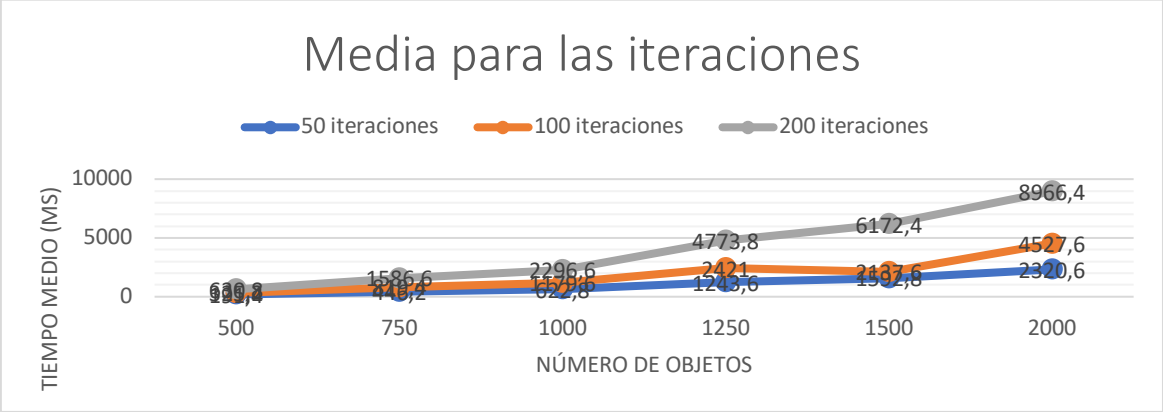
En este apartado analizaremos los resultados arrojados por la ejecución de la versión paralela y los compararemos entre sí con las distintas planificaciones de bucles. Originalmente y la mayor parte de batería de pruebas se han realizado con el planificador dinámico, debido a que hemos considerado que sería el más eficiente.

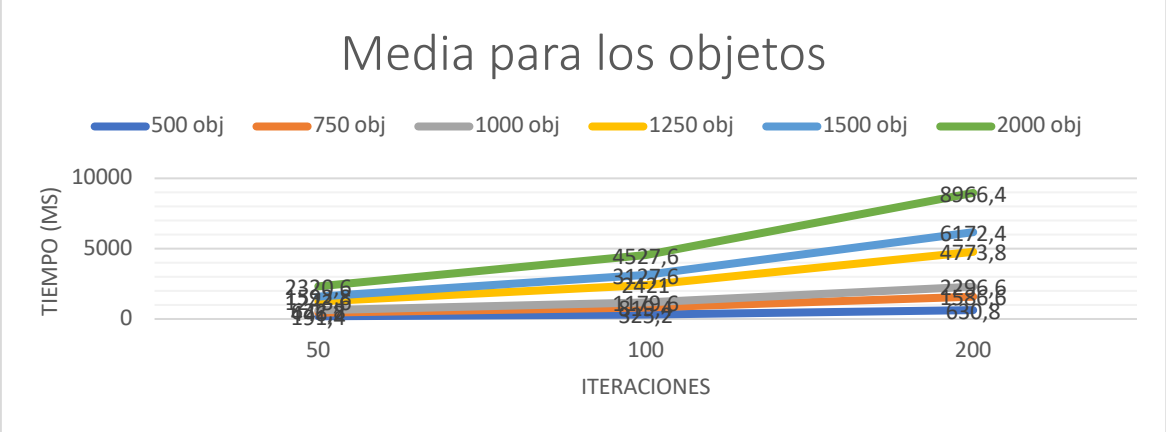
En la siguiente tabla vemos la media de tiempo de ejecución del programa con los distintos números de hilos.

Prueba	Objetos	Iteraciones	Asteroides	Planetas	Semilla	Media 2 hilos	Media 4 hilos	Media 8 hilos	Media 16 hilos
1	500	50	250	250	2000	191.4	145,8	134,4	148,4
2	500	100	250	250	2000	325,2	241,4	234	281,6
3	500	200	250	250	2000	630,8	396,6	422,2	443,6
4	750	50	500	250	2000	446,2	301,4	280,2	329,6
5	750	100	500	250	2000	819,4	553	567,8	598,4
6	750	200	500	250	2000	1586,6	1046	1051,6	1088
7	1000	50	500	500	2000	622,8	416	386,2	416
8	1000	100	500	500	2000	1179,6	757,6	775,2	796,8
9	1000	200	500	500	2000	2296,6	1485,4	1496,2	1522,6
10	1250	50	1000	250	2000	1243,6	803,4	840	823,2
11	1250	100	1000	250	2000	2421	1526,8	1594,4	1599,6
12	1250	200	1000	250	2000	4773,8	3044,4	3098,6	3154,2
13	1500	50	1000	500	2000	1592,8	1018,2	1033	1076
14	1500	100	1000	500	2000	3127,6	1962,6	2033,8	2036,2
15	1500	200	1000	500	2000	6172,4	3908,8	3974,2	4011
16	2000	50	1000	1000	2000	2320,6	1445,4	1472,6	1474,2

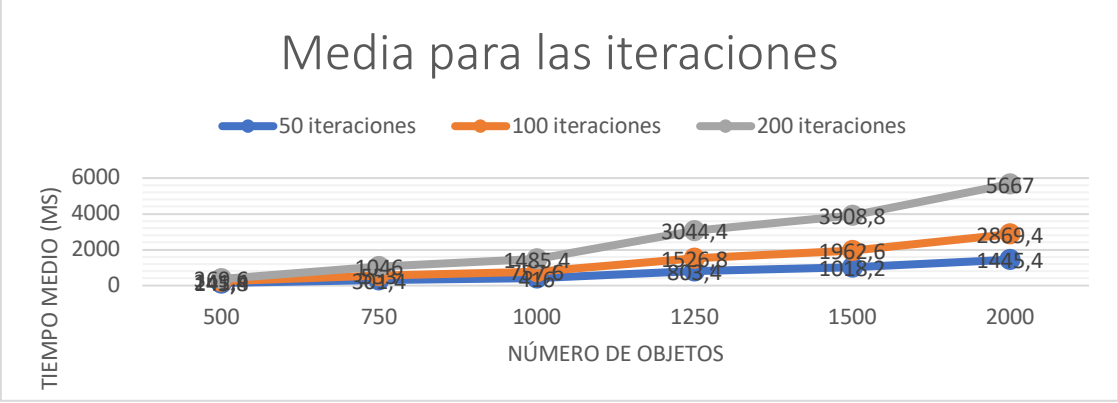
17	2000	100	1000	1000	2000	4527,6	2869,4	2920,8	2913,6
18	2000	200	1000	1000	2000	8966,4	5667	5739	5743,4

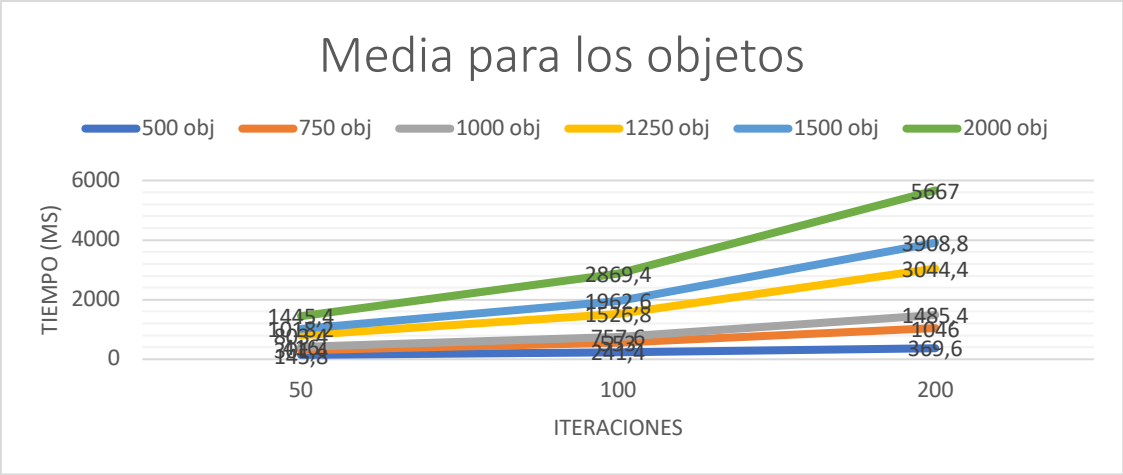
Podemos observar que la ejecución alcanza su pico de rendimiento con 4 hilos. Este hecho era de esperar debido a que el computador con el cual se han realizado las pruebas tiene 4 núcleos físicos.



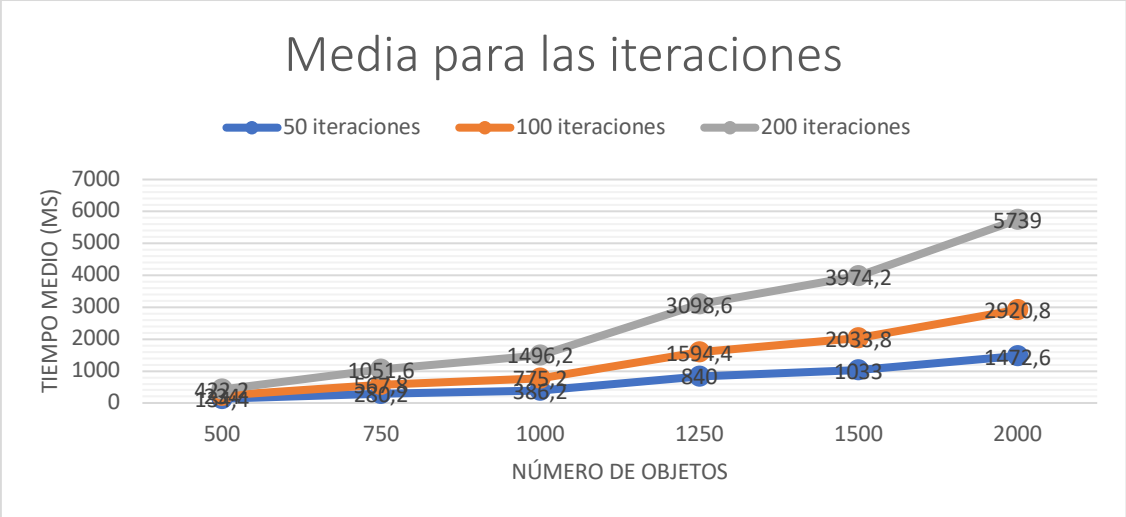


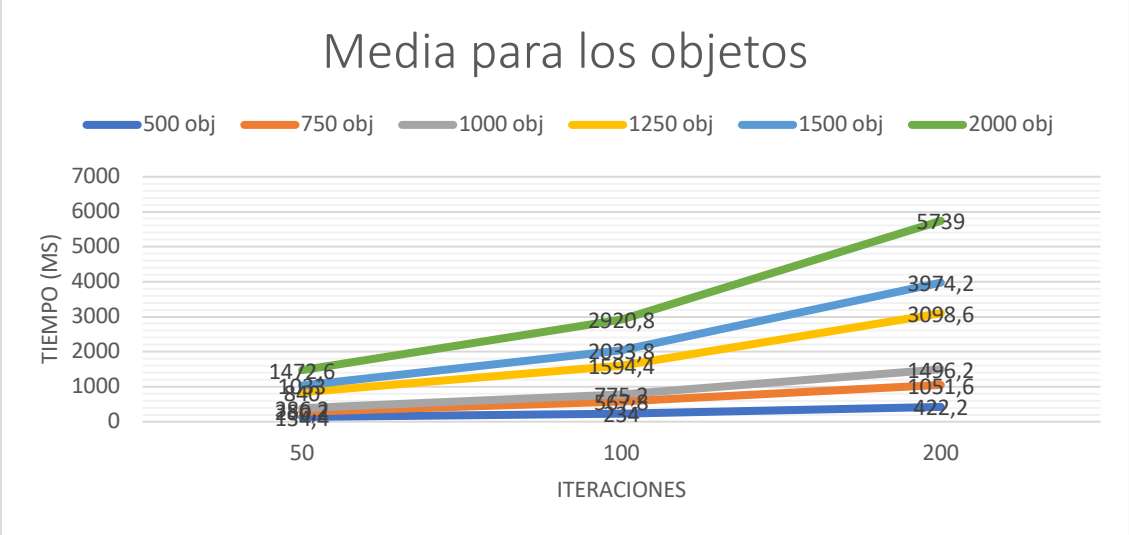
Versión con 4 hilos.





Versión de 8 hilos. A partir de aquí, detectamos que no hay más mejora de rendimiento.

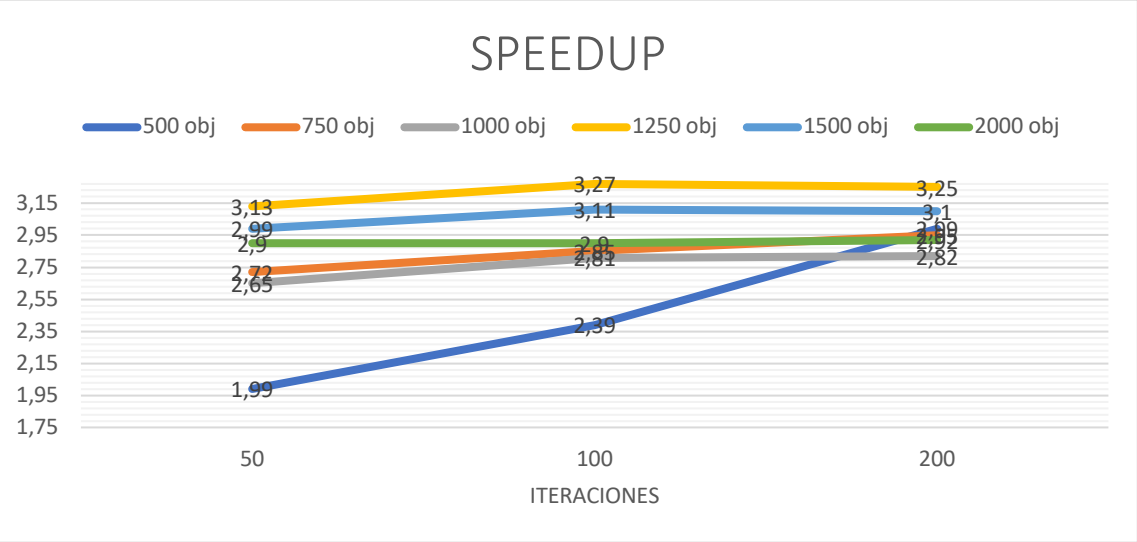




La versión de 16 hilos no ponemos gráfico porque el rendimiento es muy parecido a esta versión de 8 hilos, incluso tiene peor rendimiento. Es lógico al pensar que el ordenador tiene 4 núcleos físicos y no se puede tener 16 hilos trabajando a la vez.

Como podemos ver, el rendimiento ha mejorado bastante respecto a la versión secuencial presentada con anterioridad. En el caso óptimo, que es la ejecución con 4 hilos (los máximos hilos dados los núcleos de las máquinas donde hemos realizado las pruebas), el tiempo se reduce en 10 segundos aproximadamente, teniendo en cuenta el peor de los casos con 2000 objetos y 200 iteraciones.

A continuación, presentaremos una tabla con el estudio del speedup de la versión de 4 hilos, por ser la mejor, respecto a la versión secuencial para ver como ha mejorado la implementación de la versión paralela.

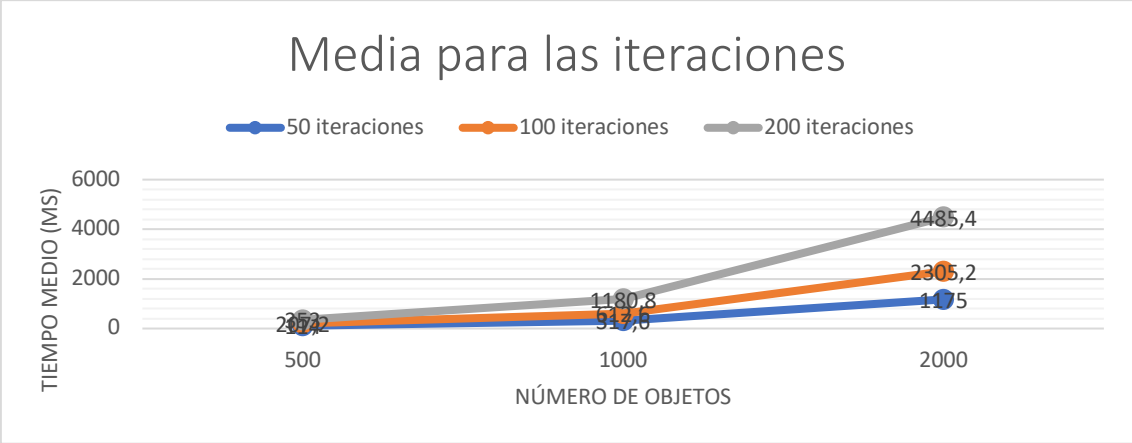


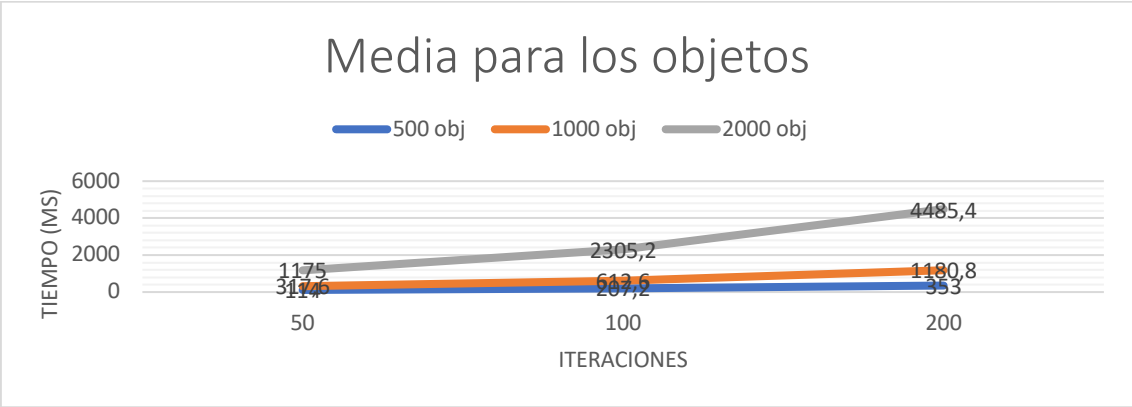
4.3 Impacto de otras planificaciones

En esta sección, vamos a estudiar el impacto de otras planificaciones diferentes a la que hemos usado como principal, en este caso la Dynamic, en la resolución del problema y su rendimiento. En este caso, estudiaremos las versiones estática y guiada con 4 y 8 hilos para ver el impacto.

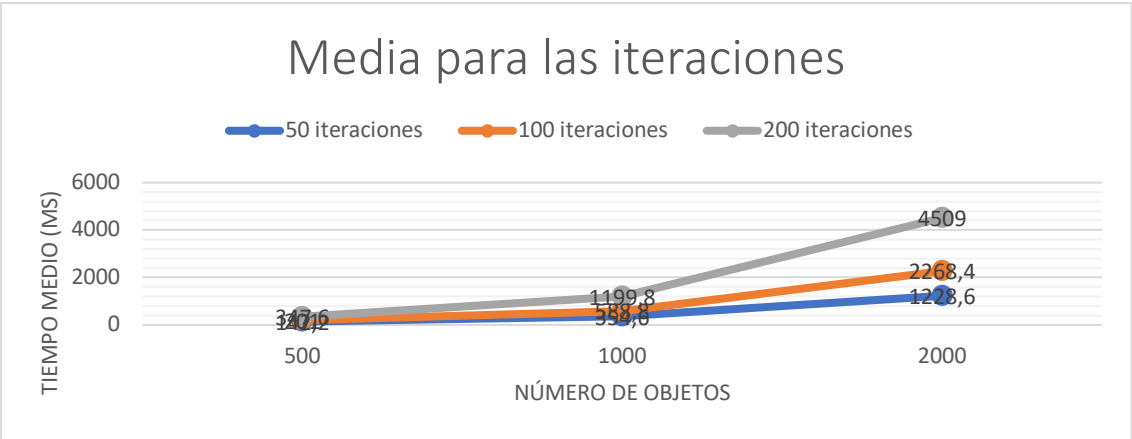
Prueba	Objetos	Iteraciones	Asteroides	Planetas	Media 4 hilos estática	Media 8 hilos estática	Media 4 hilos guiado	Media 8 hilos guiado
1	500	250	50	250	114	142,2	98,6	114,2
2	500	250	100	250	207,2	201	207,6	195,4
3	500	250	200	250	353	347,6	343	295,4
4	1000	500	50	500	317,6	354,6	369	313,8
5	1000	500	100	500	612,6	588,8	618,8	558,6
6	1000	500	200	500	1180,8	1199,8	1277,8	1073,2
7	2000	1000	50	1000	1175	1228,6	1240	1151
8	2000	1000	100	1000	2305,2	2268,4	2469,8	2262
9	2000	1000	200	1000	4485,4	4509	4717,2	4460,2

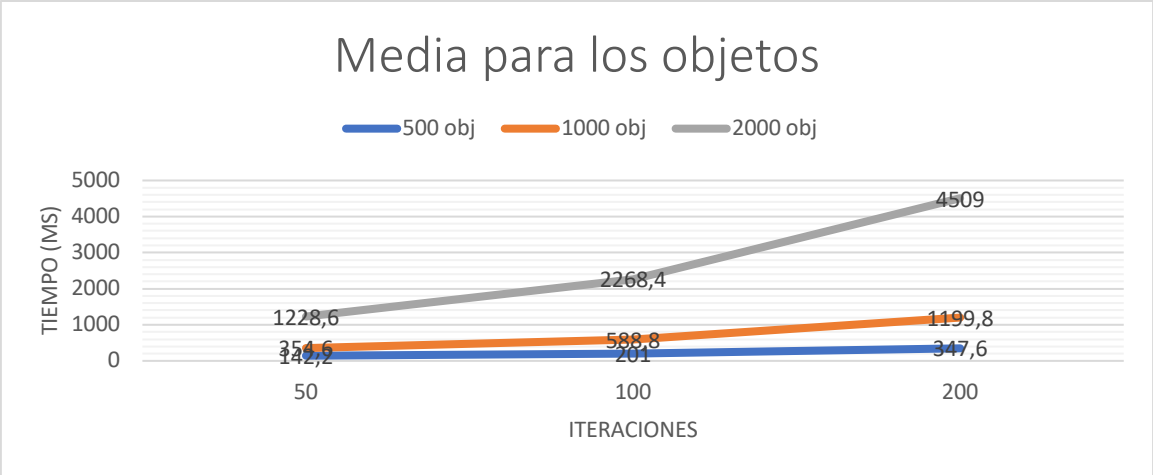
Versión paralela con planificador de bucles estático y 4 hilos:



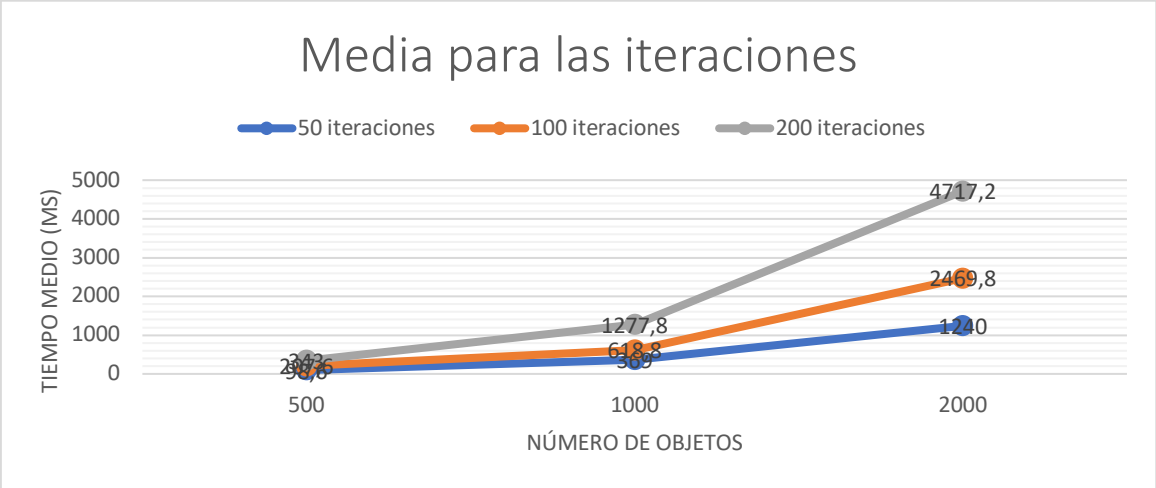


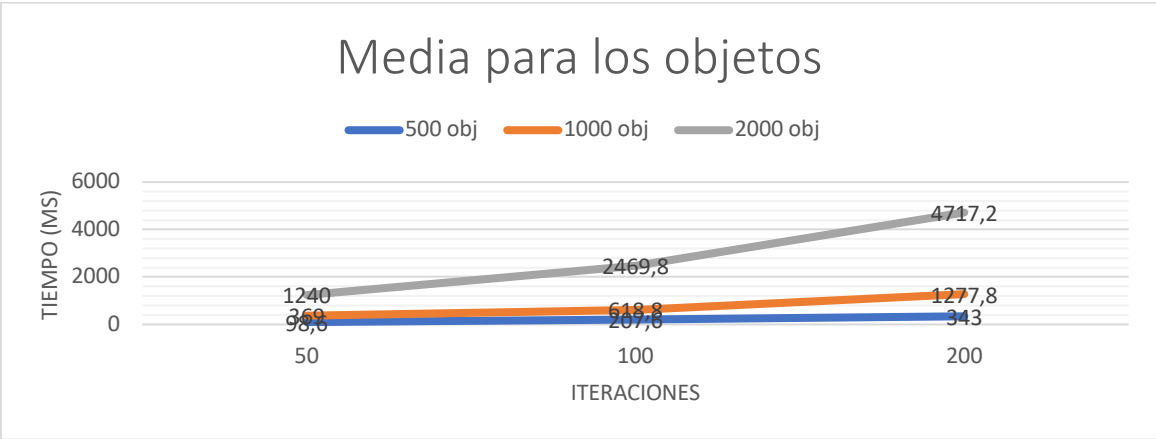
Versión paralela con planificador de bucles estático y 8 hilos:



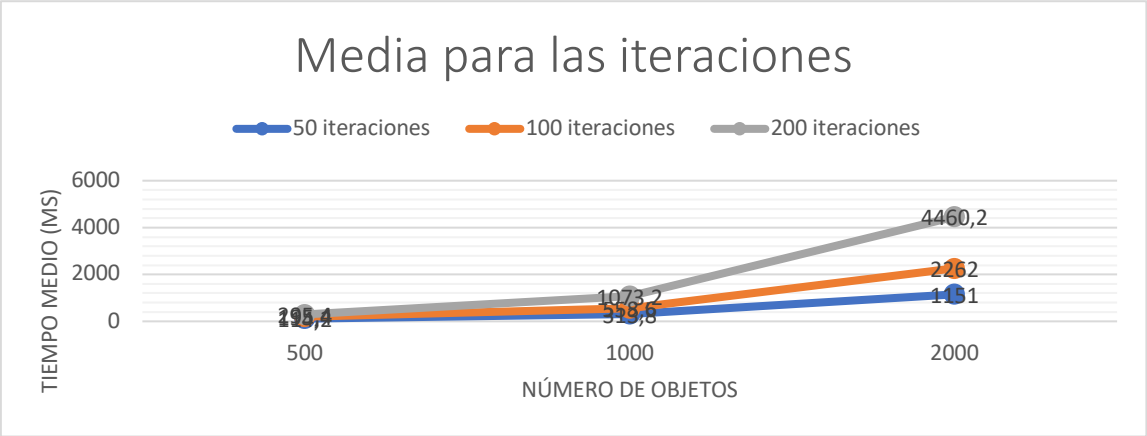


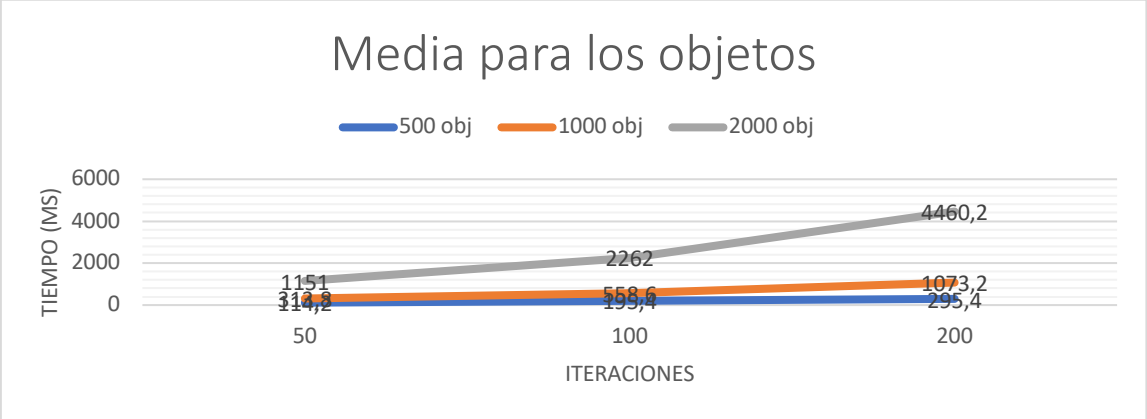
Versión paralela con planificador de bucles guiado y 4 hilos:



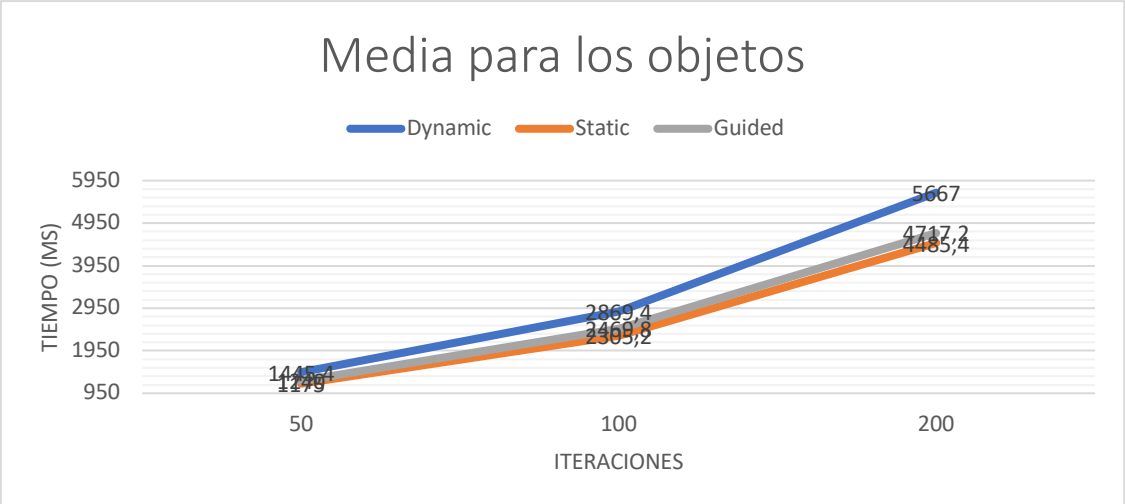


Versión paralela con planificador de bucles guiado y 8 hilos:





Comparativa de tiempos de ejecución entre los distintos modos de planificación de bucles y 4 hilos:



Después de realizar las pruebas, podemos ver como la versión estática en este problema en concreto tiene un mejor tiempo medio de ejecución que la versión

dinámica principal realizada o la versión guided. Entendemos que, al realizar la ejecución de los bucles de forma fija, esto hace que la ejecución vaya más rápido a la hora de pasar a los hilos la información que debe procesar cada uno, haciendo estos accesos un poco más rápido que hacerlo según se ejecuta el programa que pueda haber un retraso de pocos milisegundos en la ejecución final del programa. La versión de planificación guiada es la segunda mejor, ya que arroja los peores resultados de la versión paralela con static, porque asignar cada vez menos secciones a los hilos para que estos las ejecuten hace que en etapas posteriores de la ejecución pueda ir más lento frente a las demás. Por último, la versión escogida al principio es la peor para resolver este problema porque asignar proporciones a medida que se va ejecutando el programa no es buena idea al tener tantas iteraciones y cuerpos con los que calcular cada operación al poder tener segmentos tan grandes que unos hilos tengan más carga de trabajo que otros.

5. Conclusiones

La realización de esta práctica ha servido, entre otras cosas, para conocer más a fondo el funcionamiento del paralelismo en ejecuciones de programas pesados y como resolver con esta funcionalidad nos ayuda para lograr un rendimiento muy superior a las versiones secuenciales que tan común son a la hora de desarrollar.

La batería de pruebas escogida para medir el rendimiento del programa ha sido un problema con respecto al tiempo invertido en esta práctica, motivado por la gran cantidad de pruebas individuales y la monotonía y aburrimiento que provocan las mismas.

Por último, la gran decisión ha sido elegir el método de planificación de bucles para realizar las pruebas principales. En este caso ha sido una planificación dinámica, porque es la que pensábamos que iba a tener mayor rendimiento. Sin embargo, los resultados arrojados por las pruebas y el análisis de los mismos nos han revelado que estábamos completamente equivocados, ya que el mayor rendimiento del programa se ha logrado con una planificación estática.