Grupo 83

# PROGRAMACIÓN PARALELA

*Alejandro Massotti López, 100383565*

*Alberto Lobato Díaz, 100383508*

*Víctor Moreno Azofra, 100383393*

*Javier Fernández Velasco. 100383392*

# CONTENIDO

## INTRODUCCIÓN

La finalidad del trabajo es la simulación del movimiento de unos asteroides mediante el cálculo de las distancias, las velocidades, la pendiente, el ángulo, las fuerzas de atracción y los rebotes con los límites del espacio y entre asteroides. Dicha simulación tendrá que ser abordada tanto de forma secuencial como de forma paralela mediante la interfaz de programación de openMP.

El objetivo de esta práctica es implementar los conocimientos adquiridos en clase en relación con la paralelización de procesos secuenciales.

## DISEÑO SECUENCIAL

La descripción será realizada en orden de aparición en el código.

### Objeto planeta

Se define el objeto planeta con sus atributos de coordenada x, coordenada y y masa. Seguidamente se crea un constructor, un constructor vacío y una función que lo imprime por pantalla.

### Objeto asteroide

Se define el objeto asteroide con sus atributos de coordenada x, coordenada y, masa, velocidad en x, velocidad en y, aceleración en x y aceleración en y. Seguidamente se crea un constructor, un constructor vacío y una función que lo imprime por pantalla.

### distAsteroidePlaneta

Calcula la distancia entre el asteroide y el planeta que entran por parámetro según la fórmula del enunciado. Devuelve un double.

### distAsteroideAsteroide

Calcula la distancia entre los asteroides que entran por parámetro según la fórmula del enunciado. Devuelve un double.

### pendienteAsteroidePlaneta

Calcula la pendiente entre el asteroide y el planeta que entran por parámetro haciendo su posterior ajuste utilizando las fórmulas del enunciado. Devuelve un double.

### pendienteAsteroideAsteroide

Calcula la pendiente entre los asteroides que entran por parámetro haciendo su posterior ajuste utilizando las fórmulas del enunciado. Devuelve un double.

### Angulo

Calcula la arcotangente del valor introducido por parámetro. Devuelve en un double.

### fuerzaAtraccionXAsteroideAsteroide

Calcula la fuerza de atracción entre asteroides introducidos por parámetros según la fórmula del enunciado, si sobrepasa el umbral de 100 0 -100 se le aplicará un ajuste. Devuelve un double.

**fuerzaAtraccionYAsteroidePlaneta**

Calcula la fuerza de atracción entre un asteroide y un planeta introducidos por parámetros según la fórmula del enunciado, si sobrepasa el umbral de 100 0 -100 se le aplicará un ajuste. Devuelve un double.

**aplicacionDeFuerzasXAsteroideAsteroide**

Calcula la aceleración de resultante de aplicar la fórmula del enunciado en el eje x. Recibe por parámetro dos asteroides. Devuelve un objeto asteroide.

**aplicacionDeFuerzasXAsteroidePlaneta**

Calcula la aceleración de resultante de aplicar la fórmula del enunciado en el eje x. Recibe por parámetro un asteroide y un planeta. Devuelve un objeto asteroide.

**aplicacionDeFuerzasYAsteroideAsteroide**

Calcula la aplicación de fuerzas en un asteroide en base a una fuerza y un asteroide introducidos en el eje y. Devuelve un objeto asteroide.

**aplicacionDeFuerzasYAsteroidePlaneta**

Calcula la aceleración de resultante de aplicar la fórmula del enunciado en el eje y. Recibe por parámetro un asteroide y un planeta. Devuelve un objeto asteroide.

**calculoVelocidadX**

Calcula la velocidad resultante de aplicar la fórmula del enunciado en el eje x. Recibe por parámetro un asteroide. Devuelve un asteroide modificado.

**calculoVelocidadY**

Calcula la velocidad resultante de aplicar la fórmula del enunciado en el eje y. Recibe por parámetro un asteroide. Devuelve un asteroide modificado.

**modificarPosicionX**

Calcula la posición resultante de aplicar la fórmula del enunciado en el eje x. Recibe por parámetro un asteroide. Devuelve un asteroide modificado.

**modificarPosicionY**

Calcula la posición resultante de aplicar la fórmula del enunciado en el eje y. Recibe por parámetro un asteroide. Devuelve un asteroide modificado.

**limiteEspacio**

Calcula las interacciones de un asteroide con el borde del espacio, si se sale de este, se variará la posición del asteroide en 5 unidades para volver a introducirlo dentro y se le invertirá la velocidad. Devuelve un asteroide modificado.

**choqueAsteroide**

Intercambia las velocidades de los asteroides introducidos por parámetros. Devuelve un asteroide modificado.

**main**

1. Se inicializa el reloj t1 y se toma el tiempo actual.
2. Se comprueban el número de valores introducidos, si la semilla es un entero (con la función **isValidInt(argv[i])** explicada al final de este apartado) y si la semilla es mayor que 0. Si no es así el programa devuelve -1 y finaliza.
3. Se guarda el número de asteroides, planetas y la semilla introducidos.
4. Se imprimen por pantalla los datos iniciales.
5. Se inicializan los vectores de planetas y asteroides.
6. Se definen las funciones que generan números aleatorios para las posiciones en los ejes y las masas.
7. Se inicializa la salida para el fichero de texto init_conf.txt.
8. Se crean los asteroides con las funciones de aleatoriedad anteriores y se guardan en init_conf.txt.
9. Se crean los planetas con las funciones de aleatoriedad y en los bordes de los ejes según indica el enunciado, su masa se multiplica por 10. Todos estos datos se guardan en init_conf.txt.
10. Se ha utilizado para las pruebas el fichero step_by_step.txt y sus impresiones, todas estas están comentadas para interferir en los resultados finales.
11. Se crea el for que recorre el número de iteraciones introducidas por parámetro.
    a. Se inicializan vectores de para almacenar las fuerzas aplicadas en los ejes x e y, serán utilizados más adelante
    b. Se inicializa el for que recorre el vector de asteroides
        i. Se inicializan las variables sumFuerzasX y sumFuerzasY para ir almacenando el sumatorio de fuerzas en los ejes.
        ii. Se inicializa el for que recorre el vector de asteroides de nuevo, dentro de él se evaluará la distancia entre los asteroides utilizando **distAsteroideAsteroide ().** Si la distancia es mayor que 2 y no se está comparando un asteroide consigo mismo se realiza el cálculo de las fuerzas de atracción utilizando las funciones **fuerzaAtraccionXAsteroideAsteroide () y fuerzaAtraccionYAsteroideAsteroide ().** Seguidamente se incrementa su valor en las variables sumFuerzasX y sumFuerzasY.
        iii. Se inicializa el for que recorre el vector de planetas, dentro de él se evalúan las fuerzas entre el asteroide y el planeta correspondientes utilizando las funciones **fuerzaAtraccionXAsteroidePlaneta () y fuerzaAtraccionYAsteroidePlaneta ().** Seguidamente se incrementa su valor en las variables sumFuerzasX y sumFuerzasY.

c. Se inicializa el for que recorre el vector de asteroides aplicando el sumatorio de fuerzas calculada (**aplicacionDeFuerzasXAsteroideAsteroide () y aplicacionDeFuerzasYAsteroideAsteroide ()**), modificando sus velocidades (**calculoVelocidadX () y calculoVelocidadY ()**) y posiciones utilizando las funciones (**modificarPosicionX () y modificarPosicionY ()**). Seguidamente se evalúa que el asteroide no abandone el límite del espacio utilizando la función **limiteEspacio ().**

d. Se inicializan dos for anidados que recorren los asteroides evaluando los choques entre ellos. Si la distancia es menor o igual que 2 y no se está comparando un asteroide consigo mismo se realiza el cálculo de los choques utilizando las funciones **choqueAsteroide () y choqueAsteroide ().**

12. Se imprimen los resultados finales de los asteroides en un fichero out.txt mediante el recorrido de estos por un for.
13. Se inicializa el reloj t2 y se toma el tiempo actual.
14. Se imprime la diferencia de t2-t1 que es el tiempo resultante en la ejecución.
15. Si todo ha ocurrido correctamente el programa devuelve un 0.

### isValidInt

Función que comprueba mediante un bucle while si el número introducido por parámetro corresponde con un entero. Devuelve true si es así y false si no.

## DISEÑO PARALELO

Para paralelizar el programa se ha utilizado la librería de OPENMP utilizando las funciones **#pragma omp parallel num_threads(NUMERO_HILOS), #pragma omp parallel for schedule(static) y #pragma omp parallel for reduction** donde NUMERO_HILOS es una variable global definida globalmente. Se han considerado estos puntos a análisis en la paralelización:

### Punto 11 en secuencial, for de iteraciones

No se ha podido paralelizar debido a que estas ejecuciones han de realizarse en orden. Por tanto, es equivalente a un proceso secuencial con operaciones atómicas.

### Punto 11 b en secuencial, for de asteroides

Se considera paralelizable ya que el orden de aplicación de las fuerzas al ser sumadas todas al final es irrelevante (propiedad conmutativa de la suma)

### Punto 11 b ii en secuencial, for de fuerzas entre asteroides

Se considera paralelizable ya que el orden de aplicación de las fuerzas entre asteroides al ser sumadas todas al final es irrelevante (propiedad conmutativa de la suma). Se ha utilizado **#pragma omp parallel for reduction** para controlar estas variables de suma global (sumFuerzasX y sumFuerzasY)

### Punto 11 b iii en secuencial, for de fuerzas entre asteroides y planetas

Se considera paralelizable ya que el orden de aplicación de las fuerzas entre asteroides y planetas al ser sumadas todas al final es irrelevante (propiedad conmutativa de la suma). Se ha utilizado **#pragma omp parallel for reduction** para controlar estas variables de suma global (sumFuerzasX y sumFuerzasY)

### Punto 11 c en secuencial, aplicación de fuerzas y cambio de velocidad y posición

Se considera no paralelizable, ya que consta de operaciones atómicas que necesariamente han de ser realizadas en el orden de aparición. Como conclusión, genera la necesidad de que esta parte tenga una estructura secuencial.

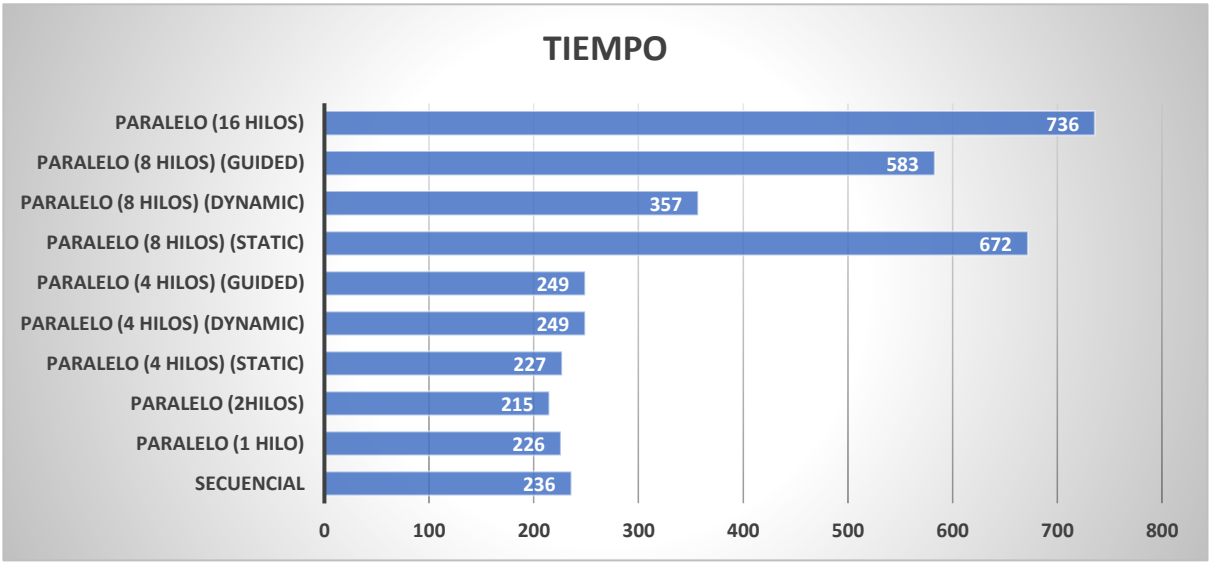
### Punto 11 d en secuencial, rebotes entre asteroides

Se considera paralelizable, ya que el orden de rebotes entre asteroides es irrelevante, ya que al final se acabarán ejecutando todos. Se utiliza **#pragma omp parallel for schedule(static)** ya que no hay variables que dividir.

## EVALUACIÓN DE RENDIMIENTO

Las pruebas se han realizado con un ordenador Intel coreTM i5-8250U, con 4 núcleos físicos y 8 virtuales, 8 GB de RAM, con la siguiente jerarquía de memoria caché: Caché 1 (256 kB), Caché 2 (1,0 MB) y Caché 3 (6,0 MB).

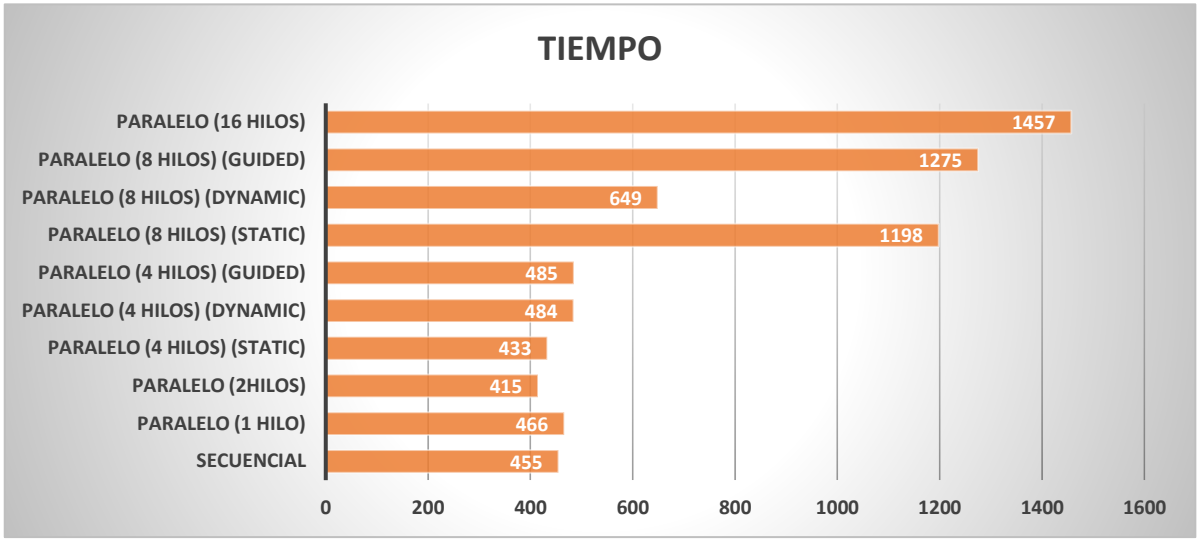
### GRÁFICA 250 OBJETOS 50 ITERACIONES




El speedup que encontramos en esta gráfica es el siguiente:

- Paralelo (1 hilo): 236/226 = 1,044
- Paralelo (2 hilos): 236/215 = 1,097
- Paralelo (4 hilos) (Static): 236/227 = 1,039

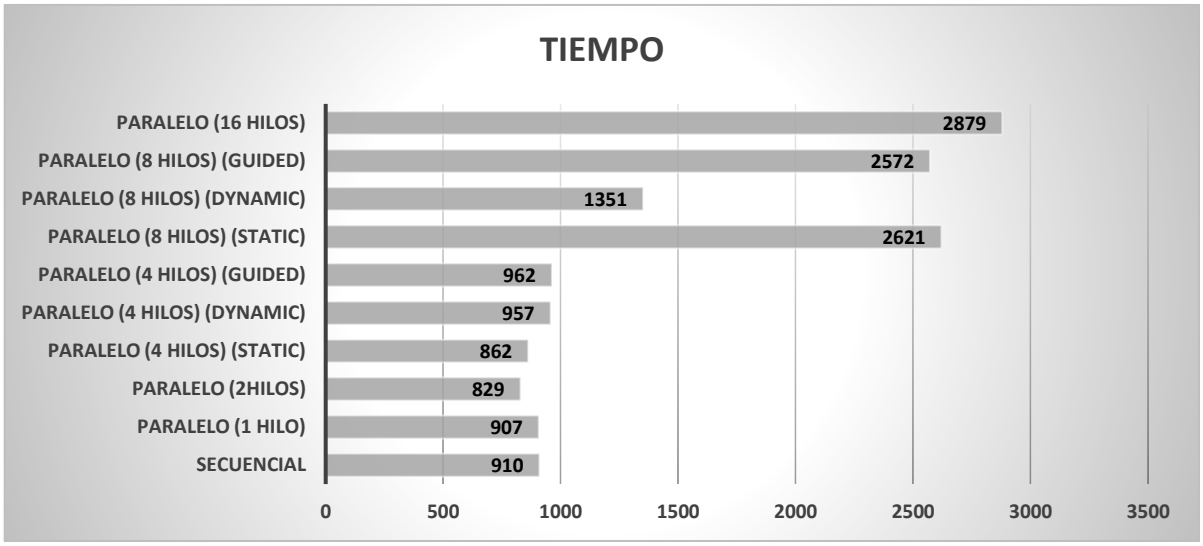
## GRÁFICA 250 OBJETOS 100 ITERACIONES




El speedup que encontramos en esta gráfica es el siguiente:

- Paralelo (2 hilos): 455/415 = 1,096
- Paralelo (4 hilos) (Static): 455/433 = 1,05

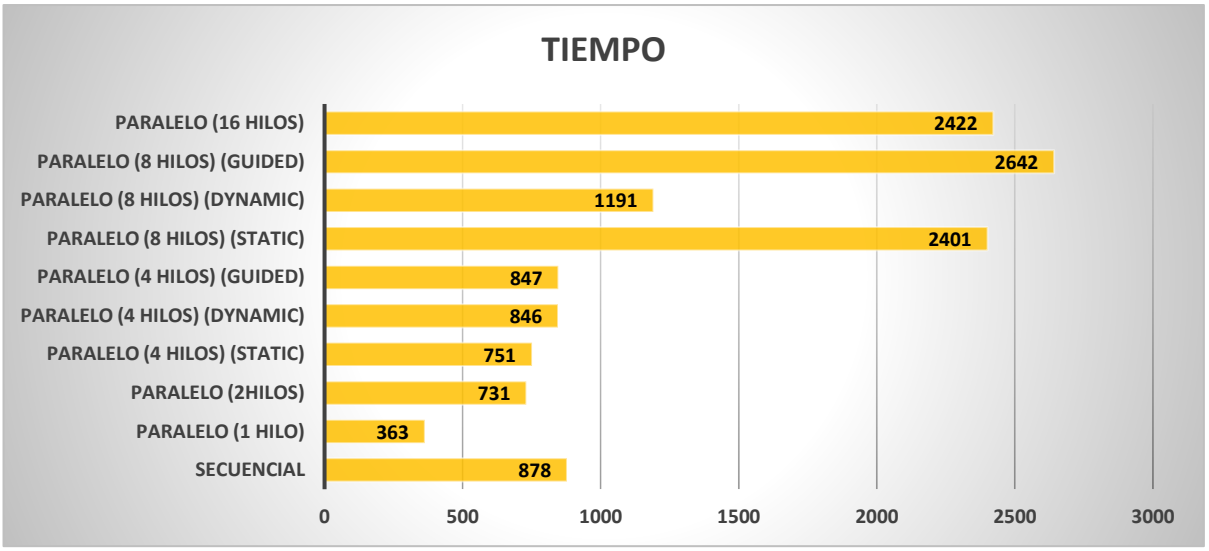
## GRÁFICA 250 OBJETOS 200 ITERACIONES




El speedup que encontramos en esta gráfica es el siguiente:

- Paralelo (1 hilo): 910/907 = 1,003
- Paralelo (2 hilos): 910/829 = 1,097
- Paralelo (4 hilos) (Static): 910/862 = 1,055

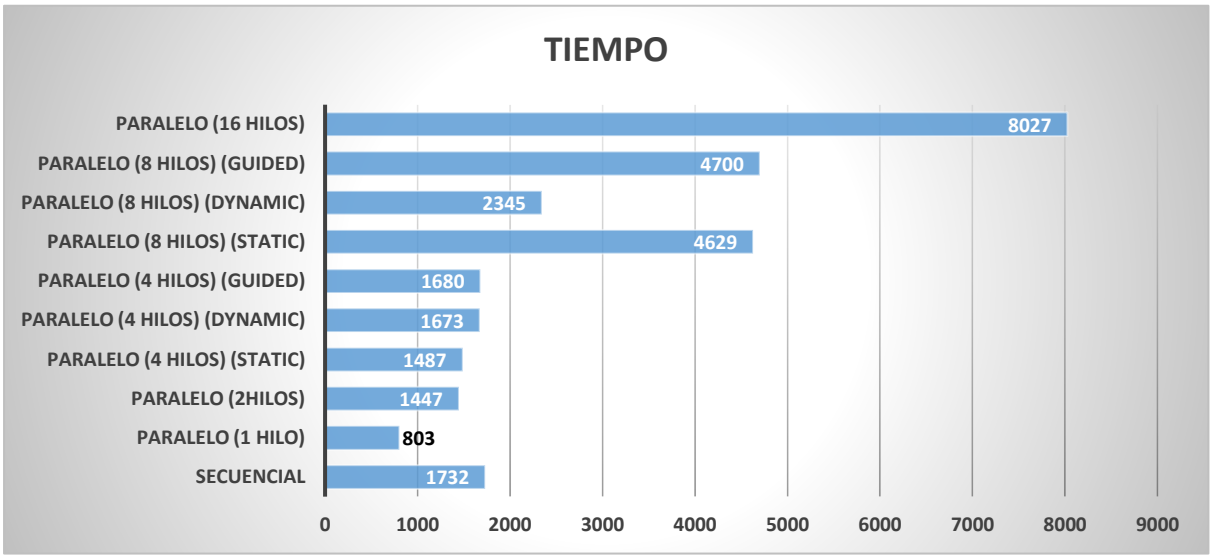
## GRÁFICA 500 OBJETOS 50 ITERACIONES




El speedup que encontramos en esta gráfica es el siguiente:

- Paralelo (1 hilo): 878/363 = 2,418
- Paralelo (2 hilos): 878/731 = 1,201
- Paralelo (4 hilos) (Static): 878/751 = 1,169
- Paralelo (4 hilos) (Dynamic): 878/846 = 1,037
- Paralelo (4 hilos) (Guided): 878/847 = 1,036

## GRÁFICA 500 OBJETOS 100 ITERACIONES

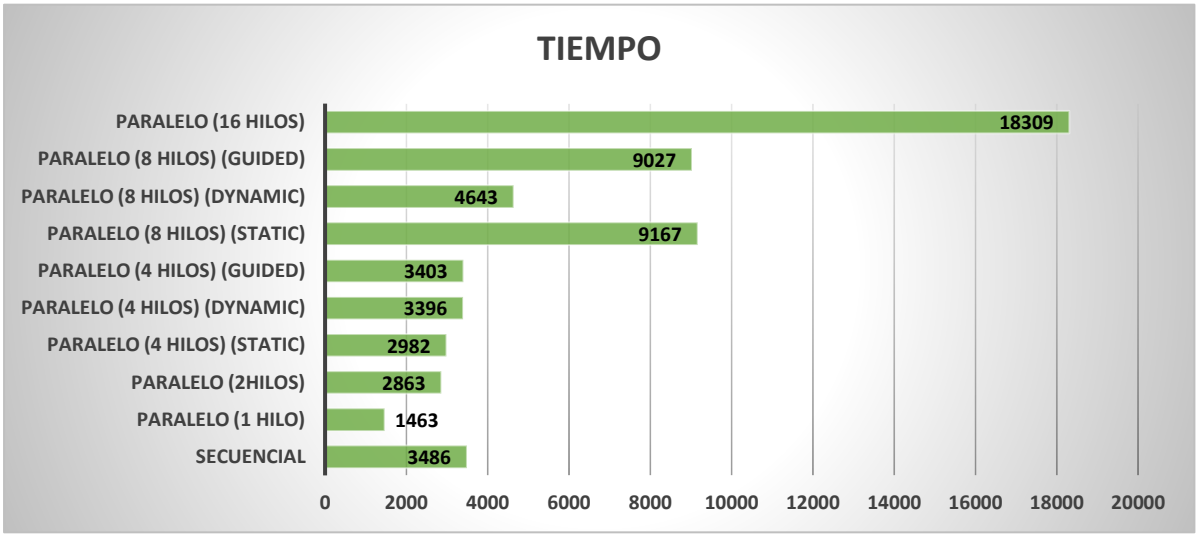



El speedup que encontramos en esta gráfica es el siguiente:

- Paralelo (1 hilo): 1732/803 = 2,156

- Paralelo (2 hilos): 1732/1447 = 1,196
- Paralelo (4 hilos) (Static): 1732/1487 = 1,164
- Paralelo (4 hilos) (Dynamic): 1732/1673 = 1,035
- Paralelo (4 hilos) (Guided): 878/847 = 1,036

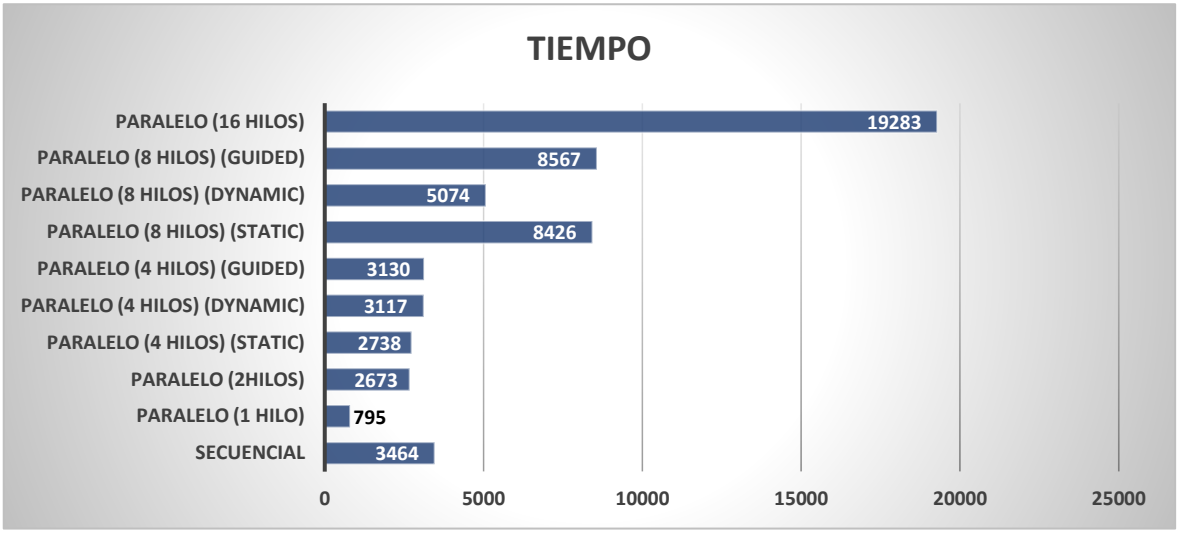
## GRÁFICA 500 OBJETOS 200 ITERACIONES




El speedup que encontramos en esta gráfica es el siguiente:

- Paralelo (1 hilo): 3486/1463 = 2,382
- Paralelo (2 hilos): 3486/2863 = 1,217
- Paralelo (4 hilos) (Static): 3486/2982 = 1,169
- Paralelo (4 hilos) (Dynamic): 3486/3396 = 1,026
- Paralelo (4 hilos) (Guided): 3486/3403 = 1,024

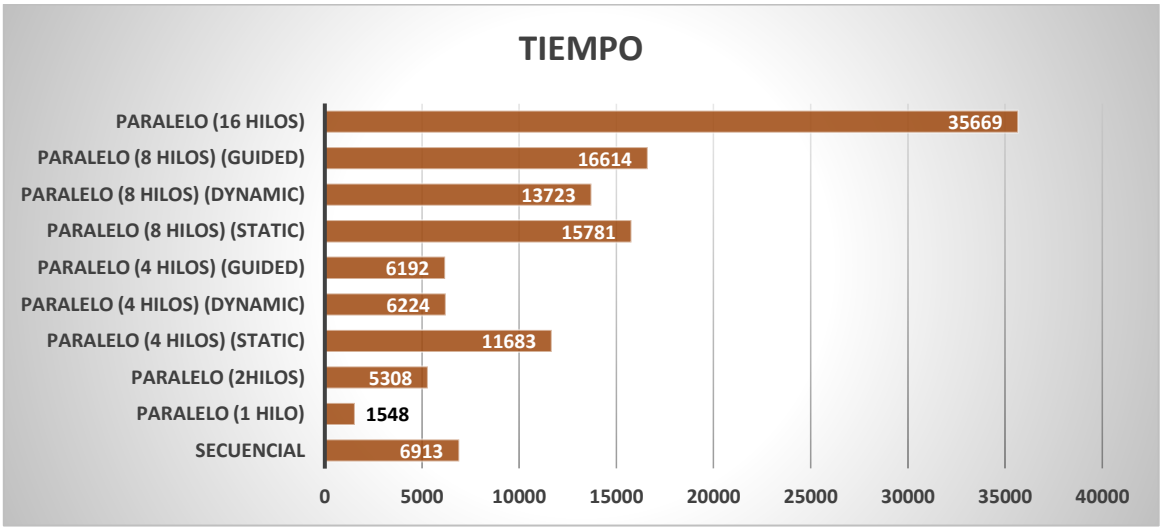
## GRÁFICA 1000 OBJETOS 50 ITERACIONES




El speedup que encontramos en esta gráfica es el siguiente:

- Paralelo (1 hilo): 3463/795 = 4,355
- Paralelo (2 hilos): 3463/2673 = 1,295
- Paralelo (4 hilos) (Static): 3463/2738 = 1,264
- Paralelo (4 hilos) (Dynamic): 3463/3117 = 1,111
- Paralelo (4 hilos) (Guided): 3463/3130 = 1,106

## GRÁFICA 1000 OBJETOS 100 ITERACIONES

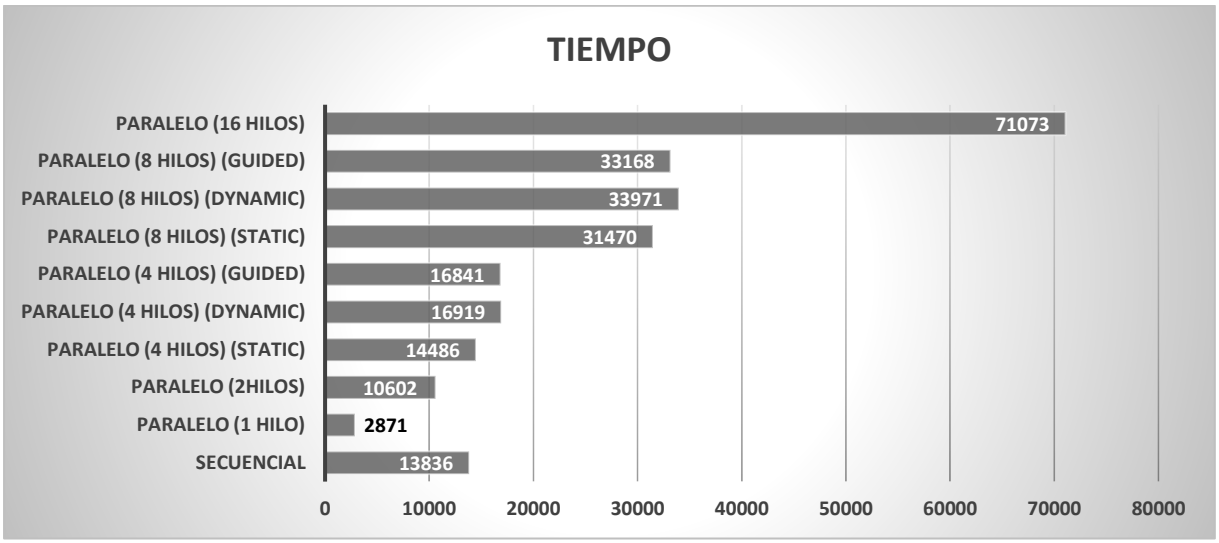



El speedup que encontramos en esta gráfica es el siguiente:

- Paralelo (1 hilo): 6913/1548 = 4,465

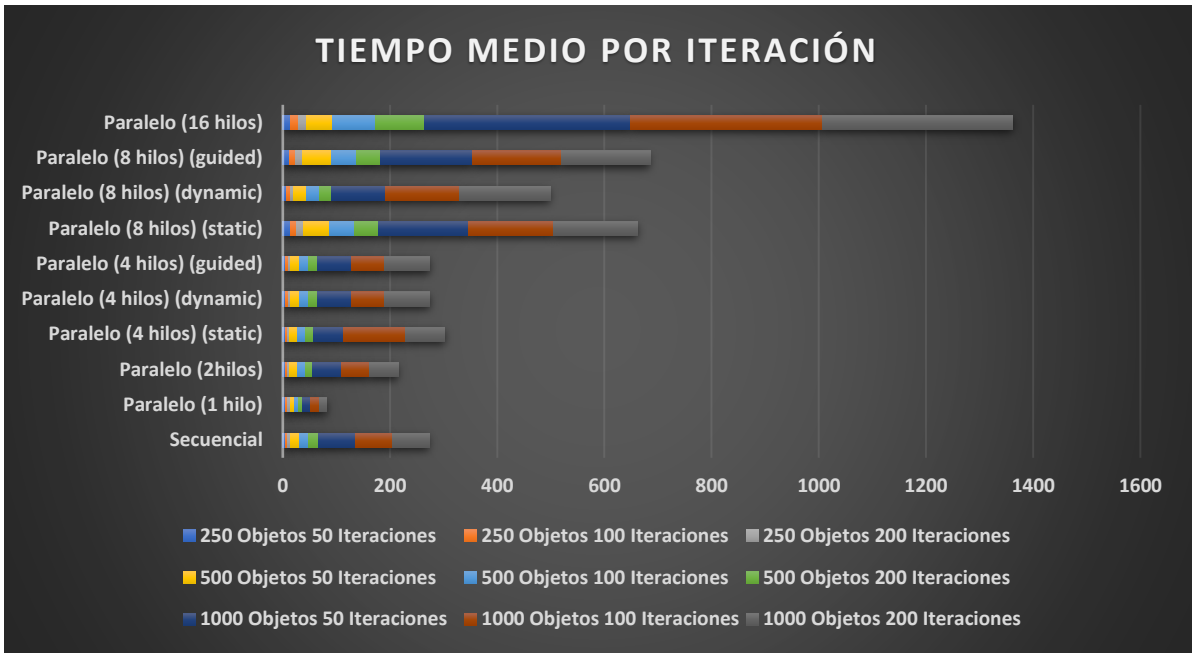
- Paralelo (2 hilos): 6913/5308 = 1,302
- Paralelo (4 hilos) (Dynamic): 6913/6224 = 1,11
- Paralelo (4 hilos) (Guided): 6913/6192 = 1,116

## GRÁFICA 1000 OBJETOS 200 ITERACIONES




El speedup que encontramos en esta gráfica es el siguiente:

- Paralelo (2 hilos): 13836/2871 = 4,819
- Paralelo (4 hilos) (Static): 13836/10602 = 1,305



Como podemos observar en el conjunto de gráficas proporcionadas, la versión paralela con un hilo, con dos hilos y con 4 hilos ejecutando una planificación estática, ejecutan más rápido que la versión secuencial, en general, exceptuando la última gráfica en el que la ejecución con 4 hilos supera ligeramente al secuencial.

A partir de los 500 objetos es recomendable utilizar las planificaciones de Dynamic y Guided, ya que como se puede comprobar en la gráfica, pueden disminuir ligeramente el tiempo de ejecución en comparación a la ejecución secuencial. El resto se ejecutan más lento que la versión secuencial, debido a que la máquina donde se han realizado las pruebas solo posee de 4 núcleos físicos, por ello con 8 y 16 núcleos el tiempo de ejecución es mayor.

Es fácil de observar los resultados generales con la gráfica situada encima, denominada “Tiempo medio de ejecución”, que comprueba que con todas las pruebas realizadas no merece la pena utilizar más de 4 hilos para el caso de nuestra máquina de 4 núcleos físicos.

Además, mientras que usando la planificación Static, disminuye el tiempo de ejecución con pocos hilos y objetos, Dynamic lo hace cuando encontramos más de 4 hilos. Static divide las tareas en fragmentos del mismo tamaño, mientras que Dynamic no utiliza un orden particular.

La diferencia entre Guided y Dynamic se encuentra en el tamaño de los fragmentos en los que se dividen las tareas, teniendo Guided menor fragmentación a la hora de ejecutar los bucles. A mayor número de hilos, parece que Dynamic funciona mejor que el resto, pero a mayor número de objetos, funciona peor. Además, observando las gráficas observamos que Guided obtiene un tiempo de ejecución similar al de Static.

## PRUEBAS FUNCIONALES

**Entrada del número de parámetros:**

Si introducimos un número diferente a 4 parámetros el sistema notificará al usuario que ha introducido un número incorrecto de datos.

```
alex19malop@alex19malop:~/Escritorio$ ./ejecutable 2 3
Tiene que introducir 4 parametros
```

Si introducimos un número negativo, o un carácter que no es un número, el programa notificará que ha introducido datos incorrectos.

```
alex19malop@alex19malop:~/Escritorio$ ./ejecutable 2 3 -2 4
Fallo en dato introducido: -2
alex19malop@alex19malop:~/Escritorio$ ./ejecutable 2 W 2 4
Fallo en dato introducido: W
```

### Comprobación de la generación de los datos

Podemos observar que nuestros datos generados son idénticos a los proporcionados con el ejecutable

| Datos de nuestro código | Datos de nasteroids2019 |
|---|---|
| 3 45 2 293 | 3 45 2 293 |
| 108.114 76.898 986.745 | 108.114 76.898 986.745 |
| 25.926 40.431 1015.558 | 25.926 40.431 1015.558 |
| 51.114 116.955 1028.133 | 51.114 116.955 1028.133 |
| 0.000 78.026 9342.348 | 0.000 78.026 9342.348 |
| 119.651 0.000 9037.973 | 119.651 0.000 9037.973 |

### Comprobación de out.txt

Comprobamos los mismos datos en el ejecutable ofrecido y en nuestros códigos en secuencial y en paralelo. Los códigos introducidos son 10 asteroides, 100 iteraciones, 3 planetas y semilla 100.

| nasteroids2019 | nasteroids-seq.cpp | nasteroids-par.cpp |
|---|---|---|
| 30.761 173.003 0.001 0.000 1034.803 | 30.761 173.003 0.001 0.000 1034.803 | 30.761 173.003 0.001 0.000 1034.803 |
| 91.446 194.003 0.001 -0.000 997.580 | 91.447 194.003 0.001 -0.000 997.580 | 91.447 194.003 0.001 -0.000 997.580 |
| 153.973 136.846 0.002 0.000 986.467 | 153.974 136.845 0.002 0.000 986.467 | 153.974 136.845 0.002 0.000 986.467 |
| 138.586 78.376 0.002 -0.001 1005.257 | 138.588 78.377 0.002 -0.001 1005.257 | 138.588 78.377 0.002 -0.001 1005.257 |
| 23.834 43.945 0.004 -0.003 1036.164 | 23.835 43.946 0.005 -0.003 1036.164 | 23.835 43.946 0.005 -0.003 1036.164 |
| 128.213 106.775 0.001 0.000 1027.558 | 128.223 106.773 0.003 -0.000 1027.558 | 128.223 106.773 0.003 -0.000 1027.558 |
| 51.020 67.108 0.003 -0.002 989.036 | 51.027 67.113 0.004 -0.001 989.036 | 51.027 67.113 0.004 -0.001 989.036 |
| 128.227 154.640 -0.000 0.001 913.372 | 128.239 154.636 0.002 0.000 913.372 | 128.239 154.636 0.002 0.000 913.372 |
| 98.135 130.147 -0.000 0.001 1045.290 | 98.148 130.145 0.002 0.000 1045.290 | 98.148 130.145 0.002 0.000 1045.290 |
| 79.610 14.582 0.015 -0.015 994.642 | 79.615 14.582 0.016 -0.015 994.642 | 79.615 14.582 0.016 -0.015 994.642 |

Podemos observar que tanto nuestro código secuencial como el paralelo dan exactamente los mismos resultados, por lo que podemos asegurar que la paralelización se ha realizado de una forma correcta sin distorsionar el resultado.

Comparando nuestro out.txt con el del ejecutable nasteroids2019 observamos una variación de centésimas, obteniendo una simulación prácticamente idéntica.

### Ejecución con flags

Ejecutando el código secuencial con los flags -std=c++14 -Wall -Wextra -Wno-deprecated -Werror -pedantic -pedantic-errors -O3 –DNDEBUG y el código paralelo con -std=c++14 -Wall -Wextra -Wno-deprecated -Werror -pedantic -pedantic-errors -fopenmp -O3 –DNDEBUG, comprobamos que nuestro código no muestra ningún warning ni error.

## CONCLUSIÓN

Como conclusión, consideramos que esta práctica nos ha sido de gran ayuda para afianzar los conocimientos adquiridos en clase, y para hacer una primera toma de contacto con un nuevo lenguaje de programación como es c++. Además, nos ha ayudado a aprender a paralelizar código secuencial en c++ con la ayuda de openMP.