Práctica 2: Computación paralela con planetas

Índice

Integrantes del equipo:

- **David Arnal García**
- **Alejandro Sánchez Melero**

1. Ejercicio 1

En el inicio del programa principal (*main*) se han declarado dos variables, las variables $t1$ y $t2$, de tipo doble (*double)*. A continuación, se imprime por pantalla el tiempo de ejecución, siendo este la resta de $t2 - t1$, así como el número de planetas, siendo este último el valor de una variable auxiliar n que toma el valor de N $(i.e. n = N)$.

```
generate_data(n, p);

t1 = omp_get_wtime();
k = process_data(n, p, &cm);
t2 = omp_get_wtime();

printf("The center of mass is in (%g, %g, %g).\n",
       cm.x, cm.y, cm.z);
printf("The planet with most neighbors is %d (%d neighbors).\n",
       k, p[k].ngb);

printf("Spent time: %f s.\n", t2 - t1);
printf("Number of planets: %d\n", n);

free(p);
return 0;
```

Ilustración 1

2. Ejercicio 2

En este ejercicio, explicaremos cada bucle por separado y, por último, el $bucle\ i\ final$ que comparten ambas versiones. También es importante mencionar que ha sido posible realizar la paralelización de todos los bucles debido a que no hay ninguna dependencia de datos entre los bucles.

2.1 Planets2i – primer *bucle i*

En este archivo se ha paralelizado el bucle externo y, por tanto, se consigue aquí la paralelización óptima. Al paralelizar el programa, se deben crear unas nuevas variables de tipo $double$ que se sustituirán por las variables del $struct\ Point3D, i.e. cm.x, cm.y, cm.z$, ya que, al paralelizar este programa, $OpenMP$

no puede acceder a las estructuras de datos, por lo que se sustituyen todos los accesos a la estructura $Point3D$ por las variables anteriormente mencionadas. El fragmento de código empleado aquí es:

```
int i;
int j;
int ngbi;
int k;
int max;
double dx;
double dy;
double dz;
double d;
double m;
Point3D cm; // center of mass
double cm_X = 0;
double cm_Y = 0;
double cm_Z = 0;
m = 0;
```

Ilustración 2

Posteriormente, se debe actualizar correctamente fuera del bucle la estructura $Point3D$ con sus valores correspondientes. El código de este fragmento es:

```
cm.x = cm_X;
cm.y = cm_Y;
cm.z = cm_Z;
```

Ilustración 3

Ahora mencionaremos las directivas para conseguir paralelización en el primer bucle que aparece en el programa. Se han incluido como privadas las variables $j, d, dx, dy, dz, ngbi$, ya que cada hilo toma sus propios valores en cada iteración. Además, las variables m, cm_X, cm_Y, cm_Z, al contribuir al estado global del programa, son de tipo reducción $(reduction)$.

```
#pragma omp parallel for private(j, dx, dy, dz, d, ngbi) reduction(+ \
                                                    : m, cm_X, cm_Y, cm_Z)
  for (i = 0; i < n; i++)
  {
```

Ilustración 4

2.2 Planets2j

En esta versión del bucle se paraleliza el bucle interno, es decir, el bucle j. Esta versión será menos óptima que la anterior y, como se podrá observar, posteriormente, en otro ejercicio, se obtendrá una mayor complejidad temporal que en el bucle externo debido a la mayor sobrecarga por la activación y desactivación de hilos. En este bucle se han declarado como privadas las variables d, dx, dy, dz, ya que, en cada iteración, cada hilo tendrá un valor propio, y no compartido.

Además, como intento de optimización de código y aseguramiento de un correcto valor para la variable $ngbi$, se ha incluido dentro de una cláusula reducción.

```
#pragma omp parallel for private(dx, dy, dz, d) reduction(+ \
                                                    : ngbi)
```

Ilustración 5

En esta versión, es importante mencionar que al realizar la instrucción $if\ (\dots)$ no se producen condiciones de carrera, ya que cada hilo mantiene su propio valor para d, pero sí que es importante incluir una directiva $atomic$ al vector p en la posición j, ya que todos los accesos y escrituras en las variables se producen de forma concurrente y j, en este caso, no es una variable privada.

```
      if (d < NGB_DST)
      {
        // Planets i and j are neighbors. Update number of neighbors
        // for both planets
        ngbi++;
#pragma omp atomic
        p[j].ngb++;
      }
    }
    p[i].ngb += ngbi;
  }
```

Ilustración 6

2.3 Ambas versiones – último *bucle i*

El último $bucle\ i$ que aparece en ambas versiones se puede paralelizar. En este bucle no se ha declarado ninguna variable como privada ni de tipo reducción, por lo que todas las variables son compartidas. Sin embargo, en este bucle sí que se produce una condición de carrera a la hora de acceder a la escritura en las variables k, max, por lo que se deberá declarar una sección crítica y duplicar la sentencia if, además de incorporar en ella el comparador $\geq$, ya que, si un valor de planetas vecinos coincide con el máximo, se deberá seleccionar el planeta de menor índice. Por este motivo, se introduce otro $if\ (\ldots)$ donde se calcula el mínimo entre los dos planetas con una función propia, llamada $minimum$, con el siguiente código:

```
int minimum(int num1, int num2)
{
  if (num1 <= num2)
  {
    return num1;
  }
  else
  {
    return num2;
  }
}
```

Ilustración 7

Siendo el código completo del último $bucle\ i$ el siguiente:

```
  // Find the planet with most neighbors
  k = 0;
  max = -1;

#pragma omp parallel for
  for (i = 0; i < n; i++)
  {
    if (p[i].ngb >= max)
    {
#pragma omp critical
      if (p[i].ngb >= max)
      {
        if (p[i].ngb == max)
        {
          k = minimum(k, i);
        }
        else
        {
          max = p[i].ngb;
          k = i;
        }
      }
    }
  }
```

Ilustración 8

3. Ejercicio 3

En el siguiente ejercicio mostraremos explícitamente distintas planificaciones para los programas. En este momento, comenzaremos analizando qué sucede si solo planificamos un solo bucle i en $planets2i$. Si, por ejemplo, se planifica el primer bucle i, **no encontraríamos diferencias significativas de rendimiento entre las distintas planificaciones.** En otro caso, **si únicamente se planifica el segundo bucle i, tampoco**

tendría por qué haber diferencias significativas en cuanto a las prestaciones, tal y como se podrá comprobar en el siguiente ejercicio.

Por otra parte, si hacemos un análisis más general, **no se encontrará ninguna planificación óptima** (es decir, mejor que las demás) **si se planifica el programa $\boldsymbol{planets2i}$ completamente**, es decir, ambos bucles i, ya que la asignación de los hilos siempre se realizará desde el inicio de los bucles y no habrá problemas de rendimiento por ello, aunque podría ser que la planificación $guided$ diera peores resultados debido a una peor gestión de hilos, ya que no son tareas cortas y sencillas, y podría no ser óptimo el tiempo de ejecución para este caso.

Sin embargo, si se planifica el programa $planets2j$ completamente, sí habrá diferencias notables, siendo **la planificación $\boldsymbol{static\ con\ chunk}$ la mejor de todas ellas**, y cuanto más grande el $chunk$, mejores prestaciones para todas las planificaciones, ya que habrá menos desactivaciones y reactivaciones de hilos. **Además, la peor planificación, en este caso, sería la planificación dinámica**, que sufriría bastante en cuanto a prestaciones debido a las continuas activaciones y desactivaciones de los hilos que se realizan con esta planificación, y destacar que, a mayor $chunk$, las prestaciones mejorarían, aunque no a un nivel óptimo, debido a que aquí la planificación dinámica no es adecuada, en general.

Todo lo anterior comentado se analizará y se podrá observar detalladamente en el posterior ejercicio.

4. Ejercicio 4

En este ejercicio, explicaremos cómo hemos lanzado los programas en $Kahan$, a continuación, analizaremos unas planificaciones parciales del programa $planets2i$ y, finalmente, analizaremos los programas $planets2i$ y $planets2j$ completamente planificados.

4.1 Ejecuciones en $\boldsymbol{Kahan}$

Antes de analizar las planificaciones gráficamente, vamos a explicar cómo lanzamos los programas en $Kahan$. Para esto, hemos creado unos archivos de extensión . sh llamados $workopenmp2.sh$ y $workopenmp3.sh$ en los que hemos ido haciendo las ejecuciones de $planets2i.c$ y $planets2j.c$, respectivamente.

En estos ficheros, se han definido las propiedades del trabajo sobre el sistema de colas, en el que elegimos trabajar con un único nodo de los seis disponibles en $Kahan$, ya que se trabaja en una máquina de memoria compartida donde todas las ejecuciones se realizan sobre el mismo nodo, y no se puede acceder a la memoria de otros nodos. Por tanto, decir, en este momento, que, se ha decidido ejecutarlo hasta con 32 hilos y no con más, ya que los programas se ejecutan solo en un nodo de $Kahan$, que tiene 2 procesadores y cada uno de ellos 16 núcleos, por lo que un nodo únicamente admite hasta 32 hilos, y no tendría sentido ejecutarlo con más hilos.

Por otra parte, junto al número de nodos elegidos, incluimos un $walltime$ con un tiempo máximo de duración para nuestras ejecuciones; En nuestro caso, hemos decidido utilizar un tiempo límite de ejecución de 10 minutos.

Además, siendo de la asignatura CPA, escogemos trabajar en la cola de dicha asignatura con el comando $-q$ y, por último, utilizamos la opción $-d$ indicando que nuestro fichero estará en el mismo directorio desde donde se ejecutará $Kahan$.

Por último, en ambos ficheros, utilizamos diferentes líneas de ejecuciones para ejecutar ambos programas con diferentes números de hilos y planificaciones. Por ejemplo, para el archivo $planets2i.c$, un fragmento de código de $workopenmp2.sh$ es:

```
#!/bin/sh
#PBS -l nodes=1,walltime=00:10:00
#PBS -q cpa
#PBS -d .

# OMP_NUM_THREADS=2 OMP_SCHEDULE=static ./planets2i
# OMP_NUM_THREADS=4 OMP_SCHEDULE=static ./planets2i
# OMP_NUM_THREADS=8 OMP_SCHEDULE=static ./planets2i
# OMP_NUM_THREADS=16 OMP_SCHEDULE=static ./planets2i
# OMP_NUM_THREADS=32 OMP_SCHEDULE=static ./planets2i

# OMP_NUM_THREADS=2 OMP_SCHEDULE=static,1 ./planets2i
# OMP_NUM_THREADS=4 OMP_SCHEDULE=static,1 ./planets2i
# OMP_NUM_THREADS=8 OMP_SCHEDULE=static,1 ./planets2i
# OMP_NUM_THREADS=16 OMP_SCHEDULE=static,1 ./planets2i
# OMP_NUM_THREADS=32 OMP_SCHEDULE=static,1 ./planets2i

# OMP_NUM_THREADS=2 OMP_SCHEDULE=static,2 ./planets2i
# OMP_NUM_THREADS=4 OMP_SCHEDULE=static,2 ./planets2i
# OMP_NUM_THREADS=8 OMP_SCHEDULE=static,2 ./planets2i
# OMP_NUM_THREADS=16 OMP_SCHEDULE=static,2 ./planets2i
# OMP_NUM_THREADS=32 OMP_SCHEDULE=static,2 ./planets2i
```

Ilustración 9

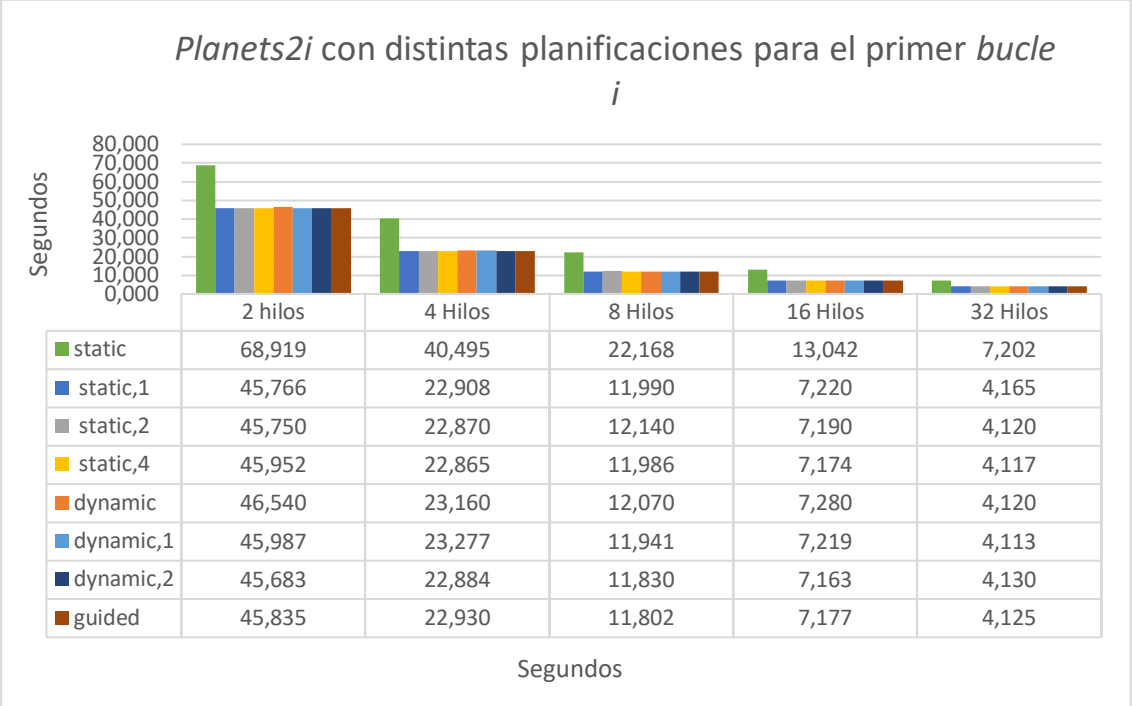
Y el archivo continuaría con ejecuciones con las planificaciones y número de hilos que deseáramos. Lo mismo sucedería para el archivo $planets2j.c$, cambiando únicamente en el archivo $.sh$ $./planets2i$ por $./planets2j$.

En la entrega, adjuntaremos los archivos $workopenmp2.sh$ y $workopenmp3.sh$.

4.2 Planificaciones parciales - planets2i con distintas planificaciones para solo un bucle i

4.2.1 Planets2i con distinta planificación para el primer bucle i

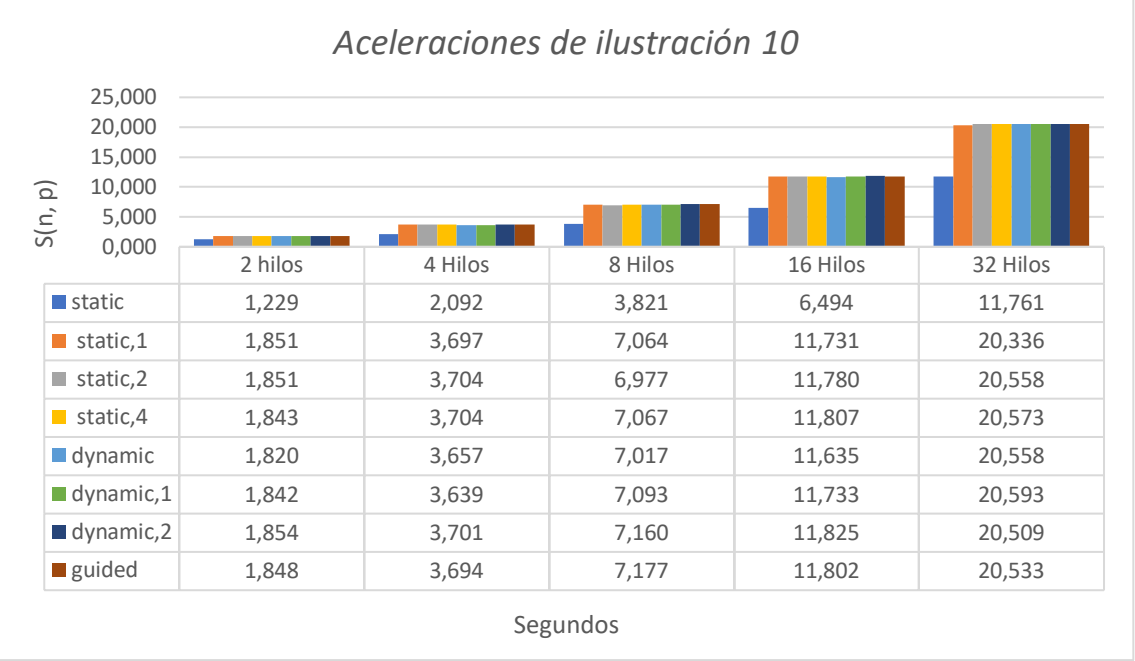
Por ejemplo, en el histograma *Planets2i con distintas planificaciones para el primer bucle i*, la planificación del segundo bucle *i* se mantiene por defecto. Debajo de cada histograma se sacarán sus correspondientes aceleraciones y eficiencias. Además, **todos los tiempos se han tomado con un número de iteraciones $N = 10.000 \cdot 8 = 80.000$ $iteraciones$.**



	2 hilos	4 Hilos	8 Hilos	16 Hilos	32 Hilos
static	68,919	40,495	22,168	13,042	7,202
static,1	45,766	22,908	11,990	7,220	4,165
static,2	45,750	22,870	12,140	7,190	4,120
static,4	45,952	22,865	11,986	7,174	4,117
dynamic	46,540	23,160	12,070	7,280	4,120
dynamic,1	45,987	23,277	11,941	7,219	4,113
dynamic,2	45,683	22,884	11,830	7,163	4,130
guided	45,835	22,930	11,802	7,177	4,125

Ilustración 10

Para tomar las aceleraciones, se ha ejecutado $planets1.c$ y ha dado como $\boldsymbol{S(n) = 84.700411\ s}$. **Se utilizará este valor para todas las aceleraciones.**

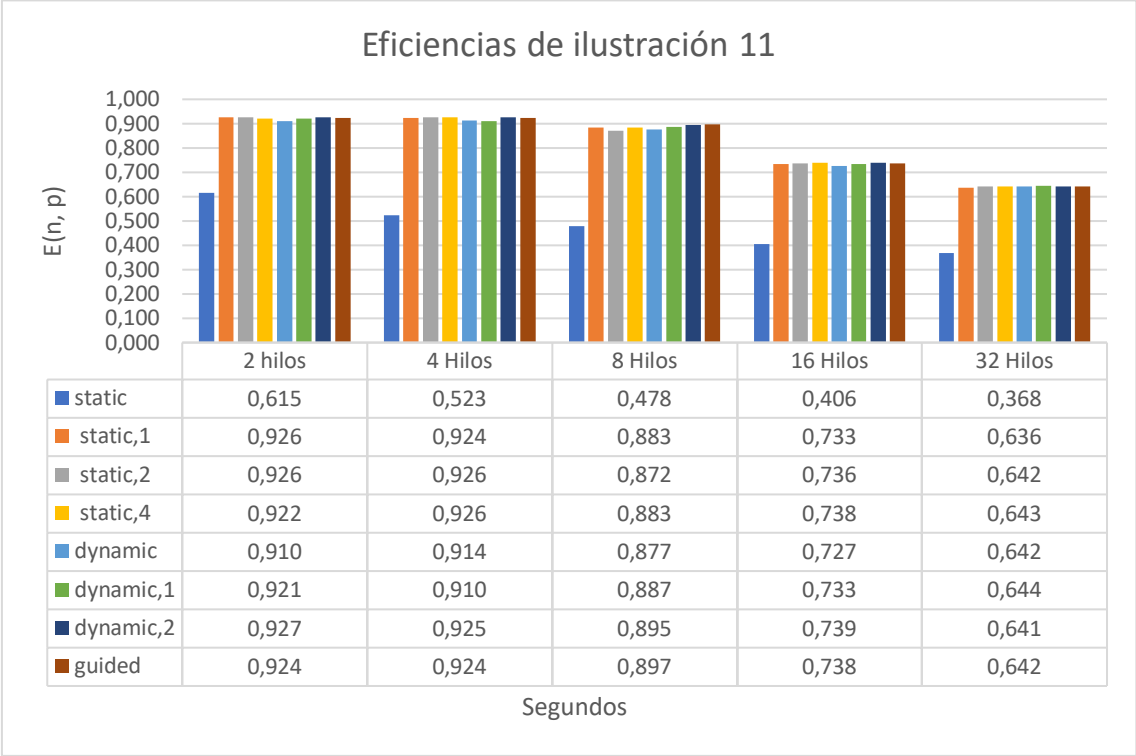


	2 hilos	4 Hilos	8 Hilos	16 Hilos	32 Hilos
static	1,229	2,092	3,821	6,494	11,761
static,1	1,851	3,697	7,064	11,731	20,336
static,2	1,851	3,704	6,977	11,780	20,558
static,4	1,843	3,704	7,067	11,807	20,573
dynamic	1,820	3,657	7,017	11,635	20,558
dynamic,1	1,842	3,639	7,093	11,733	20,593
dynamic,2	1,854	3,701	7,160	11,825	20,509
guided	1,848	3,694	7,177	11,802	20,533

Ilustración 11

Como se ha podido observar en las ejecuciones anteriores, la planificación estática $sin\ chunk$ es la que mayor tiempo ha dado, por lo que es evidente que es la que tiene

una menor aceleración en esta gráfica. Respecto a todas las planificaciones que hemos ejecutado, es la que se puede destacar, ya que las demás obtienen una aceleración muy parecida.

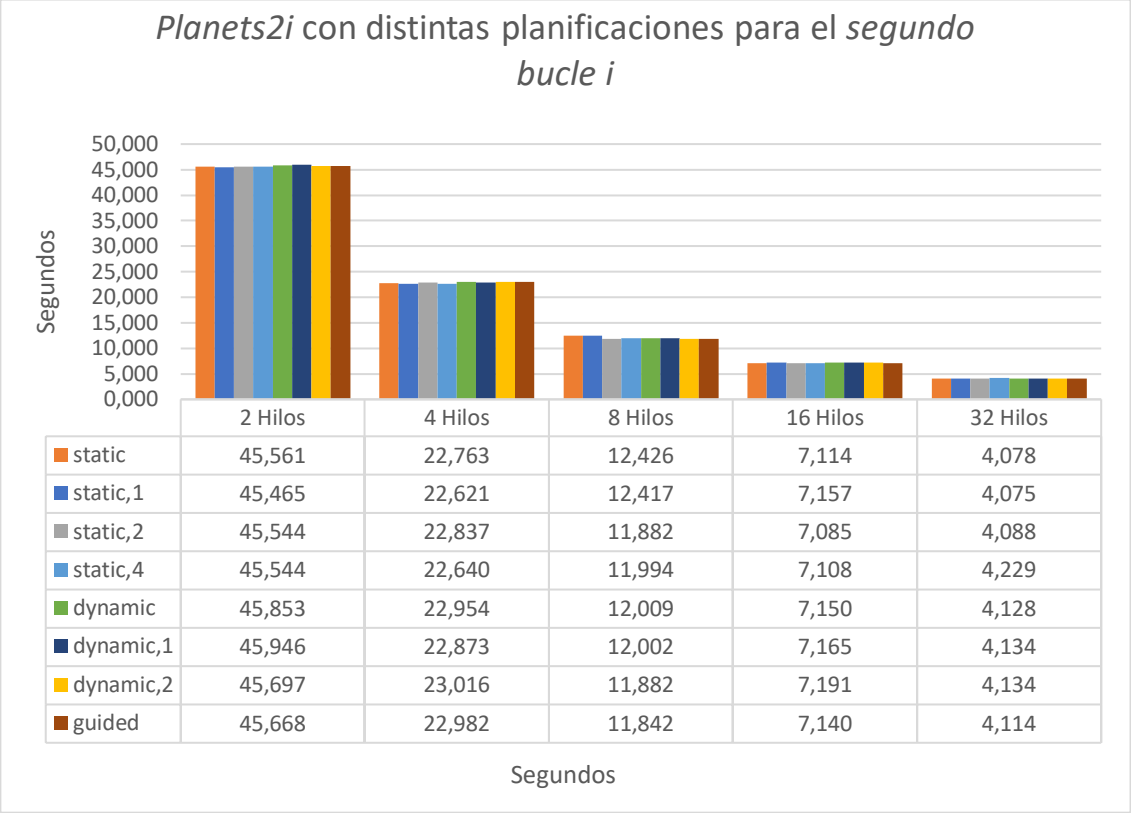


	2 hilos	4 Hilos	8 Hilos	16 Hilos	32 Hilos
static	0,615	0,523	0,478	0,406	0,368
static,1	0,926	0,924	0,883	0,733	0,636
static,2	0,926	0,926	0,872	0,736	0,642
static,4	0,922	0,926	0,883	0,738	0,643
dynamic	0,910	0,914	0,877	0,727	0,642
dynamic,1	0,921	0,910	0,887	0,733	0,644
dynamic,2	0,927	0,925	0,895	0,739	0,641
guided	0,924	0,924	0,897	0,738	0,642

Ilustración 12

Por tanto, todas las eficiencias en esta gráfica son muy parecidas, excepto para la planificación $static$ sin $chunk$, que obtiene una eficiencia menor.

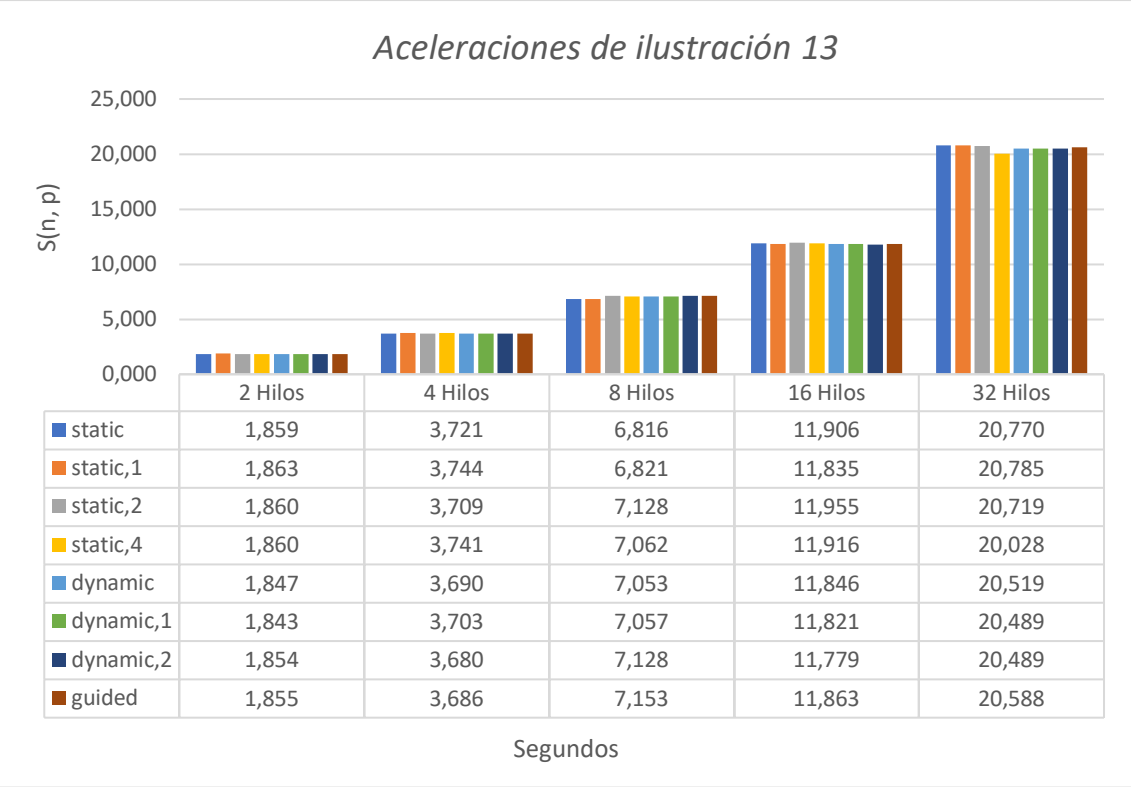
4.2.2 Planets2i con distintas planificaciones para el segundo bucle i



	2 Hilos	4 Hilos	8 Hilos	16 Hilos	32 Hilos
static	45,561	22,763	12,426	7,114	4,078
static,1	45,465	22,621	12,417	7,157	4,075
static,2	45,544	22,837	11,882	7,085	4,088
static,4	45,544	22,640	11,994	7,108	4,229
dynamic	45,853	22,954	12,009	7,150	4,128
dynamic,1	45,946	22,873	12,002	7,165	4,134
dynamic,2	45,697	23,016	11,882	7,191	4,134
guided	45,668	22,982	11,842	7,140	4,114

Ilustración 13

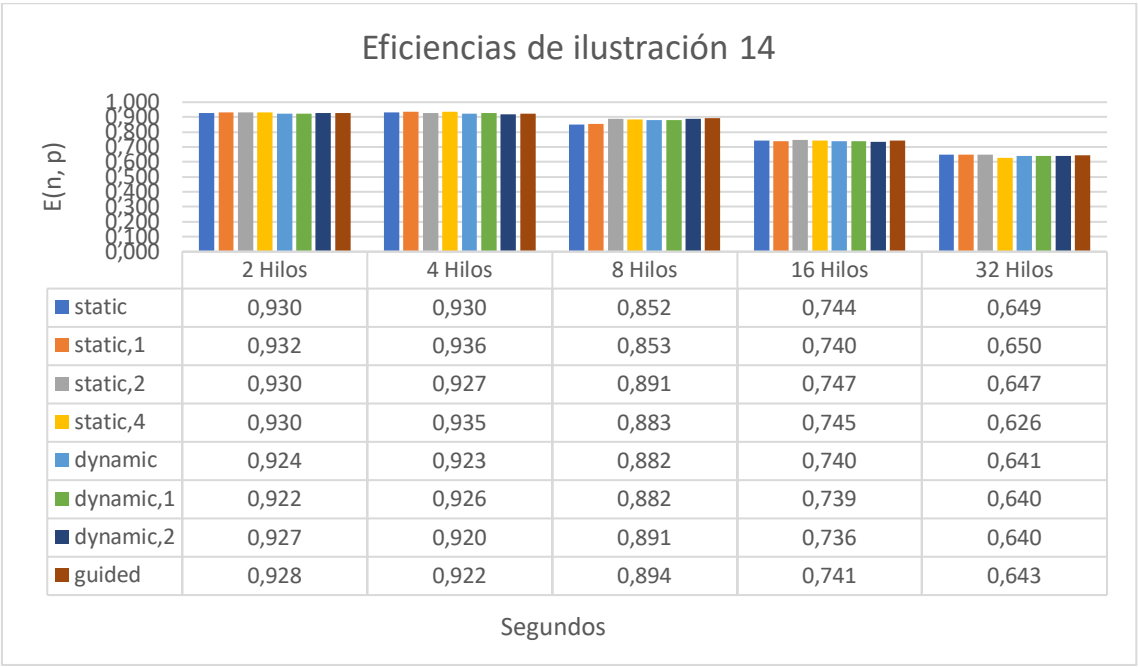
Si ejecutamos $planets2i.c$ únicamente planificando el segundo bucle i, no encontraremos ninguna diferencia significativa respecto a planificaciones.



	2 Hilos	4 Hilos	8 Hilos	16 Hilos	32 Hilos
static	1,859	3,721	6,816	11,906	20,770
static,1	1,863	3,744	6,821	11,835	20,785
static,2	1,860	3,709	7,128	11,955	20,719
static,4	1,860	3,741	7,062	11,916	20,028
dynamic	1,847	3,690	7,053	11,846	20,519
dynamic,1	1,843	3,703	7,057	11,821	20,489
dynamic,2	1,854	3,680	7,128	11,779	20,489
guided	1,855	3,686	7,153	11,863	20,588

Ilustración 14

Por esto, todas las aceleraciones serán muy parecidas.



	2 Hilos	4 Hilos	8 Hilos	16 Hilos	32 Hilos
static	0,930	0,930	0,852	0,744	0,649
static,1	0,932	0,936	0,853	0,740	0,650
static,2	0,930	0,927	0,891	0,747	0,647
static,4	0,930	0,935	0,883	0,745	0,626
dynamic	0,924	0,923	0,882	0,740	0,641
dynamic,1	0,922	0,926	0,882	0,739	0,640
dynamic,2	0,927	0,920	0,891	0,736	0,640
guided	0,928	0,922	0,894	0,741	0,643

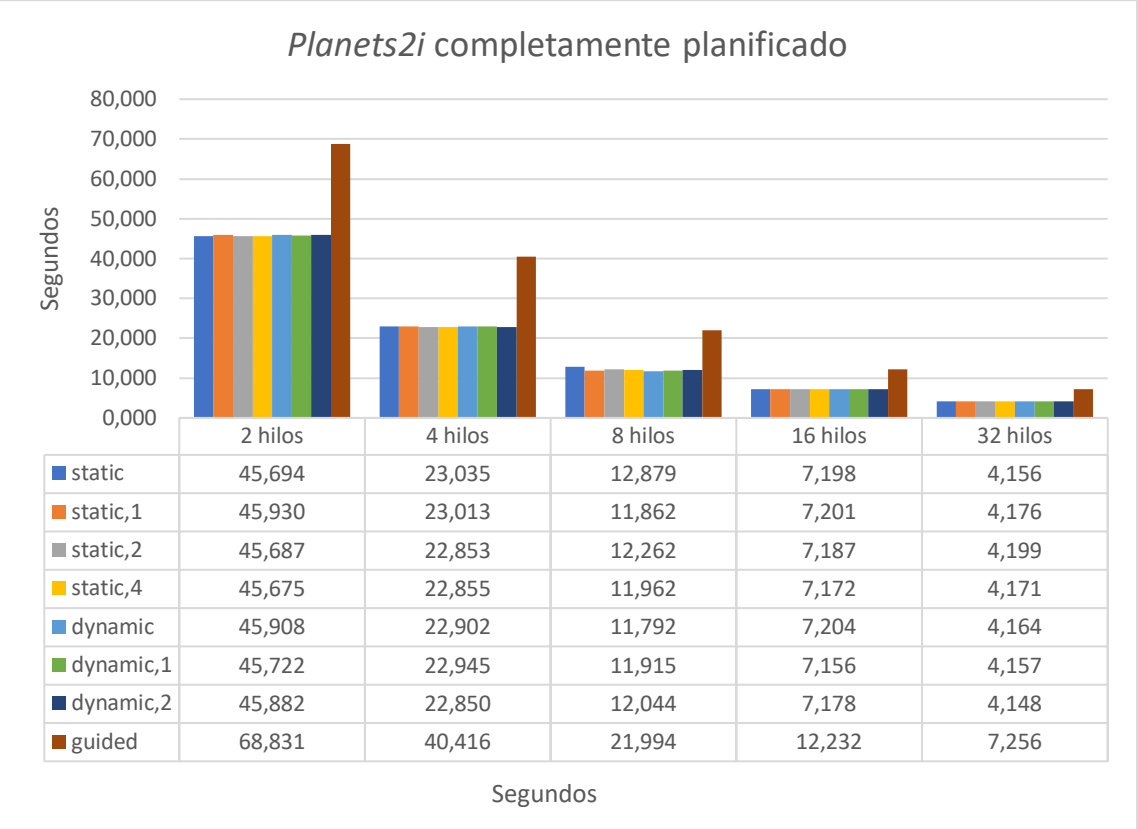
Ilustración 15

Así como sus eficiencias.

Por otra parte, se han ejecutado los programas completos con determinadas planificaciones, pudiendo observarse algunos casos particulares en los que el coste aumenta considerablemente.

4.3 Planificaciones para todo el programa

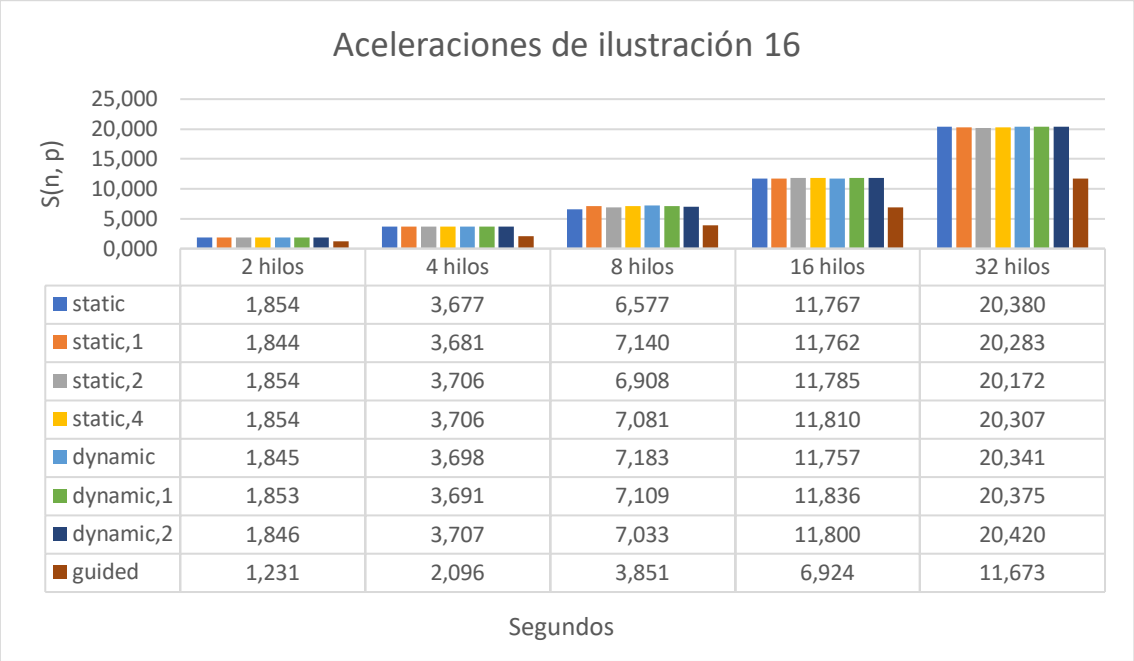
4.3.1 Planets2i completamente planificado



	2 hilos	4 hilos	8 hilos	16 hilos	32 hilos
static	45,694	23,035	12,879	7,198	4,156
static,1	45,930	23,013	11,862	7,201	4,176
static,2	45,687	22,853	12,262	7,187	4,199
static,4	45,675	22,855	11,962	7,172	4,171
dynamic	45,908	22,902	11,792	7,204	4,164
dynamic,1	45,722	22,945	11,915	7,156	4,157
dynamic,2	45,882	22,850	12,044	7,178	4,148
guided	68,831	40,416	21,994	12,232	7,256

Ilustración 16

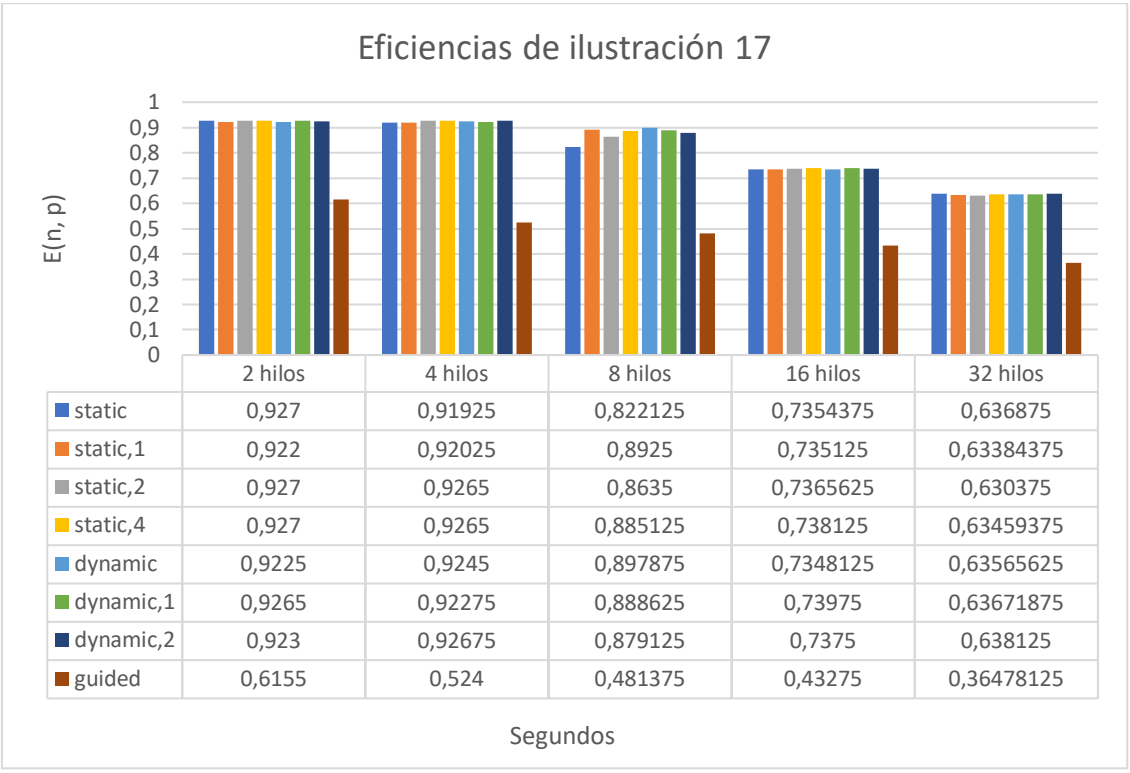
En este apartado comenzamos analizando qué sucede si se planifica el programa $planets2i.c$ completamente con una planificación determinada. En este caso, el tiempo de ejecución únicamente es mayor en el caso de la planificación $guided$, siendo mayor debido a una mala gestión de la desactivación y reactivación de los hilos. En cuanto a las demás, no hay nada importante que decir, debido a que todas ellas tienen tiempos similares.



	2 hilos	4 hilos	8 hilos	16 hilos	32 hilos
static	1,854	3,677	6,577	11,767	20,380
static,1	1,844	3,681	7,140	11,762	20,283
static,2	1,854	3,706	6,908	11,785	20,172
static,4	1,854	3,706	7,081	11,810	20,307
dynamic	1,845	3,698	7,183	11,757	20,341
dynamic,1	1,853	3,691	7,109	11,836	20,375
dynamic,2	1,846	3,707	7,033	11,800	20,420
guided	1,231	2,096	3,851	6,924	11,673

Ilustración 17

Por lo anterior mencionado, la planificación $guided$ aquí es la que tiene una menor aceleración.

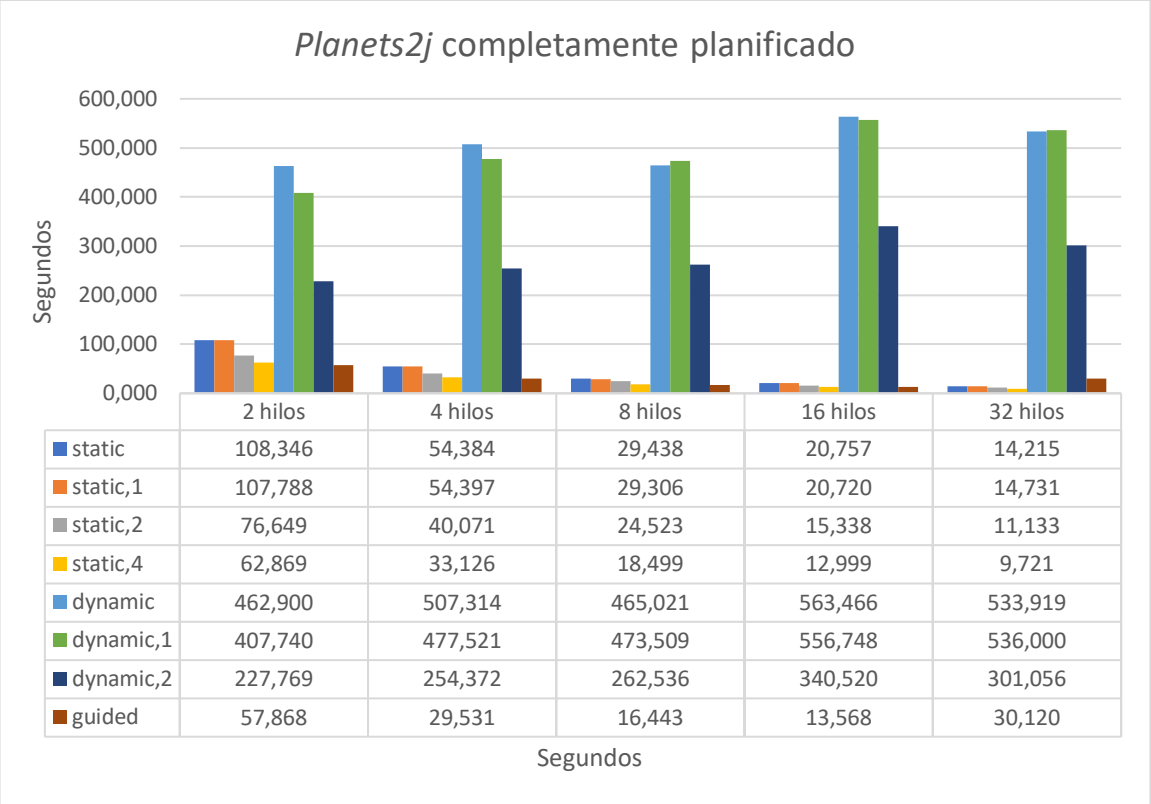


	2 hilos	4 hilos	8 hilos	16 hilos	32 hilos
static	0,927	0,91925	0,822125	0,7354375	0,636875
static,1	0,922	0,92025	0,8925	0,735125	0,63384375
static,2	0,927	0,9265	0,8635	0,7365625	0,630375
static,4	0,927	0,9265	0,885125	0,738125	0,63459375
dynamic	0,9225	0,9245	0,897875	0,7348125	0,63565625
dynamic,1	0,9265	0,92275	0,888625	0,73975	0,63671875
dynamic,2	0,923	0,92675	0,879125	0,7375	0,638125
guided	0,6155	0,524	0,481375	0,43275	0,36478125

Ilustración 18

Y, de la misma forma, es la que cuenta con una menor eficiencia.

4.3.2 Planets2j completamente planificado



	2 hilos	4 hilos	8 hilos	16 hilos	32 hilos
static	108,346	54,384	29,438	20,757	14,215
static,1	107,788	54,397	29,306	20,720	14,731
static,2	76,649	40,071	24,523	15,338	11,133
static,4	62,869	33,126	18,499	12,999	9,721
dynamic	462,900	507,314	465,021	563,466	533,919
dynamic,1	407,740	477,521	473,509	556,748	536,000
dynamic,2	227,769	254,372	262,536	340,520	301,056
guided	57,868	29,531	16,443	13,568	30,120

Ilustración 19

Aquí se puede destacar, para empezar, el mal rendimiento de la planificación dinámica, debido a las continuas desactivaciones y reactivaciones de hilos que realiza, comparado con las planificaciones estáticas, por ejemplo.

Aunque la planificación $guided$ obtendría un rendimiento parecido a la planificación estática con varios $chunks$ y con pocos hilos, se dispara cuando se ejecuta con muchos hilos, por ejemplo, con 32 hilos, por lo que no sería tan buena su gestión con las activaciones y desactivaciones de hilos.

Además, destacar aquí que cuanto mayor $chunk$ en cualquier planificación, mejor será el rendimiento, debido a que se harán menos desactivaciones y activaciones de hilos, siendo como mejor planificación, para este caso, la estática (con mayor $chunk$).

	2 hilos	4 hilos	8 hilos	16 hilos	32 hilos
static	0,782	1,557	2,877	4,081	5,958
static,1	0,786	1,557	2,890	4,088	5,750
static,2	1,105	2,114	3,454	5,522	7,608
static,4	1,347	2,557	4,579	6,516	8,713
dynamic	0,183	0,167	0,182	0,150	0,159
dynamic,1	0,208	0,177	0,179	0,152	0,158
dynamic,2	0,372	0,333	0,323	0,249	0,281
guided	1,464	2,868	5,151	6,243	2,812

Ilustración 20

Por tanto, la mayor aceleración será para el $static$ con mayor $chunk$ (*i.e.* $static$, 4), seguido del $guided$ hasta 16 hilos, y de los demás $static$ con menor $chunk$, para el caso general. Por otra parte, las planificaciones dinámicas serán la que peor rendimiento obtengan debido a lo anterior mencionado.

	2 hilos	4 hilos	8 hilos	16 hilos	32 hilos
static	0,391	0,38925	0,359625	0,2550625	0,1861875
static,1	0,393	0,38925	0,36125	0,2555	0,1796875
static,2	0,5525	0,5285	0,43175	0,345125	0,23775
static,4	0,6735	0,63925	0,572375	0,40725	0,27228125
dynamic	0,0915	0,04175	0,02275	0,009375	0,00496875
dynamic,1	0,104	0,04425	0,022375	0,0095	0,0049375
dynamic,2	0,186	0,08325	0,040375	0,0155625	0,00878125
guided	0,732	0,717	0,643875	0,3901875	0,087875

Ilustración 21

Y las eficiencias tendrán la misma proporción que las aceleraciones anteriores.

4.4 Conclusiones

Como conclusión, podemos acabar comparando la mejor planificación para $planets2i$ completamente planificado con la mejor planificación para $planets2j$ completamente planificado.

Haciendo esto, saldrían las siguientes gráficas:

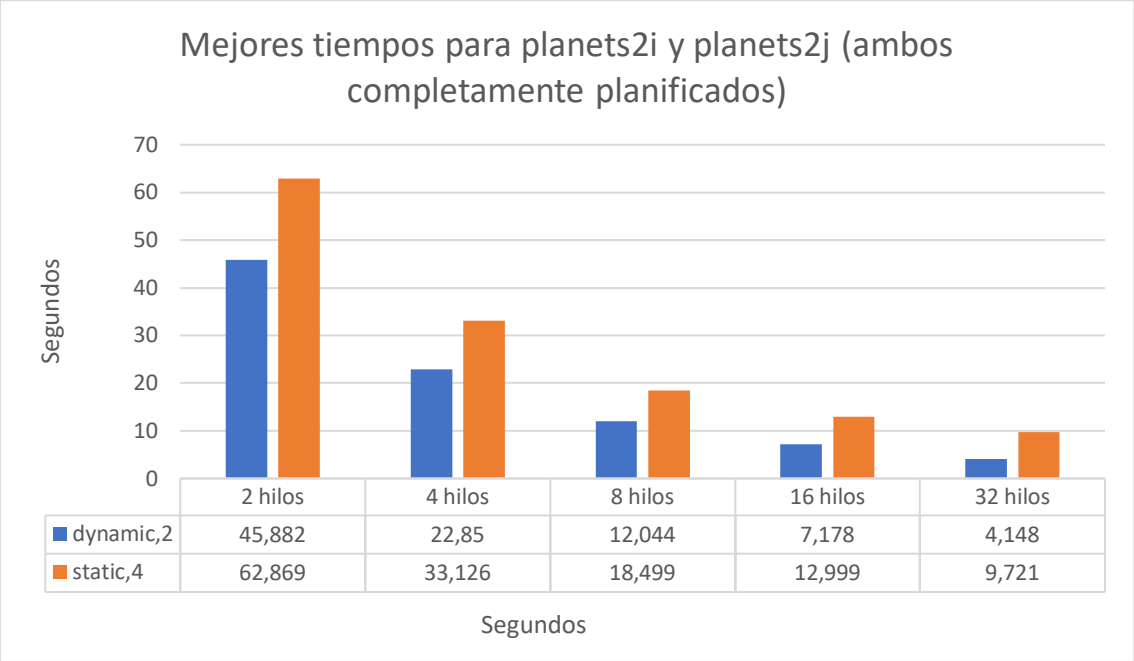


Ilustración 22

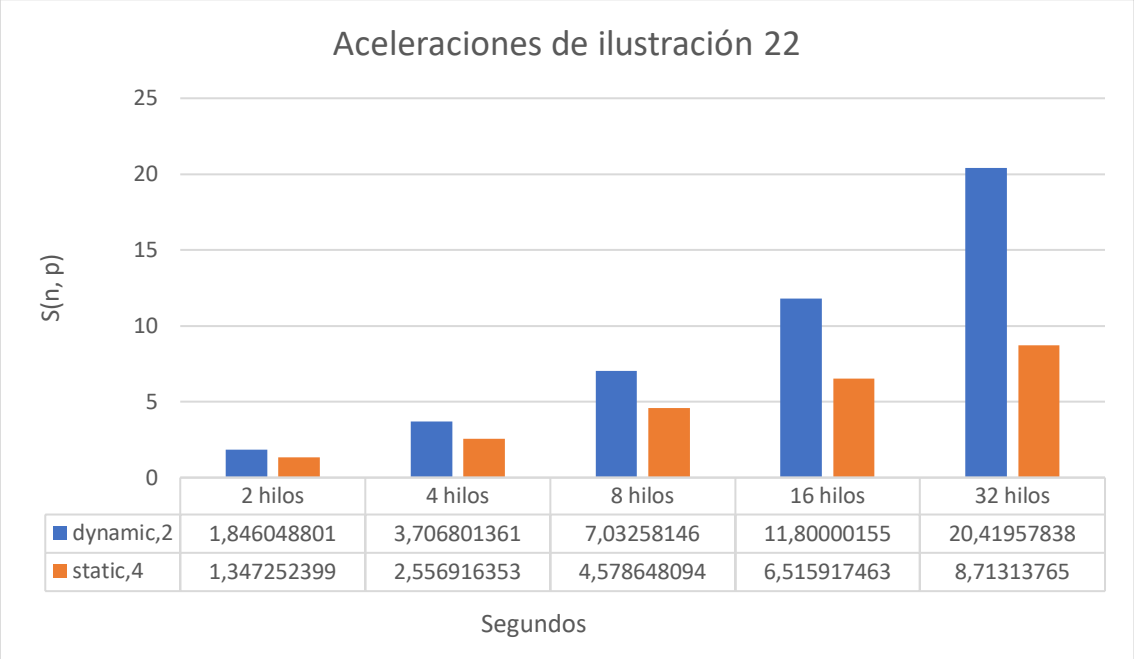
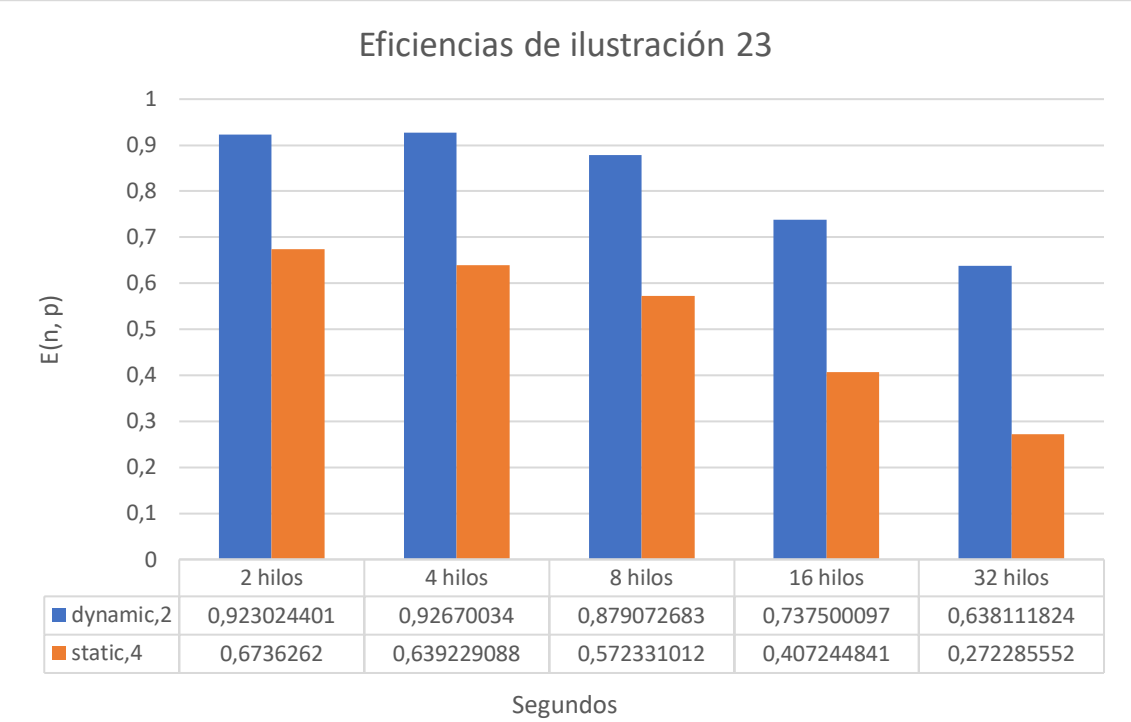


Ilustración 23



	2 hilos	4 hilos	8 hilos	16 hilos	32 hilos
dynamic,2	0,923024401	0,92670034	0,879072683	0,737500097	0,638111824
static,4	0,6736262	0,639229088	0,572331012	0,407244841	0,272285552

Ilustración 24

Para concluir, se puede observar, a través de las aceleraciones y las eficiencias, que $planets2i$ es el programa más rápido para todos los casos (en comparación con $planets2j$), teniendo sentido, ya que, cuanto más externo sea el bucle paralelizado, más eficiente debería ser y, con este último gráfico, observamos que con 2 y 4 hilos es donde más eficiencia hay, y cuantos más hilos se utilicen, más eficiencia se pierde en ambos programas.

5. Ejercicio 5

En este ejercicio hemos creado varias variables auxiliares para poder mostrar las iteraciones procesadas por cada hilo, así como el centro de masa de los planetas que ha procesado cada hilo, además de una región paralela anterior al primer bucle for.

A continuación, indicamos en la siguiente fotografía las variables auxiliares necesarias (están incluidas las variables cm_X, cm_Y, cm_Z, a pesar de no tener relevancia).

```
int nP = 0;
double cm_X = 0;
double cm_Y = 0;
double cm_Z = 0;
m = 0;

double cm_X_1 = 0;
double cm_Y_1 = 0;
double cm_Z_1 = 0;
double m1 = 0;
```

Ilustración 25

La variable $\boldsymbol{nP}$ es el contador de planetas que procesa cada hilo, las variables $\boldsymbol{cm_{X_1}}, \boldsymbol{cm_{Y_1}}, \boldsymbol{cm_{Z_1}}$, son las variables que contendrán el centro de masa en las 3 dimensiones para cada hilo, y $\boldsymbol{m1}$ la suma de las masas de todos los planetas que procesa un hilo.

Para poder desarrollar este ejercicio hemos creado una región paralela como hemos mencionado antes, la cual abarca todo el bucle y acaba antes de las asignaciones del último $bucle$ $\boldsymbol{i}$.

```
#pragma omp parallel private(nP, cm_X_1, cm_Y_1, cm_Z_1, m1)
  {
    nP = 0;
```

Ilustración 26

A continuación, mostramos cómo asignamos el valor a estas variables:

```
for (i = 0; i < n; i++)
{
  // Accumulate mass and weighted position to compute the center of mass
  m += p[i].m;
  cm_X += p[i].m * p[i].p.x;
  cm_Y += p[i].m * p[i].p.y;
  cm_Z += p[i].m * p[i].p.z;
  cm_X_1 = cm_X;
  cm_Y_1 = cm_Y;
  cm_Z_1 = cm_Z;
  m1 = m;
```

Ilustración 27

Todas las variables son privadas, ya que cada hilo mantiene un valor para cada variable, y ponemos la variable $\boldsymbol{nP}$ en cada iteración para que cada hilo comience con el valor asignado a 0, como mostramos en el siguiente fragmento de código:

```
      if (d < NGB_DST)
      {
        // Planets i and j are neighbors. Update number of neighbors
        // for both planets
        ngbi++;
        p[j].ngb++;
      }
    }
    p[i].ngb += ngbi;
    nP++;
  }

  cm.x = cm_X;
  cm.y = cm_Y;
  cm.z = cm_Z;

  // Finish computation of the center of mass
  if (m > EPS)
  {
    cm.x /= m;
    cm.y /= m;
    cm.z /= m;
  }
  if (m1 > EPS)
  {
    cm_X_1 /= m1;
    cm_Y_1 /= m1;
    cm_Z_1 /= m1;
  }
  printf("The number of planets processed by Thread %d are: %d \n", omp_get_thread_num(), nP);
  printf("The center of mass for Thread %d is: (%f, %f, %f)\n\n", omp_get_thread_num(), cm_X_1, cm_Y_1, cm_Z_1);
}
```

Ilustración 28

En cada interacción del bucle i se incrementa el contador de número de planetas para ese hilo. Luego hacemos la misma condición que para m con $m1$ y dividimos las variables que contienen el centro de masa en las tres dimensiones por la masa de los planetas. Por último, mostramos por pantalla el número de planetas procesados por cada hilo, así como el centro de masa de los planetas procesados por dicho hilo.