Memoria práctica 2

Marcos Esteve Casademunt, Jose Gómez Gadea

Noviembre 2017

Índice

1. Descripción del problema

Se desea calcular la cantidad de divisores exactos que tienen cada uno de los M primeros números naturales, con la intención de mostrar los N números enteros dentro de ese rango que tienen una mayor cantidad de divisores.

2. Definiciones

- **Speed-up** indica la ganancia de velocidad que consigue el algoritmo paralelo con respecto a un algoritmo secuencial.

$$S(n,p) = \frac{t(n)}{t(n,p)} \tag{1}$$

- **Eficiencia** mide el grado de aprovechamiento que un algoritmo paralelo hace de un computador paralelo.

$$E(n,p) = \frac{S(n,p)}{p} \tag{2}$$

3. Ejercicios

Una vez presentadas las definiciones básicas, pasamos a analizar los resultados de la práctica:

3.1. Ejercicio 1

En este ejercicio se pretende medir el tiempo del bucle principal, para ello hemos añadido las siguientes lineas.

```
t0 = omp_get_wtime();
  for ( n = 1 ; n <= M ; n++ ) {
    ...
  }
t1 = omp_get_wtime();
printf("Tiempo: %f s", t1-t0);
```

3.2. Ejercicio 2

En este ejercicio se pretende analizar si son paralelizables los tres bucles así como calcular las prestaciones.

3.2.1. Primer bucle

El primer bucle si se puede paralelizar. Para ello debemos hacer privadas las variables c, i, ini, inc que son independientes para cada iteración. Además será necesario usar una sección crítica ya que dos iteraciones distintas pueden escribir en el mismo valor del array.

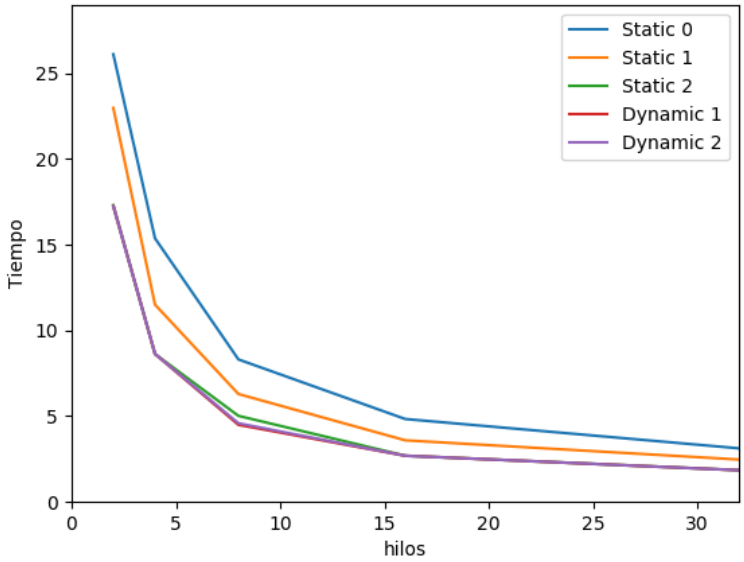
```
#pragma omp for private(c,i,ini, inc)
  for ( n = 1 ; n <= M ; n++ ) {
    c = 1; /* por el 1, que siempre es divisor */
    ...
    ...

    if ( MEJOR(n,c,N-1) ) {
      #pragma omp critical
        if ( MEJOR(n,c,N-1) ) {
        for ( i = N - 1 ; i > 0 && MEJOR(n,c,i-1) ; i-- ) {
          vc[i] = vc[i-1]; vn[i] = vn[i-1];
        }
        vc[i] = c; vn[i] = n;
      }
    }
  }
```

Pasamos por tanto a analizar los tiempos de esta paralelización:

Planificación	2	4	8	16	32
Static 0	26.098569	15.369443	8.308206	4.834699	3.126337
Static 1	22.969833	11.502033	6.292179	3.592557	2.471968
Static 2	17.304959	8.623851	5.019181	2.696642	1.853828
Dynamic 1	17.251117	8.627753	4.496865	2.693916	1.849314
Dynamic 2	17.234738	8.622996	4.586967	2.693860	1.849164

Cuadro 1: Tiempos de ejecución

Como podemos observar, una planificación dinámica reduce el tiempo de ejecución del programa. Esto se puede apreciar mejor en la gráfica mostrada a continuación.

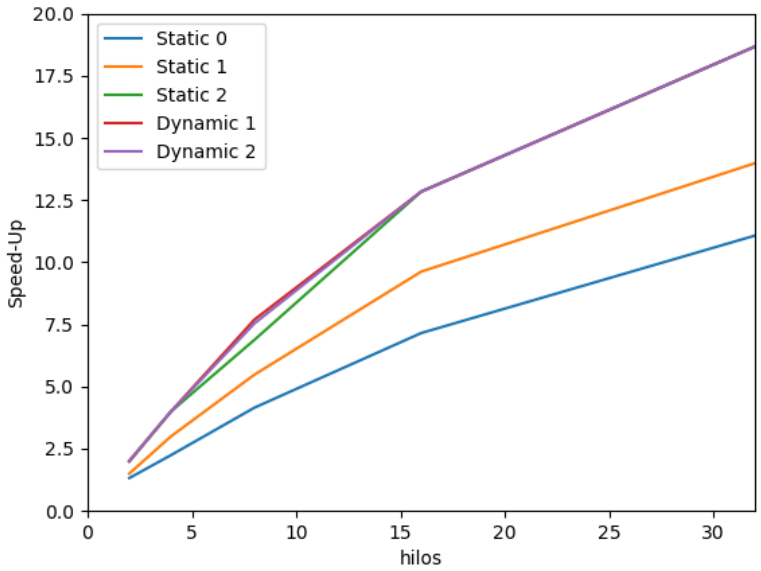


En cuanto al Speed-up (Tiempo secuencial aproximadamente 34.55s)

Planificación	2	4	8	16	32
Static 0	1.32	2.24	4.15	7.15	11.07
Static 1	1.50	3	5.48	9.62	13.98
Static 2	1.99	4	6.88	12.84	18.67
Dynamic 1	2	4	7.694	12.84	18.67
Dynamic 2	2.02	4	7.54	12.84	18.67

Cuadro 2: Speed-Up

Las planificaciones Dynamic junto con static con chunck 2 obtienen mejores Speed-Ups en comparación con el resto de planificaciones. Esto se puede observar en la siguiente gráfica.

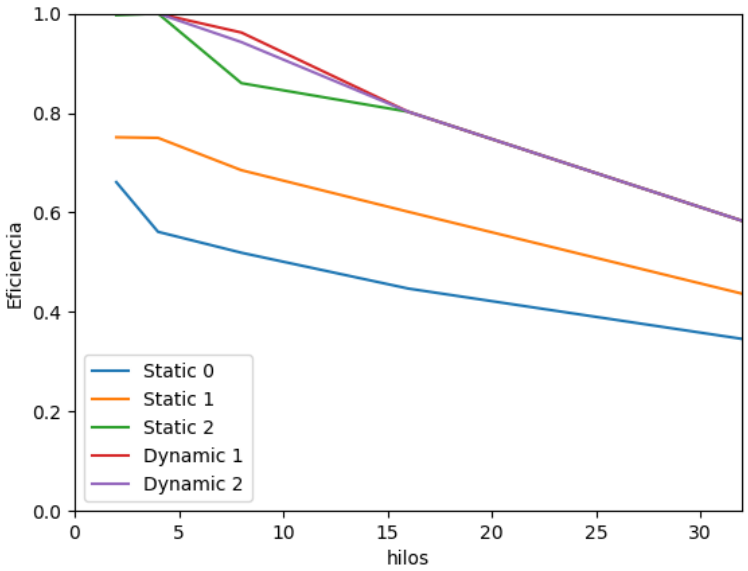


Por último analizando la eficiencia...

Planificación	2	4	8	16	32
Static 0	0.660	0.561	0.519	0.447	0.345
Static 1	0.751	0.75	0.685	0.601	0.436
Static 2	0.996	1	0.86	0.802	0.583
Dynamic 1	1	1	0.961	0.802	0.583
Dynamic 2	1	1	0.942	0.802	0.583

Cuadro 3: Eficiencia

Como podemos observar, la máquina paralela está mejor aprovechada con 2, 4 hilos y con las planificaciones Static con chunck 2 y Dynamic. Esto se puede observar mejor en la siguiente gráfica.



3.2.2. Segundo bucle

El segundo bucle si se puede paralelizar. Para ello debemos hacer una reducción sumando de c.

```
#pragma omp parallel for reduction(+:c)
for ( i = ini ; i <= n ; i += inc ) {
        if ( n % i == 0 ) c++;
}
```

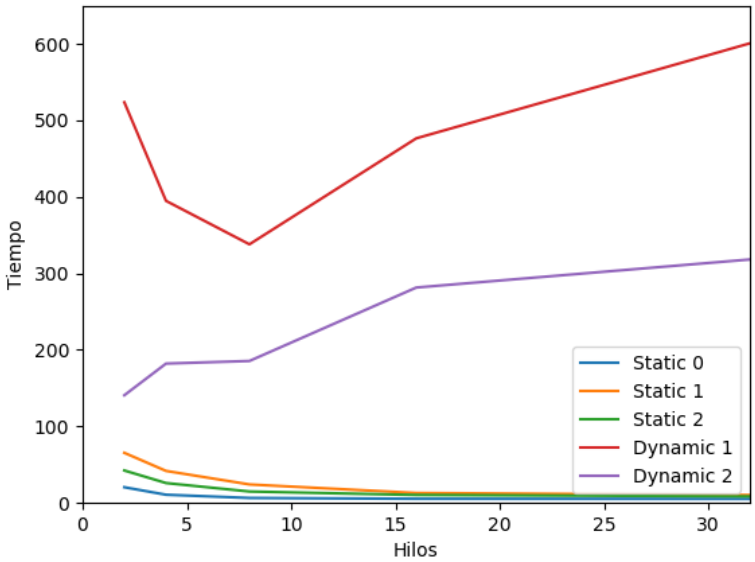
Pasamos a analizar los tiempos de ejecución al paralelizar este segundo bucle:

Planificación	2	4	8	16	32
Static 0	20.530381	10.729977	6.503967	5.394774	5.210012
Static 1	65.497120	41.851255	24.181914	13.017211	10.364424
Static 2	42.421072	26.000943	14.932108	10.405176	8.495147
Dynamic 1	523.593595	394.736502	337.955795	476.433302	600.608951
Dynamic 2	140.707958	182.115155	185.497058	281.455687	318.156567

Cuadro 4: Tiempos de ejecución

Como podemos observar, los tiempos con planificaciones dinámicas son significativamente mayores que el tiempo de planificaciones estáticas. Esto es debido a que las planificaciones dinámicas tienen un sobre coste mayor a las planificaciones estáticas. Además, esta segunda paralelización será más

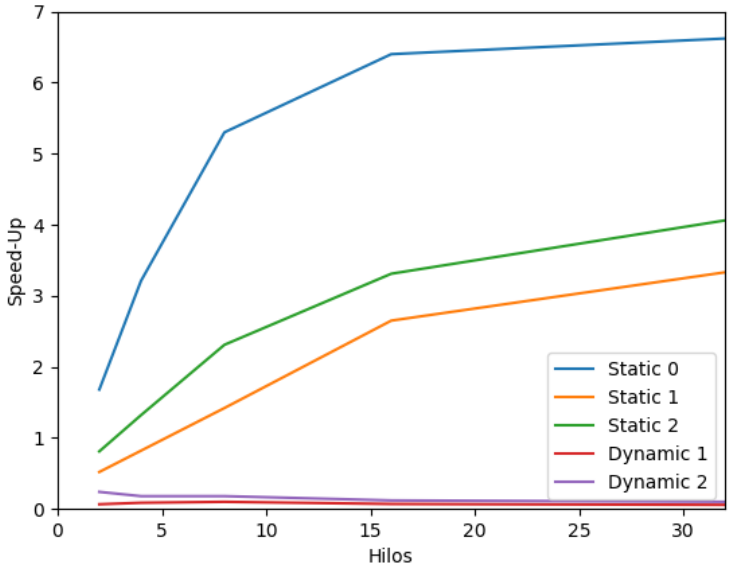
ineficiente que la paralelización del primer bucle ya que los hilos se activarán y desactivarán tantas veces como iteraciones haga el bucle principal, y esto hará que se añada cierto sobre coste que repercutirá en el tiempo de ejecución. Algunas de estas conclusiones se pueden observar visualmente en la siguiente gráfica.



En cuanto al Speed-up:

Planificación	2	4	8	16	32
Static 0	1.68	3.21	5.3	6.4	6.62
Static 1	0.52	0.82	1.42	2.65	3.33
Static 2	0.81	1.32	2.31	3.31	4.06
Dynamic 1	0.0658	0.087	0.1	0.07	0.0575
Dynamic 2	0.24	0.18	0.18	0.12	0.10

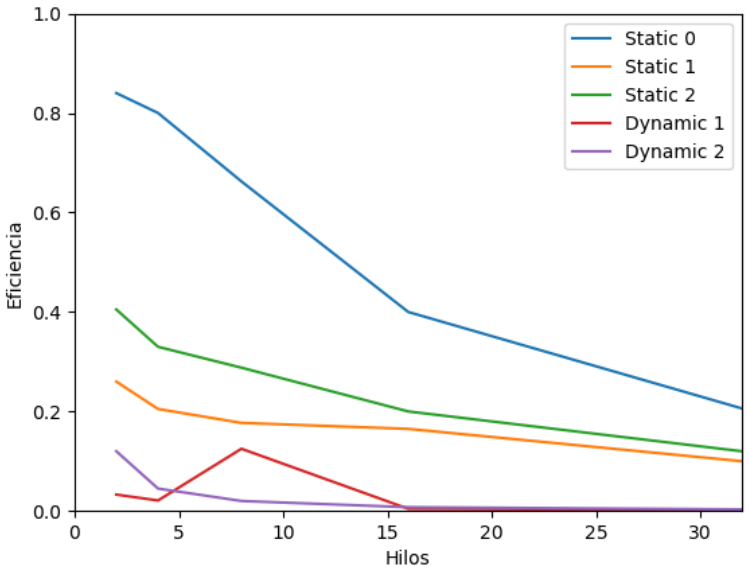
Cuadro 5: Speed-Up



Por último, analizando la eficiencia...

Planificación	2	4	8	16	32
Static 0	0.84	0.80	0.6625	0.4	0.206
Static 1	0.26	0.205	0.177	0.165	0.10
Static 2	0.405	0.33	0.288	0,20	0.12
Dynamic 1	0.0329	0.021	0.125	0.0043	0.0017
Dynamic 2	0.12	0.045	0.02	0.0075	0.0031

Cuadro 6: Eficiencia



3.2.3. Tercer bucle

El tercer bucle no se puede paralelizar. Esto es debido a que existe una dependencia entre sus iteraciones ya que, en la iteración i, se lee el valor de la iteración i-1.

```
for ( i = N - 1 ; i > 0 && MEJOR(n,c,i-1) ; i-- ) {
        vc[i] = vc[i-1]; vn[i] = vn[i-1];
}
```

3.3. Ejercicio 3

En este ejercicio se pretende que cada hilo muestre la cantidad total de divisores. Para ello, creamos una variable privada para toda la región paralela. Además, en el bucle de contar divisores incrementamos la variable y por último mostramos por cada hilo la cantidad de divisores que ha calculado.

```
int nDiv = 0;
  #pragma omp parallel private(nDiv)
  {
    #pragma omp for private(c, ini, inc, i)
    for ( n = 1 ; n <= M ; n++ ) {
      ...
      ...
       /* Contar divisores */
       for ( i = ini ; i <= n ; i += inc ) {
         if ( n %i == 0 ) {
           c++;
```

```
            nDiv++;
          }
        }
        if ( MEJOR(n,c,N-1) ) {
          #pragma omp critical
          {
            ...
            ...
        }
      }
      printf("Hilo %d: %d divisores.\n",omp_get_thread_num(),nDiv)
          ;
    }
```

3.4. Ejercicio 4

En este último ejercicio se pretende paralelizar de forma manual el bucle que funcione mejor(el más externo en nuestro caso). La parelización será idéntica al primer bucle, pero cambiando el parallel for por lo siguiente:

```
#pragma parallel  private(c, i, ini, inc, yo)
  {
    yo = omp_get_thread_num();
    numhilos = omp_get_num_threads();
    for ( n = yo +1; n <= M ; n+=numhilos ) {
      ...
      ...
      ...
       if ( MEJOR(n,c,N-1) ) {
         #pragma omp critical

           if ( MEJOR(n,c,N-1) ) {
           for ( i = N - 1 ; i > 0 && MEJOR(n,c,i-1) ; i-- ) {
             vc[i] = vc[i-1]; vn[i] = vn[i-1];
           }
           vc[i] = c; vn[i] = n;
         }
       }
    }
}
}
```

4. Conclusiones

Como conclusiones, nos gustaría recalcar:

- Como se ha podido observar, paralelizar el bucle más externo es más eficiente que paralelizar el bucle interno. Esto es debido a que paralelizar el bucle interno añade al tiempo un sobrecoste de activación y desactivación de los hilos.

- También se ha podido observar que en este caso las planificaciones Dynamic y Static con chunk 2 obtienen mejores tiempos que las demás planificaciones. Esto es debido a que todas las iteraciones no tienen la misma carga y por tanto, un reparto dinámico repartirá mejor la carga entre los distintos hilos.
- Por último, nos gustaría destacar que esta práctica nos ha ayudado a confirmar algunos conocimientos explicados en teoría (medida de prestaciones, planificaciones estáticas y dinámicas...).