Práctica 4 - ARQO

Víctor de Juan Sanz - Guillermo Julián Moreno

Diciembre 2013

Ejercicio 0

Características de las CPU de los ordenadores del laboratorio:

Número de cores físicos 2.

Número de cores virtuales 4.

Hyperthreading Sí.

Frecuencia $1.2\,\mathrm{GHz}$.

Ejercicio 1

¿Cómo se comporta OpenMP cuando declaramos una variable privada? OpenMP declara instancias privadas de la variable por cada hilo que genera, de tal forma que cada hilo tiene su propia variable a cuyo valor no acceden el resto de hilos.

¿Qué ocurre con el valor de una variable privada al comenzar a ejecutarse la región paralela? Se inicializa al valor que tenía antes de comenzar la región paralela.

¿Qué ocurre con el valor de una variable privada al finalizar la región paralela? La variable privada mantiene el valor que tenía antes de ejecutarse la región paralela.

¿Ocurre lo mismo con las variables públicas? No, las variables públicas son accesibles por todos los hilos y por lo tanto cualquier modificación que hagan los hilos se mantendrá al acabar la región paralela.

Ejercicio 2

El resultado es correcto en la versión en serie y en la versión 2 de la paralela. La versión 1 paralela no funciona porque al final no suma los resultados parciales de cada uno de los hilos. Además, en la versión 2 puede haber errores de redondeo en los números de coma flotante debido a que suma en un orden distinto al de la versión serie.

Para obtener mejores mediciones, hemos multiplicado por 100 los tamaños de matrices a ejecutar, de tal forma que las pequeñas variaciones que pueda introducir el SO no afecten tanto a la medición. También ejecutamos varias veces el bucle y obtenemos la media del tiempo de ejecución para evitar la aparición de medidas anómalas.

Preguntas

En términos del tamaño de los vectores, ¿compensa siempre lanzar hilos para realizar el trabajo en paralelo, o hay casos en los que no? No siempre compensa

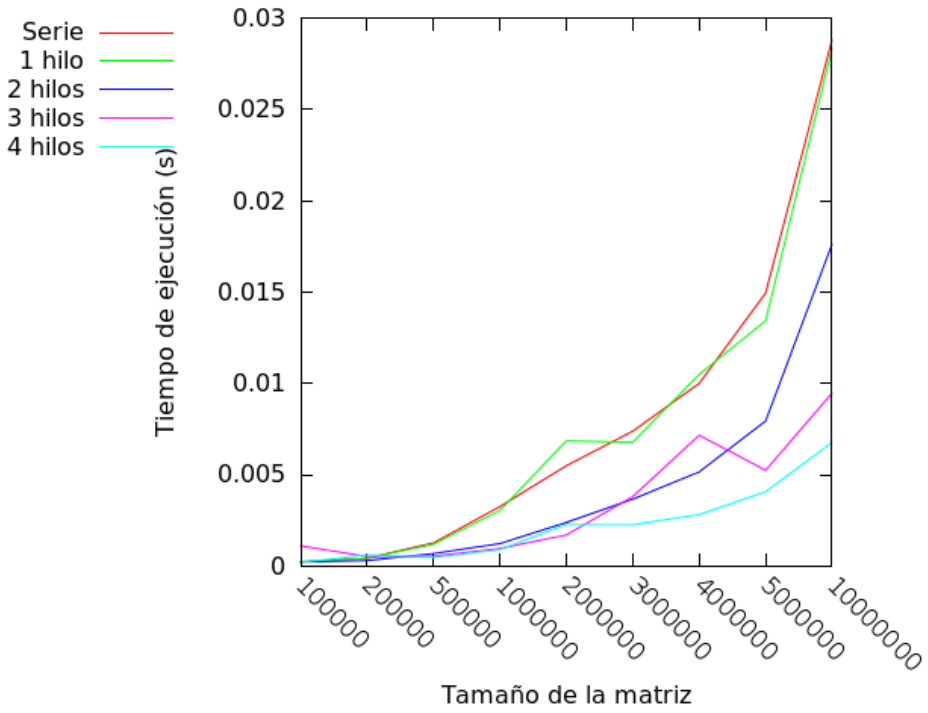

Figura 1: Tiempos de ejecución

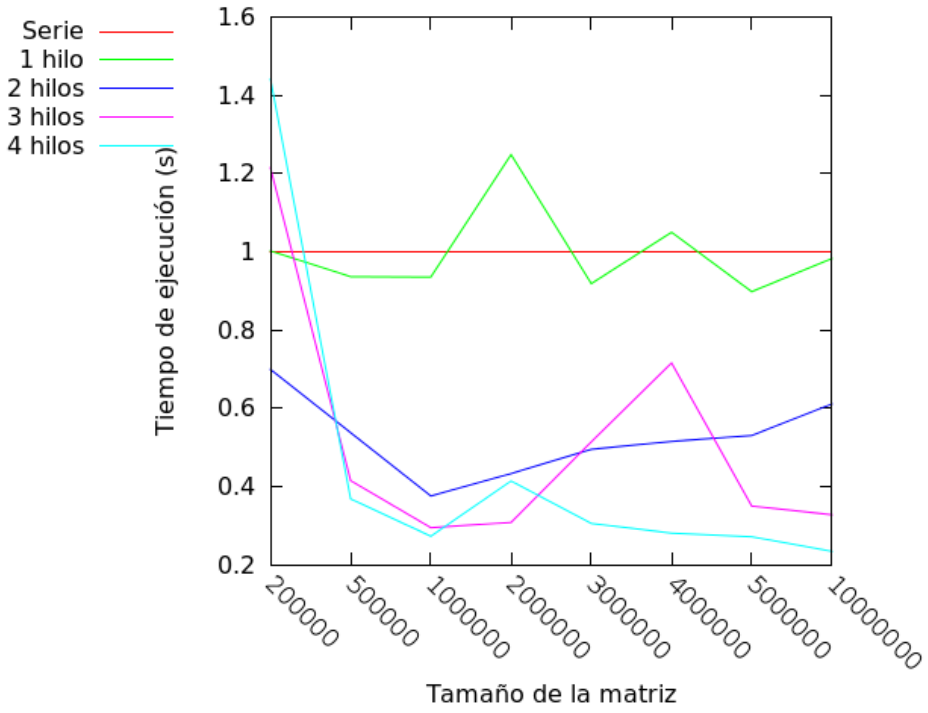

Figura 2: Aceleración

lanzar varios hilos para ejecutar el trabajo. Cuando el tamaño del vector es pequeño, invertimos más tiempo en crear los hilos y lanzarlos del que ganamos por ejecutar la operación en paralelo. En estos casos, es más rápido no lanzar hilos porque nos evitamos la carga de creación de cada hilo.

¿Se mejora el rendimiento al aumentar el número de hilos? ¿A qué se debe este efecto? Se aumenta el rendimiento hasta cierto punto. Primero tenemos que tener en cuenta el efecto anterior: la mejora obtenida por lanzar más hilos puede anularse por el tiempo que tardamos en crearlos cuando el tamaño del vector es pequeño.

Por otra parte, la mejora deja de notarse cuando tenemos más hilos que núcleos virtuales tiene nuestro ordenador. Mientras mantengamos menos hilos que núcleos virtuales, cada hilo podrá ejecutarse al mismo tiempo "virtual"[1]. Sin embargo, cuando aumentamos el número de hilos por encima del número de núcleos virtuales, los hilos compiten entre ellos por tiempo de procesador, no se ejecutan al mismo tiempo, se fuerzan más cambios de contexto y el rendimiento puede disminuir.

Ejercicio 3

Paralelización	1 hilo	2 hilos	3 hilos	4 hilos
0	3.152073	3.158909	3.190186	3.134583
1	4.258791	3.856951	3.022815	3.745091
2	3.852682	2.392473	1.896834	2.007934
3	4.367736	2.252173	1.563786	1.227189

Cuadro 1: Tiempos (en segundos) de las versiones de la multiplicación de matrices.

Paralelización	1 hilo	2 hilos	3 hilos	4 hilos
0	1	0.997836	0.988053	1.005580
1	0.740133	0.817245	1.042761	0.841655
2	0.818150	1.317496	1.661755	1.569809
3	0.721672	1.399570	2.015668	2.568531

Cuadro 2: Aceleración de las versiones de la multiplicación de matrices.

La versión más rápida es la paralelización del bucle 3, el más externo, con cuatro hilos. En esta versión es en la que menos hilos se van a crear (en las otras versiones se crean de nuevo los hilos cada vez que se ejecuta el bucle exterior), y además al usar 4 hilos aprovechamos al máximo los cuatro núcleos virtuales del procesador.

La peor versión es la de un hilo en el bucle más externo, aunque podríamos considerar las diferencias de tiempo mínimas y asumir que, en general, las versiones paralelas de un hilo son más lentas ya que se ejecuta la instrucción `#pragma omp`, que repercute en el tiempo de ejecución, pero al no lanzar más hilos no hay ninguna ganancia de tiempo.

[1] El *hyperthreading* crea dos núcleos virtuales para que un único procesador ejecute dos hilos a la vez, así que no se ejecutan al mismo tiempo realmente

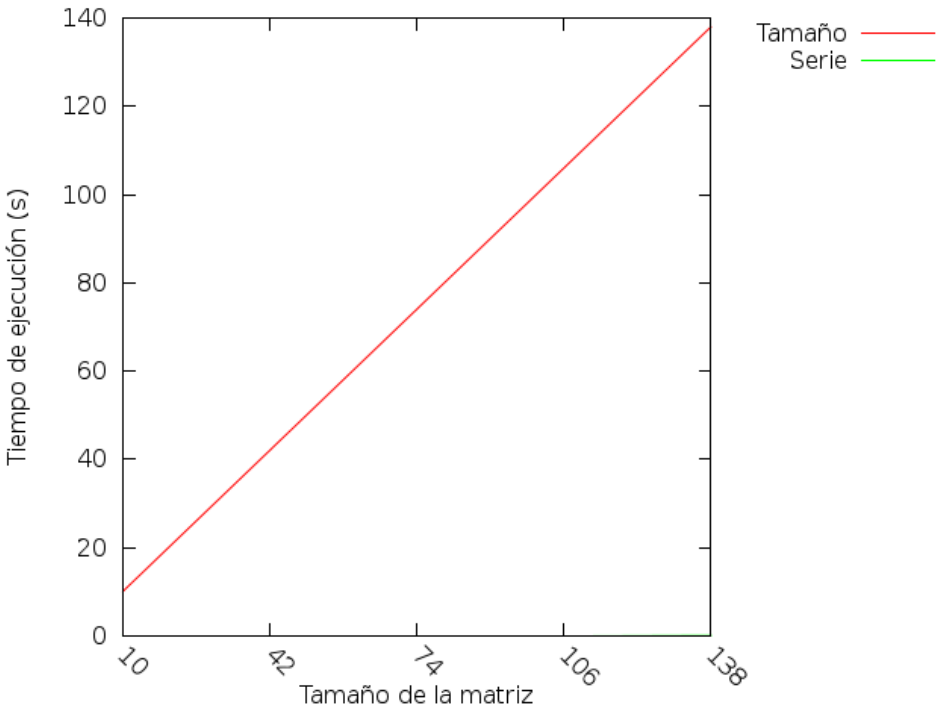

Figura 3: Tiempos de ejecución de las multiplicaciones de matrices.

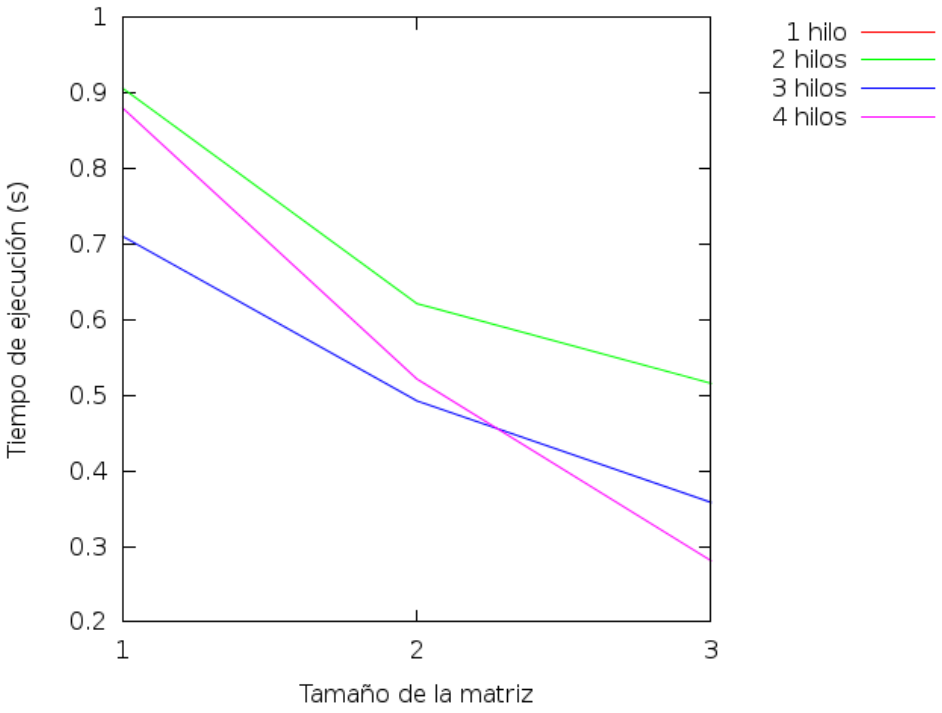

Figura 4: Aceleración de las multiplicaciones de matrices.

Ignorando las versiones de un hilo, la peor versión es la paralelización del bucle más interno (1) con dos hilos. En este caso, se crean hilos con muchísima frecuencia (cada vez que se ejecuta el segundo bucle), lo que repercute mucho en el tiempo de ejecución. Además, al usar sólo dos hilos no aprovechamos todo el potencial de la CPU y no ganamos todo lo que podríamos ganar con más hilos.

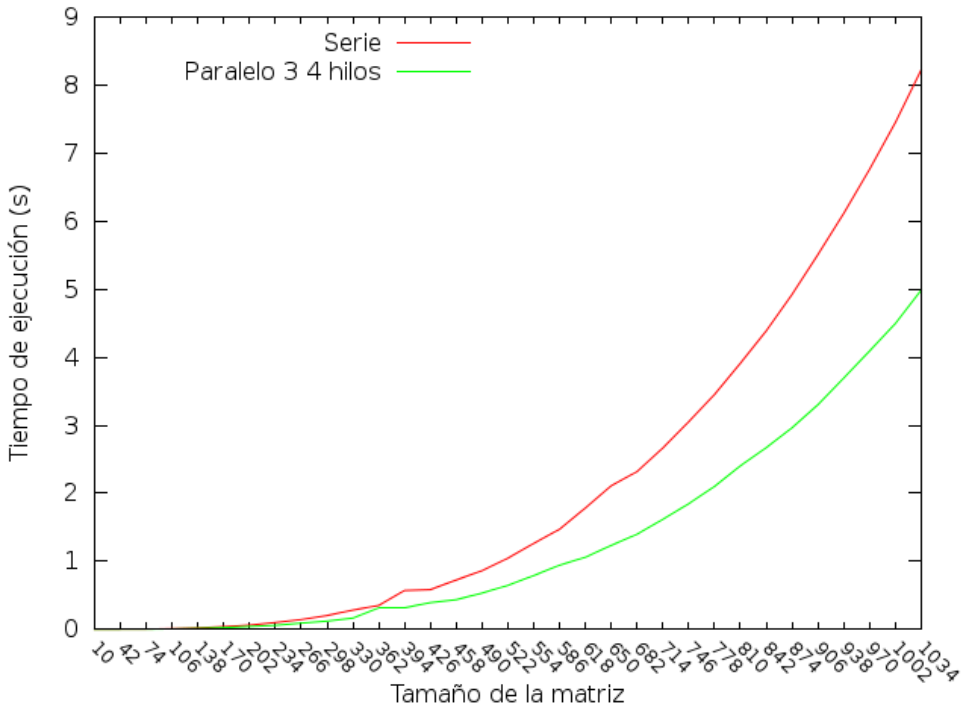

Figura 5: Comparación entre versión serie y la paralelización del bucle más externo (3) con 4 hilos

En los primeros valores es más rápido *serie* que la versión paralela debido a lo que hemos comentado antes de la carga provocada por la creación de cada hilo. Después, la versión paralela tarda menos.

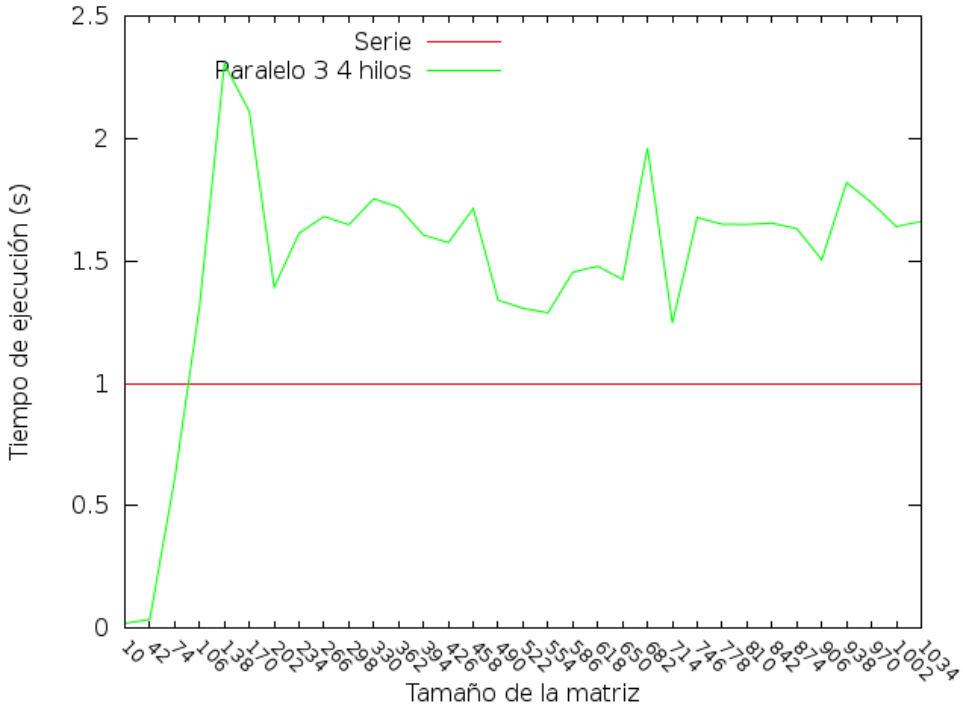

Figura 6: Aceleración de la versión serie frente a la paralelización del bucle más externo (3) con 4 hilos