Pareja 42: Carlos Gómez-Lobo Hernaiz, Rodrigo Lardiés Guillén

Practica 4: Explotar el Potencial de las Arquitecturas Modernas

Introducción

En esta práctica aprenderemos a usar la arquitectura *multicore*. Para ello hemos realizado varios ejercicios en los que se paralelizan algunas partes y hemos analizado el tiempo que tarda cada programa, así como su aceleración.

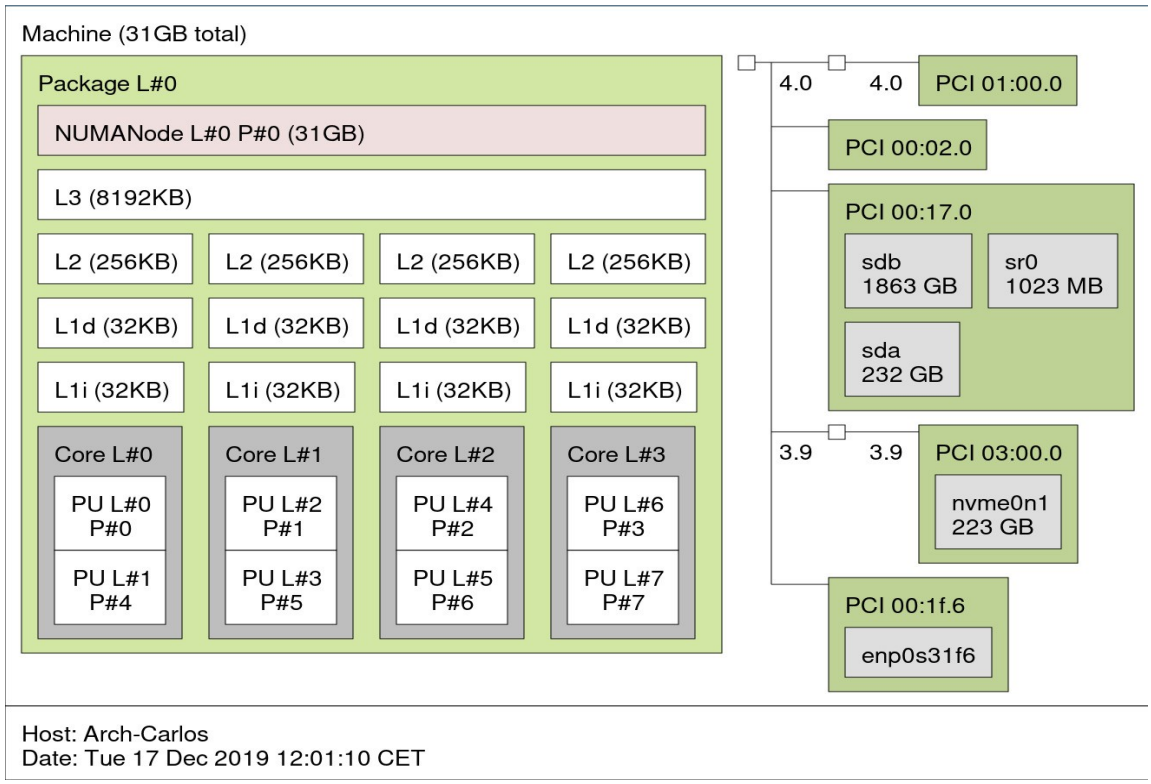
Para diferenciar las pruebas de cada pareja, se ha establecido un número P que se usará a lo largo de la práctica. En nuestro caso P=3 (pues somos la pareja 42, P ≡ 42mod8 + 1).

Ejercicio 0 (cpuinfo.dat)

Los resultados de este ejercicio corresponden al ordenador en el que se han realizado las pruebas (que no es el de los laboratorios, ni el cluster). En el fichero *cpuinfo.dat* se muestra la salida al ejecutar el comando:

cat /proc/cpuinfo

Observamos que el ordenador posee cuatro núcleos físicos y ocho virtuales, porque tiene habilitada la opción de *hyperthreading*. Esto se aprecia con más claridad con la siguiente imagen obtenida de la ejecución del comando lstopo:



Pareja 42: Carlos Gómez-Lobo Hernaiz, Rodrigo Lardiés Guillén

Ejercicio 1

En este ejercicio ejecutaremos los programas facilitados para aprender el uso de la librería *OpenMP*, y poder responder a las siguientes preguntas:

Pregunta 1) : ¿Se pueden lanzar más *threads* que cores tenga el sistema? ¿Tiene sentido hacerlo?

Si se pueden lanzar mas hilos que cores, pero no tiene sentido hacerlo. Si nuestro ordenador no tiene la opción de *hyperthreading* habilitada, lo más óptimo será lanzar tantos hilos como cores físicos tenga porque será el máximo de hilos que se puedan ejecutar al mismo tiempo. Si se envian más, no todos los hilos se ejecutarán de forma paralela. Si la opción de *hyperthreading* está habilitada, lo más óptimo será lanzar tantos hilos como cores virtuales tengamos, ya que es el número máximo de hilos que puede ejecutar de forma paralela.

Pregunta 2) : ¿Cuántos *threads* debería utilizar en los ordenadores del laboratorio? ¿y en su propio equipo?

Puesto que los sistemas del laboratorio cuentan con cuatro cores físicos, y en ninguno de ellos está habilitado el *hyperthreading*, el número óptimo de hilos sería cuatro, uno por core. (Aunque no siempre nos compensará crear tantos hilos).

En el equipo en el que hemos ejecutado la práctica, el número óptimo serían ocho, ya que cuenta con 4 cores físicos habilitados con *hyperthreading*, por lo que tenemos ocho cores virtuales.

Para el responder el resto de preguntas, hemos ejecutado el programa *omp2.c* que se nos facilita, obteniendo la siguiente salida:

```
rodrigo@rodrigo-SATELLITE-PRO-A50-D:~/Escritorio/Universidad/Tercero/Arqo/P4$ ./omp2
Inicio: a = 1,   b = 2,  c = 3
        &a = 0x7fff520c0214,x   &b = 0x7fff520c0218,    &c = 0x7fff520c021c

[Hilo 0]-1: a = 0,       b = 2,  c = 3
[Hilo 0]         &a = 0x7fff520c01b0,    &b = 0x7fff520c0218,    &c = 0x7fff520c01ac
[Hilo 0]-2: a = 15,      b = 4,  c = 3
[Hilo 3]-1: a = 0,       b = 2,  c = 3
[Hilo 3]         &a = 0x7f8dbcd89e20,    &b = 0x7fff520c0218,    &c = 0x7f8dbcd89e1c
[Hilo 3]-2: a = 21,      b = 6,  c = 3
[Hilo 2]-1: a = 0,       b = 2,  c = 3
[Hilo 2]         &a = 0x7f8dbd58ae20,    &b = 0x7fff520c0218,    &c = 0x7f8dbd58ae1c
[Hilo 2]-2: a = 27,      b = 8,  c = 3
[Hilo 1]-1: a = 0,       b = 2,  c = 3
[Hilo 1]         &a = 0x7f8dbdd8be20,    &b = 0x7fff520c0218,    &c = 0x7f8dbdd8be1c
[Hilo 1]-2: a = 33,      b = 10,         c = 3

Fin: a = 1,      b = 10,         c = 3
        &a = 0x7fff520c0214,    &b = 0x7fff520c0218,    &c = 0x7fff520c021c
```

Pareja 42: Carlos Gómez-Lobo Hernaiz, Rodrigo Lardiés Guillén

Pregunta 3) : ¿Cómo se comporta *OpenMP* cuando declaramos una variable privada?

Al declarar una variable privada, cada hilo asigna una dirección de memoria diferente para esa variable en su espacio de direcciones asignado. De esta forma, el acceso a la variable queda restringido para cada hilo, el cual solo podrá acceder y modificar sus variables privadas sin que estas queden reflejadas en el resto de hilos.

En el programa *omp2*, vemos cómo las variables a y c son privadas, y tienen asginada una posición de memoria diferente por cada hilo. Además, al acabar la ejecución, el valor de las variables en el hilo principal, no se ha visto afectado por las modificaciones realizadas en el resto de hilos.

Pregunta 4) : ¿Qué ocurre con el valor de una variable privada al comenzar a ejecutarse la región paralela?

Al comenzar a ejecutarse la región paralela, una variable privada no estará inicializada, por lo que tomará el valor que tenga en el espacio de direcciones asignado por ese hilo para esa variable.

En el programa *omp2.c*, las operaciones que se realizan son las siguientes:

```
b += 2;
c = a*a + 3;
a = b*3 + c;
```

La variable c no se ve modificada. Esto se debe a que la variable a no ha sido inicializada y ha tomado el valor 0 en cada hilo, pues las nuevas direcciones estaban vacías.

Por el contrario, la variable c sí que comienza con el valor 3, ya que esta se ha declarado con *firstprivate*, tomando así el valor que se le dio al inicializarla en el hilo principal.

Pregunta 5) : ¿Qué ocurre con el valor de una variable privada al finalizar la región paralela?

Como se aprecia en la salida del programa, el valor final de las variables privadas no se ve afectado por las modificaciones realizadas en los hilos, ya que en estas, el valor de las variables privadas se guarda en un espacio de direcciones diferente al del hilo principal. Por lo que una vez finalizada la región paralela, las variables privadas conservarán el valor almacenado en el hilo principal.

Pregunta 6) : ¿Ocurre lo mismo con las variables públicas?

No, las variables públicas se almacenan en una única posición de memoria, que es común a todos los hilos, por lo que todos los hilos pueden acceder a ella y modificarla si es preciso. Por lo que al finalizar la región paralela del programa, el valor de la variable pública, como se puede apreciar en la salida del programa, si que se habrá visto afectado por los cambios realizados en los hilos.

Pareja 42: Carlos Gómez-Lobo Hernaiz, Rodrigo Lardiés Guillén

Ejercicio 2

En este ejercicio hemos usado los diferentes programas que se nos facilitan para realizar el producto escalar. Hay uno que lo realiza en serie, mientras que los otros dos usan paralelización. Calcularemos tiempos de ejecución y rendimiento para ambas versiones.

Pregunta 1) y 2) : ¿En qué caso es correcto el resultado?. ¿A qué se debe esta diferencia?

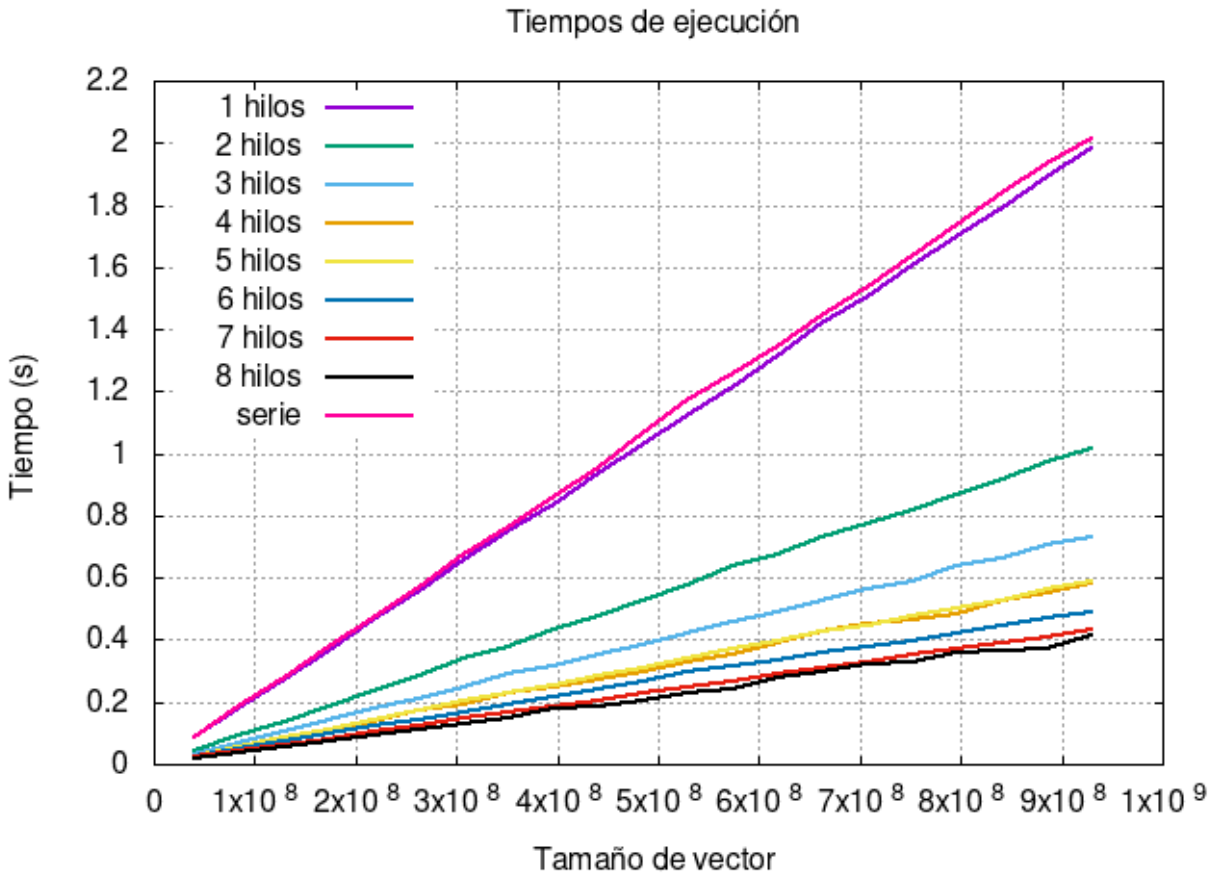
Al ejecutar los tres programas observamos que el resultado devuelto por *pescalar_par1.c* difiere de los otros dos y es incorrecto. Esto se produce porque en dicho programa no se usa en la paralelización del bucle *reduction (+:sum)* sino que se utiliza direcatmente la misma variable para todos los hilos, sin gentionar los posibles problemas de concurrencia, lo uqe produce inconsistencias en la variable sum, que se traducen en un resultado incorrecto. Esto no ocurre en pescalar_par2.c porque al utilizar la directiva reduction(+:sum), se hace que cada hilo tenga una variable sum privada, y luego se sumen, solucionando ese problema.

- Gráficas

En el enunciado se nos pide realizar un *script* que pruebe los programas de producto escalar con diversos tamaños para analizar el rendimiento de ambos variando el número de hilos asignados. Para elegir los tamaños se especifica que el menor dure aproximadamente 0,1 segundos (tamaño: 40.000.000) y el mayor 10 segundos. Para la elección del tamaño mayor tuvimos problemas ya que no había suficiente memoria para una matriz de ese tamaño, así que finalmente escogimos 930.000.000 que aunque no llegaba a 10 segundos, se acercaba lo suficiente sin dar problemas de memoria.

Además para aumentar la precisión de las medidas, para cada tamaño se repite 5 veces y se hace la media de los resultados obtenidos.

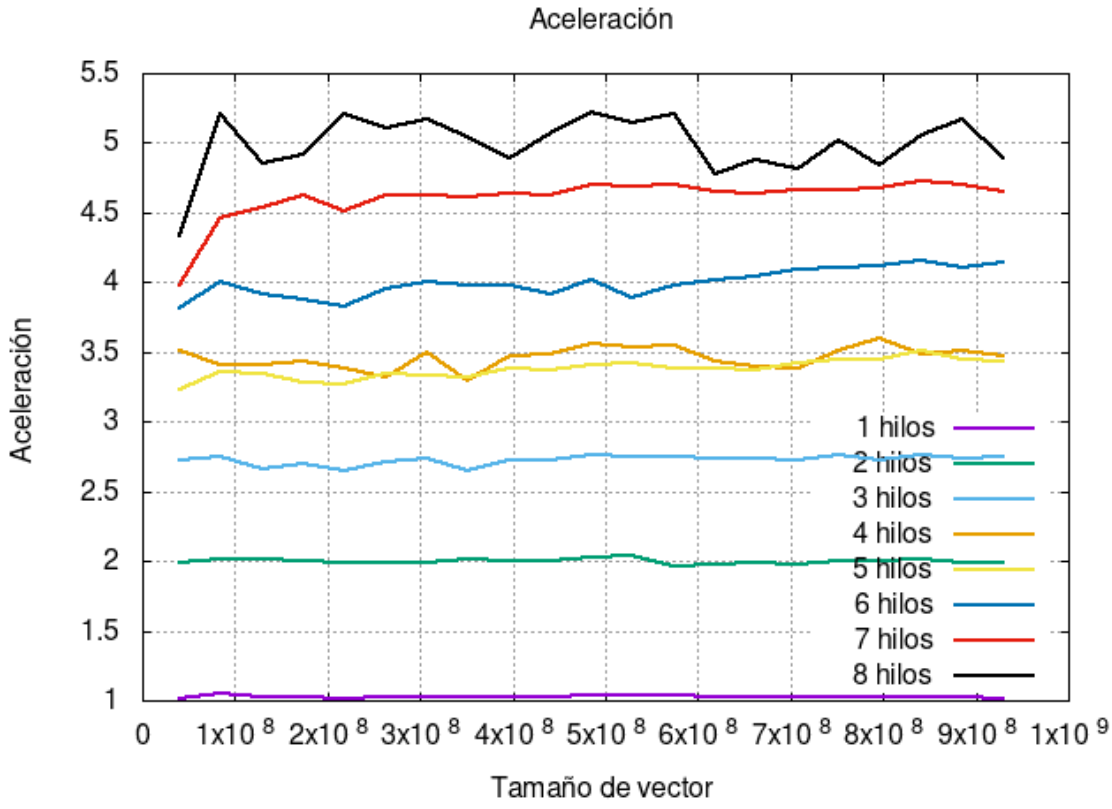
Pareja 42: Carlos Gómez-Lobo Hernaiz, Rodrigo Lardiés Guillén



Podemos observar que los tiempos siguen una progresión lineal al ir aumentando el tamaño de la matriz, y que a mayor número de hilos, como es lógico, el tiempo de ejecución es menor. También observamos que el tiempo del producto escalar en serie y el de paralelo con un hilo es muy similar.

Para calcular la aceleración, hemos tomado el cociente entre el tiempo en serie (sin mejora) y el tiempo en paralelo (con mejora):

Pareja 42: Carlos Gómez-Lobo Hernaiz, Rodrigo Lardiés Guillén



Podemos observar que para un hilo la aceleración es 1, ya que el tiempo coincide prácticamente con el tiempo en serie. Y que al aumentar el número de hilos, la aceleración va aumentando, aunque cada vez menos, ya que usar por ejemplo, 6 hilos no tarda una sexta parte de lo que tarda con uno (excepto con 2, que se puede observar que tarda aproximadamente la mitad).

Pregunta 3) y 4) : En términos del tamaño de los vectores, ¿compensa siempre lanzar hilos para realizar el trabajo en paralelo, o hay casos en los que no?. Si compensara siempre, ¿en qué casos no compensa y por qué?

Para tamaños muy pequeños, la mejora es muy pequeña, incluso puede ser que llegue el caso en el que se tarde más al lanzar varios hilos que al hacerlo en serie. Pero al ir aumentando el tamaño se aprecia claramente como el tiempo se va reduciendo considerablemente y compensa lanzar varios hilos. Pero para los tamaños que hemos tomado nosotros compensa siempre lanzar varios hilos, y cuantos más hilos mejor (hasta 8 hilos), exceptuando para 4 y 5 hilos, que se observa que apenas hay mejora, incluso para algún tamaño empeora.

Pregunta 5) y 6): ¿Se mejora siempre el rendimiento al aumentar el número de hilos a trabajar?. Si no fuera así, ¿a qué debe este efecto?

Sí, exceptuando cuando tomamos 4 y 5 hilos, que apenas hay mejora, y a veces incluso empeora. Esto puede deberse a que el número de núcleos físicos es 4, aunque haya 8 núcleos virtuales, estos no mejoran el rendimiento igual que si tuviéramos 8 físicos, por eso

al parar de los 4 hilos, el rendimiento mejora, pero no tanto como antes. Además, el rendimiento será el mismo e incluso empeorará si se lanzan más hilos de los que la máquina pueda ejecutar al mismo tiempo.

Pregunta 7) : Valore si existe algún tamaño del vector a partir del cual el comportamiento de la aceleración va a ser muy diferente del obtenido en la gráfica.

Para tamaños muy pequeños de vectores la aceleración será mucho menos ya que no será tan rentable la creación de tantos hilos para el número de operaciones a realizar.

Ejercicio 3

Para este ejercicio, partimos del código de la multiplicación traspuesta de matrices de la práctica anterior. Realizaremos diversas pruebas con dicho programa calculando el producto en serie, y en paralelo, (haciendo una versión diferente dependiendo del buble que paralelicemos.

Los tamaños escogidos para seleccionar que programa es el más óptimo son 1500 y 2700:

Versión\ # Hilos	1	2	3	4
Serie	10.618550			
Paralelo - 1	8,580256	5,929898	4,841697	4,239595
Paralelo - 2	10,617374	5,688539	3,937060	2,736221
Paralelo - 3	10,950619	5,525569	3,718026	2,856777

Tamaño 1500: Tiempo de ejecución (s)

Versión\ # Hilos	1	2	3	4
Serie	1	1	1	1
Paralelo - 1	1.2375563153	1.7906800420	2.1931463286	2.5046142379
Paralelo - 2	1.0001107618	1.8666567988	2.6970759907	3.8807355107
Paralelo - 3	0.9696757781	1.9217115920	2.8559644284	3.7169684578

Tamaño 1500: Aceleración

Pareja 42: Carlos Gómez-Lobo Hernaiz, Rodrigo Lardiés Guillén

Versión\ # Hilos	1	2	3	4
Serie	62.897145			
Paralelo - 1	47.490655	29.443537	22.211391	18.628562
Paralelo - 2	62.097235	31.625585	21.305243	16.080522
Paralelo - 3	62.258334	31.712163	21.213459	15.911131

Tamaño 2700: Tiempo de ejecución (s)

Versión\ # Hilos	1	2	3	4
Serie	1	1	1	1
Paralelo - 1	1.3244109814	2.1361952879	2.8317517349	3.3763821920
Paralelo - 2	1.0128815719	1.9888057406	2.9521909231	3.9113870184
Paralelo - 3	1.0102606504	1.9833760629	2.9649641296	3.9530279148

Tamaño 2700: Aceleración

Pregunta 1) : ¿Cuál de las tres versiones obtiene peor rendimiento? ¿A qué se debe?

El peor rendimiento se da en la versión 3 lanzando un único hilo. Esto es posible que sea a esta ejecución en concreto, ya que en nuestra opinión tiene más sentido que fuera el paralelo 1, ya que crea muchos más hilos por estar en el bucle más interno.

Pregunta 2) : ¿Cuál de las tres versiones obtiene mejor rendimiento? ¿A qué se debe?

El mejor rendimiento se obtiene también en la versión 3 pero esta vez lanzando 4 hilos. Esto se debe a que es la que utiliza los 4 hilos pero lanzándolos solo para el bucle más exterior, que es el que menos veces se ejecuta, tardando menos tiempo en la creación de hilos consiguiendo el mismo grado de paralelización.

- Gráficas

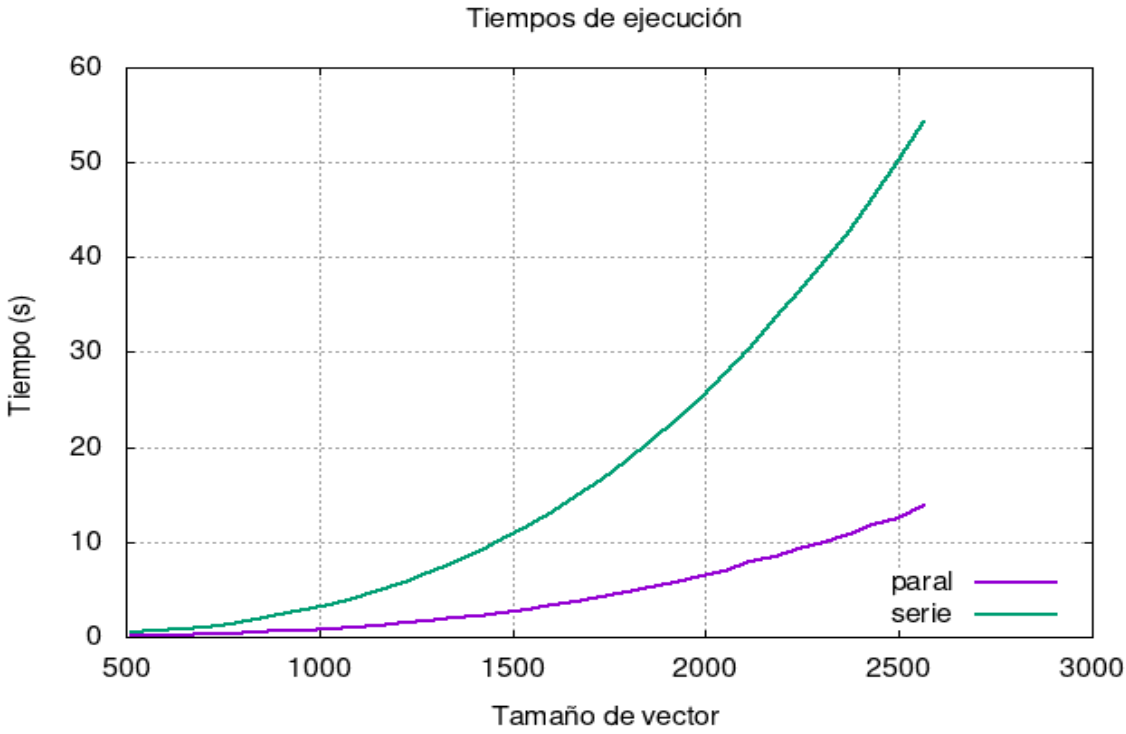
Para realizar las gráficas hemos usado los programas de multiplicación en serie, y en paralelo con la versión 3 lanzando 4 hilos (que es el más óptimo de todos).

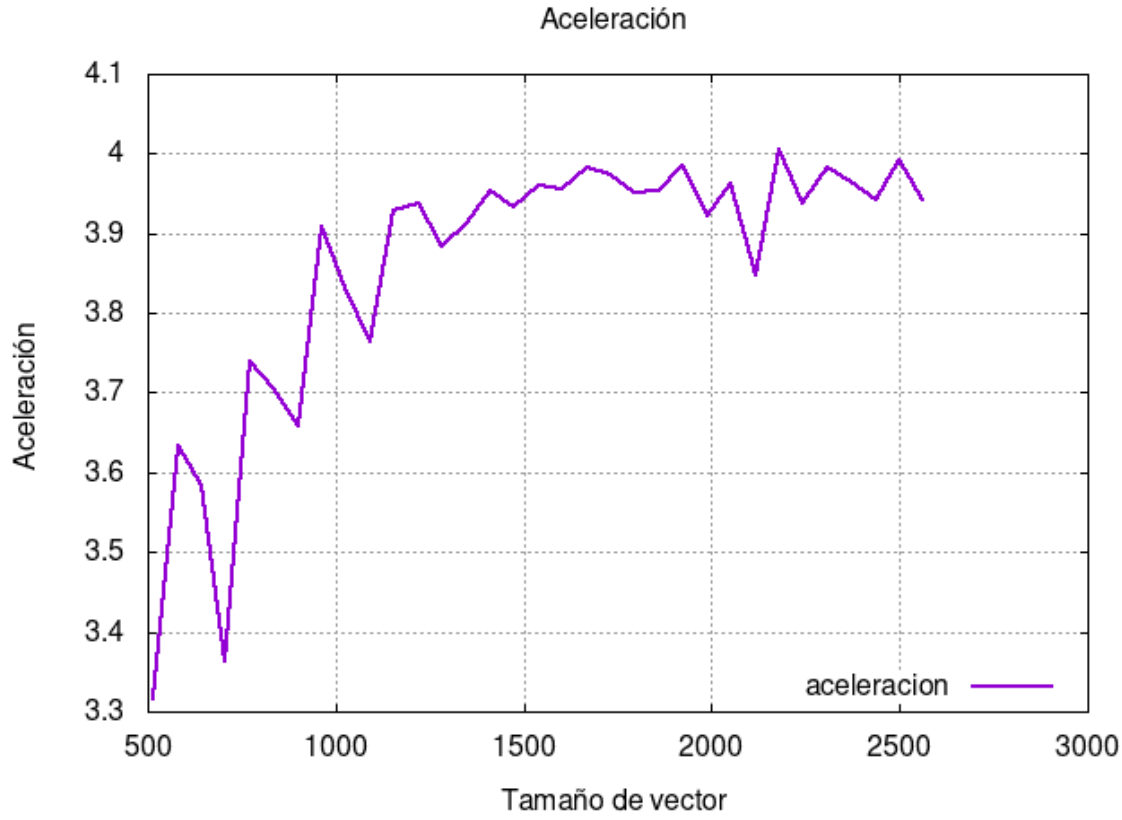
Hemos tomado los tamaños variando de 515 a 2563, como se indica en el enunciado del ejercicio, y además hemos realizado 5 repeticiones y la media de los resultados obtenidos para aumentar la precisión.

La aceleración la hemos calculado tomando el cociente entre el tiempo del programa en serie (sin mejora) y el tiempo en paralelo (con mejora).

Las gráficas obtenidas son las siguientes:

Pareja 42: Carlos Gómez-Lobo Hernaiz, Rodrigo Lardiés Guillén





Observamos que los tiempos de ejecución siguen un crecimiento exponencial, donde el tiempo en paralelo crece mucho más rapido. Además en la gráfica de la aceleración

comprobamos efectivamente que es mucho más rápida estabilizándose al aumentar el tamaño en torno a 1200-1500.

Pregunta 3) : Con los valores en función de nuestro P=3 no se apreciaba la astabilización de la aceleración, de modo que añadimos 1024 tamaños más.

Ejercicio 4

Pregunta 1) : ¿Cuántos rectángulos se utilizan en la versión del programa que se da para realizar la integración numérica?

Se utilizan n=100000000 rectángulos, que a su vez se dividen en dos para calcular la imagen del punto medio del rectángulo, pero se calcula la suma de las áreas de esos 100000000.

Pregunta 2) : ¿Qué diferencias observa entre estas dos versiones?

Que la v4 utiliza una variable privada para ir guardando los resultados de las sumas y luego guarda el resultado final en su parte del array sum, mientras que la v1 utiliza un array común para todos los hilos.

Ejercicio 3) : Ejecute las dos versiones recién mencionadas. ¿Se observan diferencias en el resultado obtenido? ¿Y en el rendimiento? Si la respuesta fuera afirmativa, ¿sabría justificar a qué se debe este efecto?

En el resultado no se obtiene ninguna diferencia, sin embargo, el rendimiento de la v4 es mejor que el de la v1 en aproximadamente un 70%. Esto se debe a que al guardar la variable *sum* en la cache, varias posiciones del array se guardan en el mismo bloque, entonces, como la variable sum es compartida en la v1, si varios hilos acceden a la vez a partes del array que están en el mismo bloque de la caché, habrá un retardo por concurrencia ya que solo una puede acceder a los datos a la vez para evitar inconsistencia en los datos, lo que retarda la ejecución del programa. Esto no sucede en la v4, ya uqe cada hilo tiene su propia variable privada.

Ejercicio 4) : Ejecute las versiones paralelas 2 y 3 del programa. ¿Qué ocurre con el resultado y el rendimiento obtenido? ¿Ha ocurrido lo que se esperaba?

El resultado en ambos es el mismo he igual al resto. En cuanto al rendimiento de la v2, es muy similar al de la v1, ya que aunque comparta la variable *sum* donde se guarda la dirección de memoria del array, este sigue siendo compartido y no soluciona el problema de la v1. La v3 sin embargo, al ver el tamaño de cada bloque de la caché, y asignar un bloque a cada hilo, aunque consuma más memoria, consigue solucionar el problema de concurrencia y conseguir un rendimiento similar al de la v4.

Pareja 42: Carlos Gómez-Lobo Hernaiz, Rodrigo Lardiés Guillén

Ejercicio 5) : Abra el fichero pi_par3.c y modifique la línea 32 del fichero para que tome los valores fijos 1, 2, 4, 6, 7, 8, 9, 10 y 12. Ejecute este programa para cada uno de estos valores. ¿Qué ocurre con el rendimiento que se observa?

Se aprecia que a medida que aumenta el número se va mejorando el rendimiento, hasta llegar a 8, a partir del cual se estabiliza y tarda aproximadamente lo mismo. Esto tiene sentido pues una vez se llega a 8, aunque dejes más espacio entre las posiciones del array que utiliza cada hilo, solo habrá una a la que se acceda en cada bloque de la caché, eliminando así la concurrencia para cualquier valor mayor o igual a 8.

Ejercicio 5

Ejercicio 1) : Ejecute las versiones 4 y 5 del programa. Explique el efecto de utilizar la directiva critical. ¿Qué diferencias de rendimiento se aprecian? ¿A qué se debe este efecto?

Al usar la directiva critical, se establece que solo un hilo puede ejecutar el bloque de código asociado (en este caso, la línea) al mismo tiempo. Esto permite sumar directamente el valor de las sumas que realiza cada hilo al valor de pi provisional cunado estos terminan la ejecución en lugar de hacerlo sobre una variable privada falsa y luego sumárselo después. Sin embargo, debido a que la i utilizada en el bucle *for* y a que la variable pi no está inicializada, el resultado de *pi_par5* es incorrecto, y tiene un tiempo de ejecución considerablemente mayor al esperado. Por ello, para poder comparar ambos programas correctamente, hemos corregido estos errores y hemos comprobado que los tiempos de ejecucíon son muy similares. Esto tiene sentido ya que aunque en *pi_par4* también hay concurrencia cuando los espacios del array están en el mismo bloque, habrá menos que en pi_par5, en el que simepre hay concurrencia; aunque por otro lado, en pi_par5 se ahorra el bucle que suma las sumas parciales, por lo que le tiempo de ejecución tendrá que ser similar.

Ejercicio 2) : Ejecute las versiones 6 y 7 del programa. Explique el efecto de utilizar las directivas utilizadas. ¿Qué diferencias de rendimiento se aprecian? ¿A qué se debe este efecto?

En *pi_par6*, en cada hilo se calculan todas las sumas parciales que se realizan para calcular pi, aunque a su vez en hilos distintos, sin embargo se hacen las mismas sumas tantas veces como núcleos tenga la máquina, lo que no es muy eficiente. En cambio en *pi_par7,* lo que se hace con la directiva *reduction,* es hacer que las variables *sum* sean privadas en cada hilo, y al terminar las suma, por lo que elimina la concurrencia y evita el bucle que las suma todas, lo que aumenta considerablemente el rendimiento respecto a *pi_par6*.