Práctica 4:

Explotar el potencial de las arquitecturas modernas

Arquitectura de Ordenadores

Grupo 1312

Curso 2017-2018

Nuria Cuaresma Saturio

María Barroso Honrubia

Cálculos iniciales para obtener el valor de P:
Pareja numero 40
40 = 0 mod 8
P = 0 +1 = 1

Ejercicio 0: Información sobre la topología del sistema

Tras ejecutar el comando *cat /proc/cpuinfo* y examinar su salida, comprobamos que nuestro equipo tiene un total de 4 CPUs físicas (cpu cores) y 4 CPUs virtuales (siblings).

Por tanto, el número de cores que tiene nuestro equipo es :
CPU's físicas / CPUs virtuales = 4 / 4 = 1, luego no hay hyperthreading.

Ejercicio 1: Programas básicos de OpenMP

Tras ejecutar el programa omp1.c, respondemos a las siguientes cuestiones:

1.1 ¿Se pueden lanzar más threads que cores tenga el sistema? ¿Tiene sentido hacerlo?

Si se pueden lanzar. Sin embargo, no tiene sentido ya que al no estar activado el hyperthreading solo se ejecutan simultáneamente tantos como cores tenga, en nuestro caso 4.

1.2 ¿Cuántos threads debería utilizar en los ordenadores del laboratorio? ¿y en su propio equipo?

En los ordenadores del laboratorio utiliza 4 threads y en nuestro propio equipo obtenemos que tiene 8 cores, por lo que debería utilizar 8 threads.

Tras ejecutar el programa omp2.c, obtenemos la siguiente salida:

```
e336543@localhost:~/Desktop/Practica4/materialP4$ ./omp2
Inicio: a = 1,   b = 2,  c = 3
        &a = 0x7fffe5334194,x   &b = 0x7fffe5334198,    &c = 0x7fffe533419c

[Hilo 0]-1: a = 0,       b = 2,  c = 3
[Hilo 0]         &a = 0x7fffe5334168,    &b = 0x7fffe5334198,    &c = 0x7fffe533
4164
[Hilo 0]-2: a = 15,      b = 4,  c = 3
[Hilo 2]-1: a = 0,       b = 2,  c = 3
[Hilo 2]         &a = 0x7f7b75e48e58,    &b = 0x7fffe5334198,    &c = 0x7f7b75e4
8e54
[Hilo 2]-2: a = 21,      b = 6,  c = 3
[Hilo 3]-1: a = 0,       b = 2,  c = 3
[Hilo 3]         &a = 0x7f7b75647e58,    &b = 0x7fffe5334198,    &c = 0x7f7b7564
7e54
[Hilo 3]-2: a = 27,      b = 8,  c = 3
[Hilo 1]-1: a = -1,      b = 2,  c = 3
[Hilo 1]         &a = 0x7f7b76649e58,    &b = 0x7fffe5334198,    &c = 0x7f7b7664
9e54
[Hilo 1]-2: a = 34,      b = 10,         c = 4

Fin: a = 1,      b = 10,         c = 3
        &a = 0x7fffe5334194,    &b = 0x7fffe5334198,    &c = 0x7fffe533419c
```

1.3 ¿Cómo se comporta OpenMP cuando declaramos una variable privada?

Cuando declaramos una variable privada, esta es copiada en el almacenamiento local de cada hilo, es decir, cada hilo la usará como variable temporal.

En este programa hemos declarado como variables privadas a y tid, de esta manera cada hilo tendrá un valor individual de estas variables, ajeno a las modificaciones que realizan sobre ellas el resto de hilos. Esto no solo lo observamos en los datos, sino también en las direcciones, ya que la dirección de esta variable a es única y diferente para cada hilo ya que funciona como variables distintas e individuales a cada uno.

Podemos observar también en el código la declaración semiprivada firstprivate de la variable c. Actúa similar a las variables privadas de forma que cada hilo tiene una copia local del dato, pero en este caso se inicializa con el valor global que tiene en el momento de la declaración. Esto también queda reflejado en el resultado obtenido ya que comienza en cada hilo con el valor 3 (con el que se inicializó anteriormente) y a continuación tanto las direcciones, que son diferentes para cada hilo, como los datos, que no se ven afectados por modificaciones en otros hilos, funcionan como privados.

1.4 ¿Qué ocurre con el valor de una variable privada al comenzar a ejecutarse en la región paralela?

Como las variables privadas no se inicializan, cuando se comienza a ejecutar la región paralela, estas no tomarán el valor 1 que le asignamos previo a su declaración como privada, sino que toman, en nuestro caso, un valor entre 0 y -1, como podemos observar para la variable a declarada como privada.

1.5 ¿Qué ocurre con el valor de una variable privada al finalizar la región paralela?

Cuando finaliza la región paralela, estas variables no conservan los cambios realizados durante el periodo paralelo en el que fueron privadas, mantienen el valor según el cual habían sido inicializadas antes de su uso como tal, así como su dirección de memoria.

1.6 ¿Ocurre lo mismo con las variables públicas?

Por el contrario, para las variables públicas, los datos de la región paralela son compartidos, lo que significa que son visibles y accesibles por todos los hilos. En nuestro programa se ha definido como pública la variable b y vemos cómo se mantiene la misma dirección tanto dentro como fuera de la región paralela, y es común a todos los hilos.

En cuanto al valor, al comenzar a ejecutarse en la región paralela, la variable b está inicializada con el mismo valor que tenía guardado, sin embargo, sus modificaciones se mantienen en todos los hilos. Al finalizar la región paralela, el valor de la variable privada es aquel que tenía en la última modificación dentro de la zona paralela.

Ejercicio 2: paralelizar el producto escalar

2.1 ¿En qué caso es correcto el resultado?

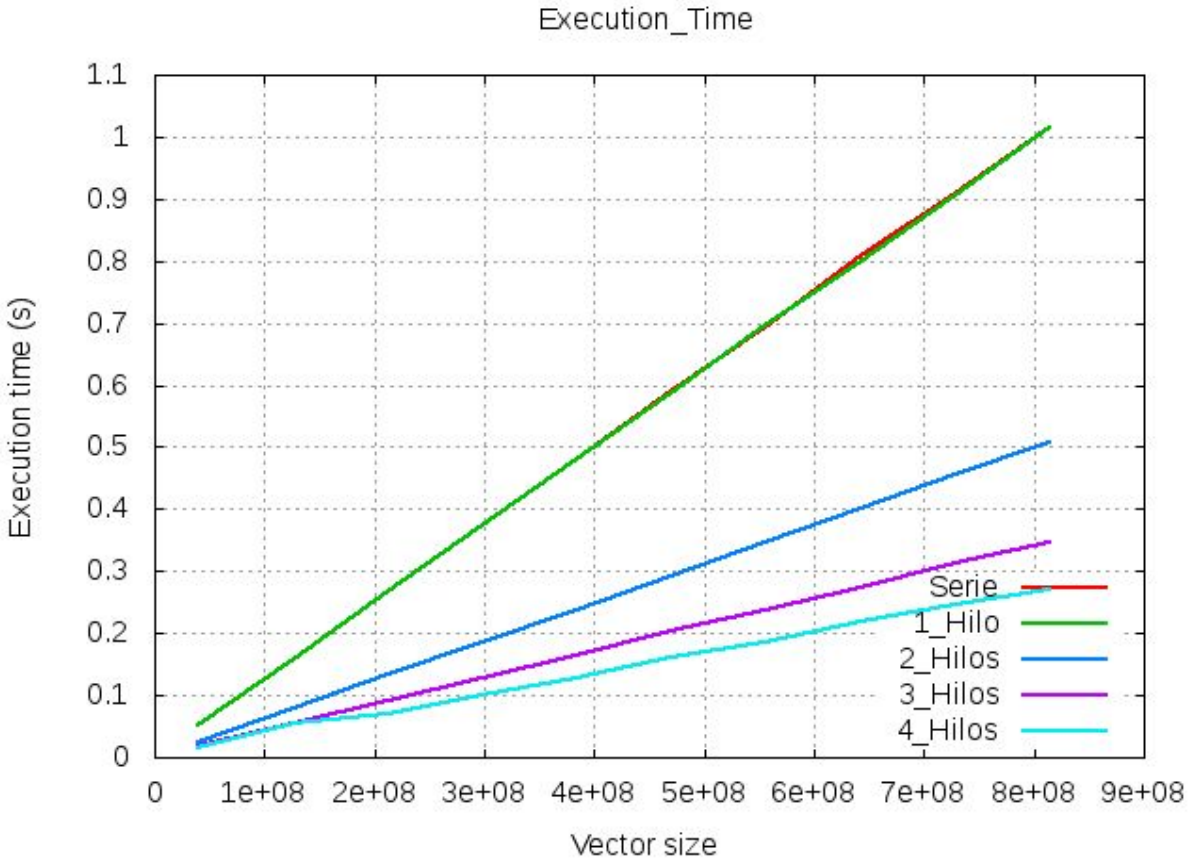
El resultado es correcto tanto para el programa pescalar_serie.c, que realiza el producto escalar sin paralelización, como para el programa pescalar_par2.c, que realiza la paralelización del producto escalar entre los hilos del equipo utilizando la cláusula reduction para la compartición de datos.

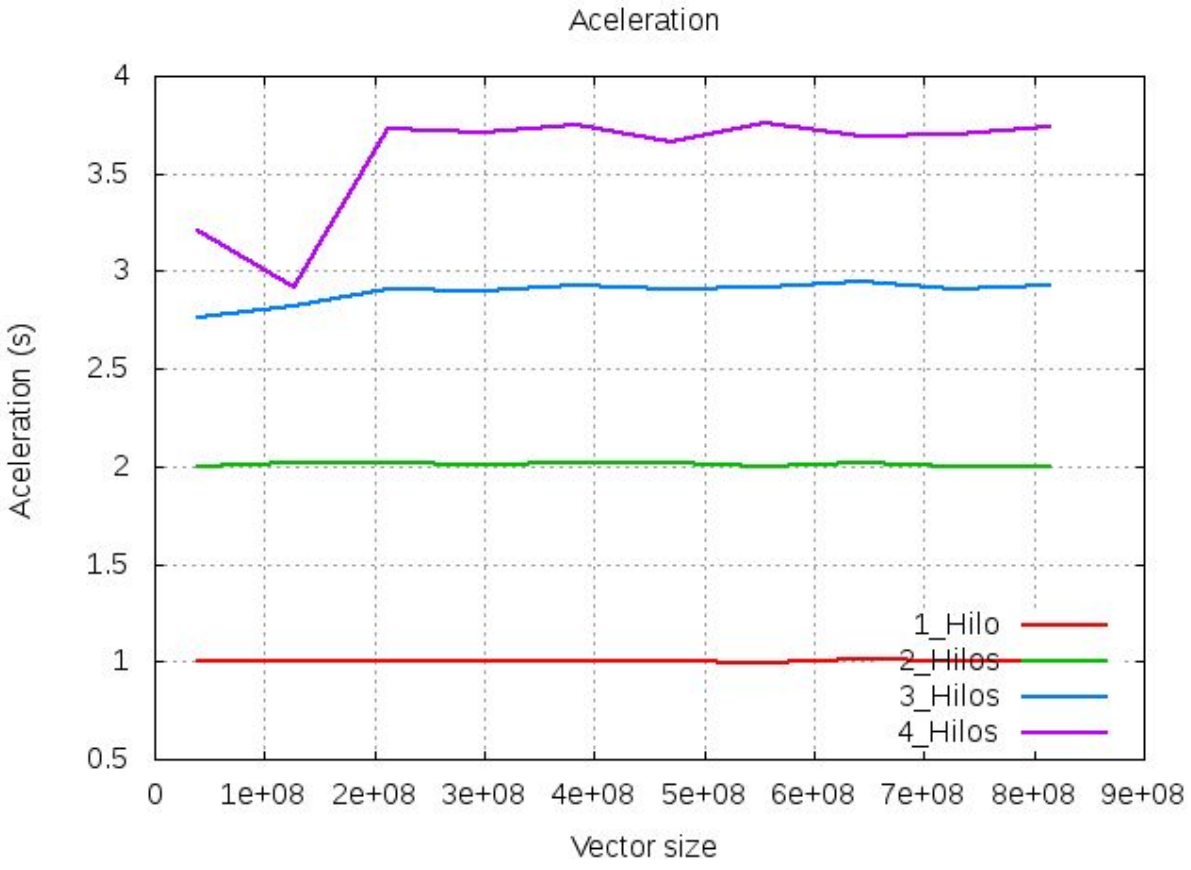
2.2 ¿A qué se debe esta diferencia?

En el programa pescalar_par2.c, cada hilo creará la variable privada sum de nuestro programa, que será combinada usando el operador “+” y se asignará a la variable original global, también sum,al final del bloque de hilo.

Sin embargo, el programa pescalar_par1.c no realiza de forma correcta esta operación ya que al no utilizar la cláusula reduction la variable suma toma un valor distinto para cada hilo, quedándose de esta forma con el resultado que generó el último hilo que realizó la ejecución.

Tras ejecutar el programa pescalar_serie.c con diferentes tamaños de vectores, la dimensión que más aproxima su tiempo de ejecución a 0,1 segundos es 40000000. Por otra parte, para obtener el tamaño cuyo tiempo estuviese alrededor de 10 segundos, hemos decidido utilizar un valor de 900000000. Con dichos valores establecemos un rango a la hora de tomar medidas del tiempo de ejecución y aceleración para los diferentes programas que pide el enunciado y representamos los resultados.


Execution_Time
Execution time (s)
Vector size
Serie
1_Hilo
2_Hilos
3_Hilos
4_Hilos
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
0
1e+08
2e+08
3e+08
4e+08
5e+08
6e+08
7e+08
8e+08
9e+08


Aceleration
Aceleration (s)
Vector size
1_Hilo
2_Hilos
3_Hilos
4_Hilos
0.5
1
1.5
2
2.5
3
3.5
4
0
1e+08
2e+08
3e+08
4e+08
5e+08
6e+08
7e+08
8e+08
9e+08

2.3 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 se obtiene una aceleración mayor que 1, el procesamiento paralelo será más rápido que en serie, por tanto sólo compensará lanzar hilos cuando se consigan valores mayores a la unidad.

En nuestra simulación, el programa que utiliza un único hilo es prácticamente igual de eficiente que el programa serie ya que se obtiene una aceleración muy próxima a 1 para todos los tamaños tomados. Sin embargo, para el resto de programas sí que se aprecia una gran mejora.

2.4 Si compensará siempre, ¿en qué casos no compensa y por qué?

No compensa para tamaños muy pequeños, ya que se invierte tiempo en lanzar los hilos y gestionarlos, mientras que con procesamiento en serie se empieza antes a realizar los cálculos.

2.5 ¿Se mejora siempre el rendimiento al aumentar el número de hilos a trabajar?

El rendimiento no mejora siempre que se aumente el número de hilos ya que dependerá del número de procesadores que disponga el ordenador en el que se esté trabajando. En nuestro caso, estamos trabajando con un ordenador que consta de 4 procesadores reales y 4 virtuales, luego si se empleasen más de 4 hilos, sólo se ejecutarían simultáneamente 4.

2.6 Si no fuera así, ¿a qué debe este efecto?

Por lo comentado en el apartado de anterior, podemos establecer diferencias entre la forma que tiene los procesadores de gestionar los hilos.

Cuando se utiliza un número de hilos menor que los disponibles por el equipo, el programa se ejecuta entre los 4 procesadores, invirtiendo tiempo innecesario en crear dichos hilo cuando se podría ejecutar de la misma forma en serie. Cuando se lanzan 4 hilos, se asocia cada hilo a un procesador, siendo de esta forma la ejecución mucho más rápida. Para el resto de casos, hay que diferenciar los casos en los que se tiene un número de hilos múltiplos de 4 y cuando no. Para el primer caso, se reparten los hilos entre todos los procesadores pero para el segundo, los procesadores se turnaran la ejecución del hilo o de hilos “sobrantes”.

2.7 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.

Utilizando tamaños de vectores muy pequeños si que se obtendría un comportamiento diferente ya que utilizar hilos no sería eficiente como hemos comentado en el apartado 2.4.

Ejercicio 3: Paralelizar la multiplicación de matrices

Tras desarrollar tres versiones paralelas del programa que calcula la multiplicación de matrices de la práctica anterior, obtenemos los siguientes resultados de tiempo de ejecución y rendimiento.

Tiempo de ejecución (1000)

Versión\#hilos	1	2	3	4
Serie	5,35437			
Paralela-bucle1	6,60139	4,449513	3,396985	3,095649
Paralela-bucle2	6,681941	3,347872	2,247635	1,751716
Paralela-bucle3	6,966703	3,354096	2,272081	1,72265

Aceleración (1000)

Versión\#hilos	1	2	3	4
Serie	1			
Paralela-bucle1	0,81109736	1,20336091	1,57621244	1,72964377
Paralela-bucle2	0,801319557	1,59933534	2,38222398	3,05664274
Paralela-bucle3	0,768565848	1,59636755	2,35659292	3,10821699

Tiempo de ejecución (2100)

Versión\#hilos	1	2	3	4
Serie	70,577183			
Paralela-bucle1	69,740432	38,36311	28,112787	33,391446
Paralela-bucle2	78,324476	40,058128	26,690503	21,353446
Paralela-bucle3	67,036743	33,675747	22,627729	17,246118

Aceleración (2100)

Versión\#hilos	1	2	3	4
Serie	1			
Paralela-bucle1	1,01199808	1,83971485	2,51050111	2,11363063
Paralela-bucle2	0,9010872	1,76186923	2,64428074	3,30518938
Paralela-bucle3	1,05281343	2,09578671	3,11905729	4,09235186

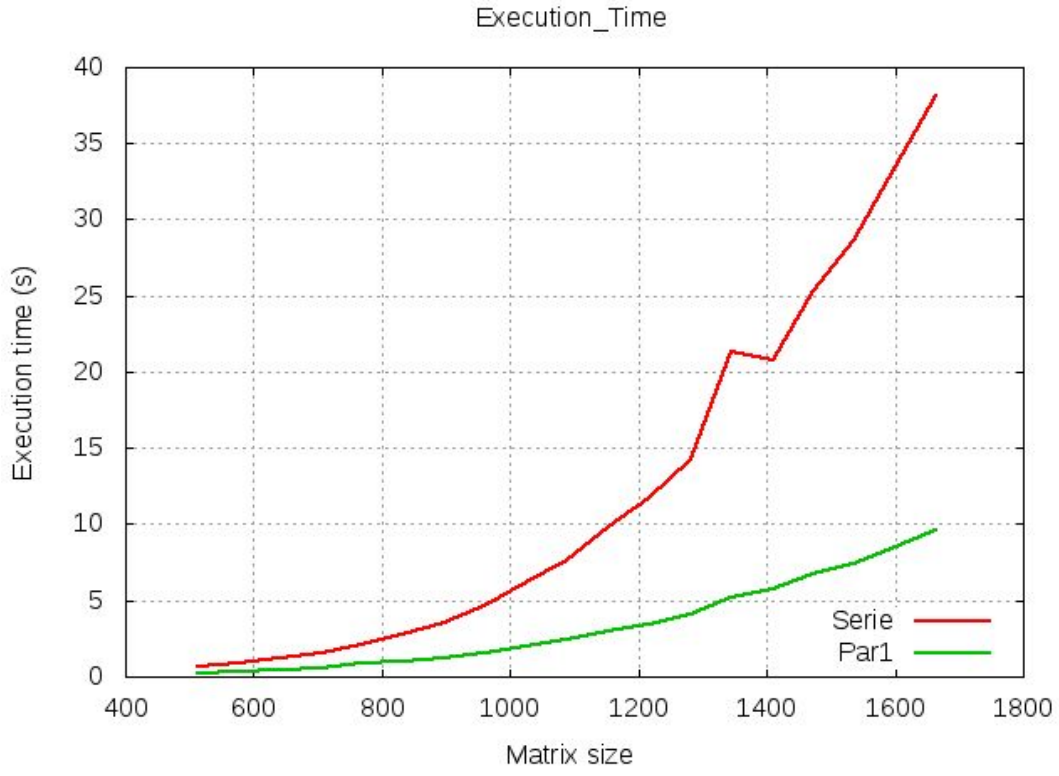
3.1 ¿Cuál de las tres versiones obtiene peor rendimiento? ¿A qué se debe?

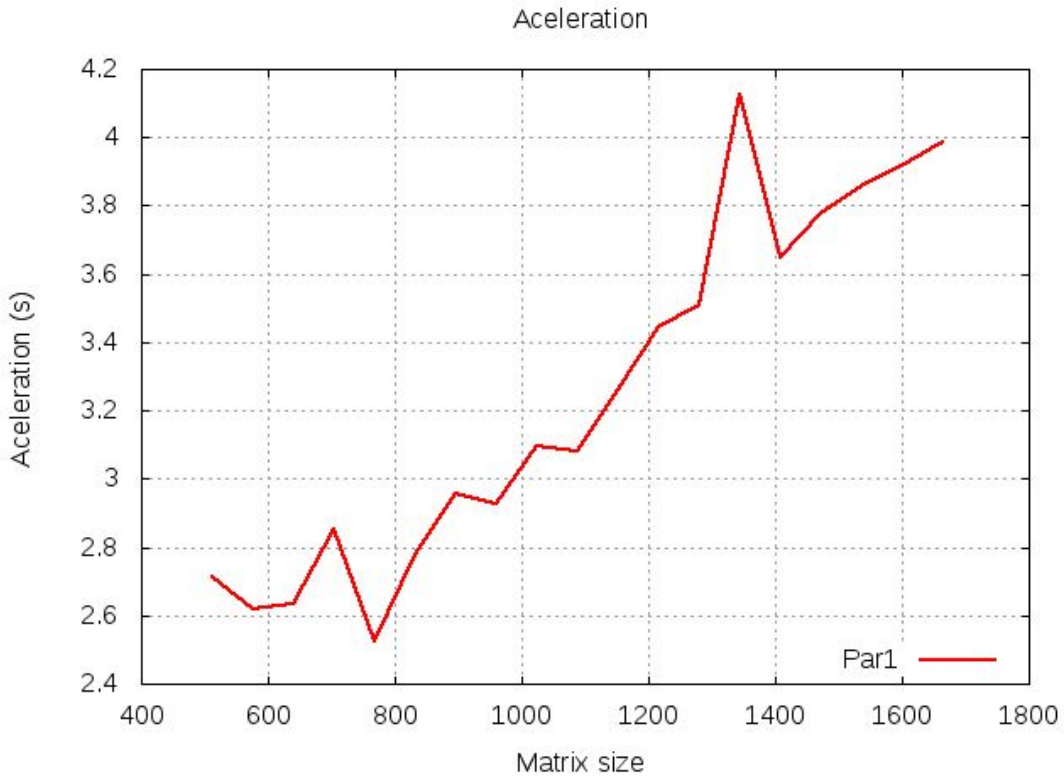
El peor rendimiento se consigue ejecutando el programa prod_par3 (Paralela-bucle3) con un valor aproximadamente de 0,76 segundos para un tamaño de matriz de 1000. Sin embargo, no existen grandes diferencias entre las tres versiones, esto puede deberse a que el equipo con el que hemos trabajado no dispone de los procesadores necesarios para aprovechar el máximo rendimiento de los hilos.

3.2 ¿Cuál de las tres versiones obtiene mejor rendimiento? ¿A qué se debe?

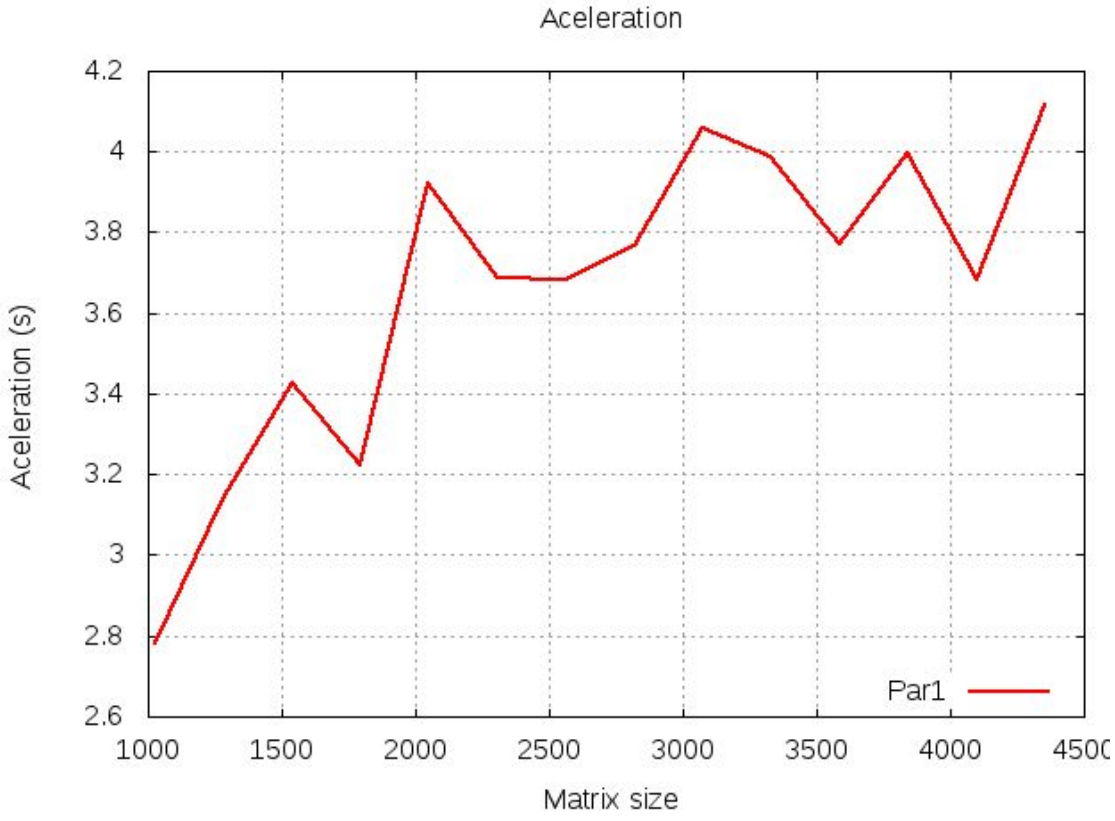
El mejor rendimiento se consigue ejecutando el programa prod_par3 (Paralela-bucle3) con un valor aproximado de 4 segundos para un tamaño de matriz de 2100. Esto se debe a que paralelizar el bucle externo supone agrupar las n tareas adaptándolas a los cores disponibles, mientras que hacerlo en el bucle más interno supondría adaptar un total de n*n*n tareas, lo cual es muy costoso en tiempo de ejecución ya que se invierte mucho más tiempo en hacer la sincronización.

Representación de los resultados obtenidos de tiempo de ejecución y rendimiento para la version que obtiene el mejor rendimiento utilizando 4 hilos.





3.3 Si en la gráfica anterior no obtuvo un comportamiento de la aceleración en función de N que se estabilice o decrezca al incrementar el tamaño de la matriz, siga incrementando el valor de N hasta conseguir una gráfica con este comportamiento e indique para qué valor de N se empieza a ver el cambio de tendencia.



Como la aceleración no llega a estabilizarse ni empieza a decrecer para los tamaños tomados, hemos aumentado el tamaño de las matrices (de 2^10 a 2^12 aproximadamente). De esta forma y tal y como podemos observar, vemos que a partir de un valor de 2500 se empieza a apreciar este cambio de tendencia en la aceleración y se estabiliza.

Ejercicio 4: Ejemplo de integración numérica

4.1 ¿Cuántos rectángulos se utilizan en la versión del programa que se da para realizar la integración numérica?

En el fichero pi_serie.c facilitado podemos ver que se utilizan un total de 100.000.000 rectángulos.

4.2 ¿Qué diferencias observa entre estas dos versiones?

Fijándonos concretamente en los ficheros pi_par1.c y pi_par4.c podemos ver que estas versiones se diferencian en el método empleado para calcular el valor de la función $f(x) = \frac{1}{1+x \cdot x}$ en las diferentes regiones establecidas. La versión pi_par1 calcula directamente dicha función utilizando para ello una variable compartida por los diferentes hilos, mientras que la versión pi_par4 utiliza una variable auxiliar privada para calcular el resultado individual de cada hilo y posteriormente guardarlo en la variable compartida.

4.3 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?

Versión\Medida	Tiempo ejecución	Aceleración	Resultado
Serie	1,385634	1	3,141593
Pi - par 1	0,568321	2,438118598	3,141593
Pi - par 4	0,364929	3,796996128	3,141593

Con respecto a los ficheros mencionados anteriormente pi_par1.c y pi_par4.c, al ejecutarlos podemos observar que el resultado es exactamente el mismo, con la misma precisión y mismo número de decimales, pero sin embargo con respecto al tiempo empleado o rendimiento, vemos que el fichero pi_par4.c emplea menor tiempo y la aceleración es mayor que para pi_par1. Esto se debe a que en la versión pi_par1.c cada core tiene una copia del array de forma que un cambio realizado por un procesador supone un cambio en el resto de las copias del array.

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

Versión\Medida	Tiempo ejecución	Aceleración	Resultado
Serie	1,385634	1	3,141593
Pi - par 2	0,568977	2,435307578	3,141593
Pi - par 3	0,375176	3,693290616	3,141593

En este caso ocurre prácticamente lo mismo que en el anterior siendo el 2 el que dedica más tiempo y se obtiene un rendimiento menor que el 3. El programa pi_par2 actúa

de forma muy similar a pi_par1, y en pi_par3 se ha aumentado el tamaño del array donde se guardan los resultados del sumatorio que realiza cada hilo aumentando así su espacio de direcciones en función de las características de la caché. Sin embargo se sigue necesitando recargar los datos en cada core por cada edición de la variable sum.

4.5 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?

Versión\Medida	Tiempo ejecución	Aceleración	Resultado
Serie	1,385634	1	3,141593
Pi - par 3 - 1	0,598829	2,313905973	3,141593
Pi - par 3 - 2	0,575934	2,405890258	3,141593
Pi - par 3 - 4	0,363761	3,809187901	3,141593
Pi - par 3 - 6	0,450047	3,078865096	3,141593
Pi - par 3 - 7	0,451893	3,066287816	3,141593
Pi - par 3 - 8	0,376082	3,684393297	3,141593
Pi - par 3 - 9	0,396517	3,494513476	3,141593
Pi - par 3 - 10	0,376828	3,677099366	3,141593
Pi - par 3 - 12	0,369158	3,753498502	3,141593

Tras cambiar el fichero pi_par3 como indica el enunciado, vemos que se obtiene un rendimiento mayor para el numero 4 ya que nuestro ordenador tiene este mismo número de cores, aunque también se aprecia para 8 al ser múltiplo de este, ya que estamos trabajando con 4 cores y de esta forma se reparten de una manera más eficiente las tareas al trabajar de forma simultanea, como hemos comentado en otros apartados.

Ejercicio 5: Uso de la directiva critical y reduction

5.1 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?

Versión\Medida	Tiempo ejecución	Aceleración	Resultado
Serie	1,385634	1	3,141593
Pi - par 4	0,364929	3,796996128	3,141593
Pi - par 5	1,13549	1,220296084	2,599699

Primeramente cabe destacar que el programa pi_par5 utiliza la directiva *critical*, la cual restringe la ejecución asociada al bloque a un único hilo, obteniendo por lo tanto un valor incorrecto. Además disminuye la eficiencia al reducir el paralelismo, ya que cada hilo pasa mucho tiempo en espera.

5.2: Ejecute las versiones 6 y 7 del programa. Explique el efecto de utilizar las directiva empleadas. ¿Qué diferencias de rendimiento se aprecian? ¿A qué se debe este efecto?

Versión\Medida	Tiempo ejecución	Aceleración	Resultado
Serie	1,385634	1	3,141593
Pi - par 6	0,57741	2,399740219	3,141593
Pi - par 7	0,370942	3,735446512	3,141593

Ambos programas utilizan las directivas parallel, default(shared), private y for, y el programa pi_par7 emplea a su vez reduction.
Con la directiva *parallel* se forma un equipo de hilos y se inicia la ejecución paralela, con *default(shared)* se controla el atributo por defecto de compartición de datos para variables referenciadas en el contructor de la región paralela o tarea, con *private* se declaran privadas en el programa pi_par6 la variable *numThreads* y para pi_par7 las variables i,j utilizadas en el bucle for. La directiva *for* se utiliza para especificar que las iteraciones se distribuirán y ejecutarán por un equipo de hilos y además en caso de pi_par7 con *reduction* cada hilo tiene su propia variable sum que utiliza por separado, y al final de la ejecución de los hilos todas las variables sum se suman para formar una sola.

El programa pi_par7 es el doble de eficiente que el programa pi_par6, esto se debe al uso de la directiva reduction la cual es muy potente por lo comentado en el párrafo anterior.