ARQUITECTURA DE ORDENADORES

PRACTICA 4

Salvador Martín Barcia

Alberto Martínez González

**Ejercicio 0:**

A screenshot of a cell phone

Description automatically generatedAl ejecutar el comando cat /proc/cpuinfo se nos muestra la información correspondiente a cada procesador del ordenador del laboratorio. En nuestro caso el ordenador tiene 4 núcleos, todos de aproximadamente 800 MHz de frecuencia, como se puede ver en la línea cpu MHz. El hyperthreading no esta activo ya que el número de la línea cpu\_cores y el de la línea siblings es el mismo.

Abajo se muestra una captura de la información del primer procesador.

**Ejercicio 1:**

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

Hemos ejecutado omp1 con un argumento mayor que el número de núcleos que sabemos que tiene nuestro ordenador y hemos visto que el programa genera 10 hilos siendo 4 el número de núcleos de que disponemos, por lo tanto, si es posible lanzar más hilos que núcleos tenga el sistema.

A screenshot of a cell phone

Description automatically generated

Lanzar más hilos que procesadores tiene el sistema generalmente optimiza el tiempo de ejecución de un programa, pero debemos no exceder el punto en el que los procesadores tengan que manejar tantos hilos que el tratamiento de los hilos ralentice la ejecución del propio programa.

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

En el caso ideal, el número óptimo de hilos que debería tener nuestro programa es el número de núcleos del que dispone el sistema que va a ejecutar el programa. En el caso de los ordenadores del laboratorio, que tienen 4 núcleos, en el escenario óptimo funcionarían más rápido con 4 hilos. En el caso del cluster, que tiene 128 núcleos, si este solo se dedicara a ejecutar nuestro programa deberíamos hacer uso de 128 hilos. Y en el caso de nuestro propio equipo, deberíamos ver qué procesador tenemos, buscar el número de núcleos que tiene y ese número sería el optimo.

**Captura de pantalla de la ejecución de omp2:**

A screenshot of a cell phone

Description automatically generated

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

En el caso de las variables privadas en OpenMP, la variable a en el programa, cada hilo guarda la variable en una dirección diferente y la inicializa a 0 como podemos ver en el resultado de la ejecución del programa.

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

El valor de las variables privadas se iguala a cero a no ser que sea del tipo firstprivate en lugar de private, esto permite que una variable privada pueda ser inicializada, pero se comporta de la misma forma en todos los demás casos.

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

El valor de las variables privadas al finalizar la región paralela se iguala al valor que tenían al iniciar esta misma.

**1.6 ¿Ocurre lo mismo con las variables públicas?**

En el caso de las variables públicas el valor inicial es igual al valor que tenían antes de entrar a la región paralela, la dirección para acceder a la variable se comparte entre hilos, lo que implica que las variables originales pueden ser modificadas en la ejecución de los hilos y al finalizar la ejecución de la región paralela, los valores de las variables persisten.

**Ejercicio 2:**

A screenshot of a cell phone

Description automatically generated**Resultado de las ejecuciones de pescalar\_serie, pescalar\_par1, pescalar\_par2:**

**2.1 ¿En qué caso es correcto el resultado?**

Como podemos observar, al ejecutar pescalar\_serie y pescalar\_par2 obtenemos el resultado correcto, mientras que pescalar\_par1 no solo no da el mismo valor que pescalar\_serie, sino que si es ejecutado más de una vez da resultados diferentes.

**2.2 ¿A qué se debe esta diferencia?**

Esto se debe a que en la línea en la que se declara que se va a ejecutar un bucle for con hilos de pescalar\_par1 se escribe:

#pragma omp parallel for

Mientras quen en pescalar\_par2 se escribe:

#pragma omp parallel for reduction(+:sum)

En el caso de pescalar\_par1 es posible que múltiples hilos intenten modificar la misma variable, lo que provoca que el resultado sea erróneo. Mientras que en el caso de pescalar\_par2 se hace uso de la clausula reduction, que permite hacer cálculos de forma recurrente en paralelo sin pisar datos.

**Ejercicio 3:**

**Tabla 1210 x 1210:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versión\#hilos | 1 | 2 | 3 | 4 |
| Serie | 9,697608 |  |  |  |
| Paralela – bucle1 | 10,986696 | 6,351142 | 4,724200 | 3,961497 |
| Paralela – bucle2 | 10,563498 | 5,335582 | 3,572199 | 2,765028 |
| Paralela – bucle3 | 10,324923 | 5,666243 | 3,732129 | 2,839574 |

**Speedup 1210x1210:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versión\#hilos | 1 | 2 | 3 | 4 |
| Serie | 1 |  |  |  |
| Paralela – bucle1 | 0,882668 | 1,526907 | 2,052751 | 2,447965 |
| Paralela – bucle2 | 0,918029 | 1,817535 | 2,714744 | 3,507236 |
| Paralela – bucle3 | 0,939242 | 1,711470 | 2,598411 | 3,415162 |

**Tabla 1950 x 1950:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versión\#hilos | 1 | 2 | 3 | 4 |
| Serie | 61,690951 |  |  |  |
| Paralela – bucle1 | 55,544041 | 27,838593 | 19,992917 | 15,850305 |
| Paralela – bucle2 | 53,595174 | 27,711784 | 18,697823 | 14,419734 |
| Paralela – bucle3 | 55,046644 | 27,848934 | 18,831843 | 14,252818 |

**Speedup 1950 x 1950:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versión\#hilos | 1 | 2 | 3 | 4 |
| Serie | 1 |  |  |  |
| Paralela – bucle1 | 1,110667 | 2,216022 | 3,085640 | 3,892098 |
| Paralela – bucle2 | 1,151054 | 2,226163 | 3,299365 | 4,278230 |
| Paralela – bucle3 | 1,120703 | 2,215199 | 3,275884 | 4,328333 |

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

El que tiene peor rendimiento es el que esta en el bucle mas interno porque se realizan el numero de hilos x el numero de veces que se ejecuta ese bucle, lo que genera un gasto de recursos innecesario.

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

**Ejercicio 4:**

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

En pi\_serie.c se utilizan tantos rectángulos como valor tenga la variable n, la cual guarda el valor 100 000 000.

**4.2 ¿Qué diferencias observa entre estas dos versiones?**

En el fichero pi\_par1.c el bucle en el que cada hilo calcula su valor correspondiente accede a la variable compartida sum en cada iteración, mientras que en el fichero pi\_par4.c cada hilo guarda una suma por su cuenta y accede al final a la variable sum para almacenar su resultado.

**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?**

**Ejecución (desde un ordenador de otro laboratorio que tiene 6 núcleos):**

A screenshot of a cell phone

Description automatically generated

El resultado obtenido es 3.141593 en ambos casos, no hay diferencias. Sin embargo, sí hay diferencias en cuanto al rendimiento. Eso probablemente se deba a que en pi\_par1.c los hilos están accediendo múltiples veces a una variable compartida para realizar sus cálculos, mientras que en pi\_par4.c se van guardando los cálculos en una variable para cada hilo y luego se accede una única vez a la variable compartida. En el caso de pi\_par1.c habrá que dedicar más recursos a manejar la variable compartida lo cual explicaría el aumento de tiempo en el cálculo.

**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?**

**Ejecución:**

A screenshot of a cell phone

Description automatically generated

Los resultados siguen siendo iguales, y en este caso pi\_par2.c parece tener un rendimiento bastante semejante a pi\_par1.c y pi\_par3.c parece haber conseguido un rendimiento semejante a pi\_par4.c.

En el archivo pi\_par2.c el único cambio que se produce con respecto a pi\_par1.c es que sum se vuelve una variable privada, pero esto no debería cambiar el rendimiento ya que se siguen utilizando los mimos recursos y haciendo las mismas operaciones, así que el rendimiento debería ser equivalente.

En el archivo pi\_par4.c se reserva más memoria y los datos correspondientes a cada hilo están mas espaciados, lo que parece permitir que se optimice el manejo de recursos y, por consiguiente, el rendimiento.

**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?**

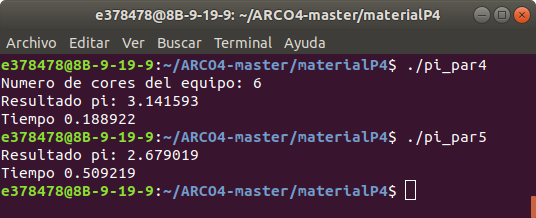
A close up of a map

Description automatically generated**Grafica obtenida de las ejecuciones:**

Como se puede observar, el tiempo de ejecución decrece rápidamente hasta cierto punto en el que se estanca y el rendimiento no mejora más.

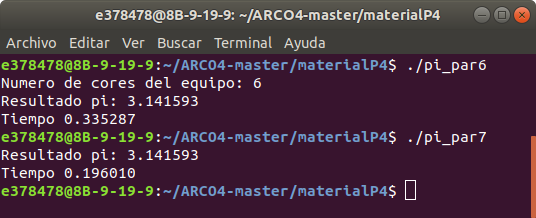
**Ejercicio 5:**

**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?**



El que se ejecuta con la directiva critical es pi\_par5, al usar esta directiva restringimos una parte del código para que solo la pueda ejecutar un hilo al mismo tiempo. En el rendimiento vemos que el tiempo tardado esta en el orden de 3 veces mayor lo cual es mucho y que el número pi no se parece en nada al que debería ser, esto se debe a que se reduce el paralelismo de la ejecución y cada hilo esta mucho tiempo en espera.

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



Ambos usan las directivas parallel default private y en concreto pi\_par7 usa la directiva reduction.

La directiva parallel sirve para iniciar la ejecución paralela con un grupo de hilos, default controla el atributo por defectos de compartición de datos y con private se declaran variables privadas, en pi\_par7 la directiva reduction que hace que cada hilo tenga su propia variable y como dentro de esta directiva pone +=: sum al final de los hilos se suman todas las variables locales en una global.

Gracias a la directiva reduction se ve que el tiempo de ejecución baja ya que esta clausula realiza este problema de forma mucho mas eficiente.