Luis M Benítez Ruiz

994361

Preguntas Practicas OpenMP

Practica 1

¿Cuántas veces se imprime? ¿Por qué?

4 veces, porque mi computadora tiene 4 CPUs y por default openMP ejecuta un thread por cada procesador disponible.

¿Qué versión de OpenMP tienes?

201107

Practica 2

No hay preguntas.

Practica 3

-¿Qué sucede cuando se paraleliza el problema con tamaño de 1000 elementos con la máxima cantidad de hilos que soporta la computadora?

El promedio de tiempo de ejecución es de 0.1155 segundos. Lo cual es muy lento cuando se compara con el siguiente caso

-¿Qué sucede cuando se paraleliza el problema con un tamaño de 1000000 elementos?

El tiempo de ejecución se mantiene bastante estable, el promedio es de 0.0151

-¿Qué sucede si se paraleliza con 2,4, u 8 threads? Especificar el thread desde a variable de entorno. Realiza una tabla comparativa

Threads	Tiempo de ejecución (promedio de 10 ejecuciones) con 10 000 000 de datos.
2	0.124778
4	0.126222
8	0.123222

-¿Qué instrucción se utiliza para especificar la cantidad de hilos dentro del código?

omp_set_num_threads(int numthreads);

-¿Qué sucede si se especifican más threads que cores disponibles en la computadora?

Un core se hará cargo de más de un hilo. openMP se encargara de crear los threads, y el sistema operativo está a cargo de asignar los threads a los cores disponibles y agendar el tiempo disponible para cada thread en los cores.

-Realiza una comparativa en tiempos de ejecución en forma secuencial y paralela utilizando un tamaño del vector de 500000 elementos. Explica el resultado.

Threads	Tiempo de ejecución (promedio de 10 ejecuciones) con 50 000 000 de datos.
1	0.731
4	0.610
16	0.594
300	0.624

En este caso el costo de coordinación de procesos es bajo, así que la ejecución paralela, incluso con 300 hilos (mucho más que los núcleos disponibles) sigue siendo notoriamente más rápida que la ejecución secuencial. Sin embargo se necesito utilizar una gran cantidad de datos, 50 millones, para poder notar claramente las diferencias en tiempos de ejecución.

Otra cosa importante es que cuando se ejecuta el problema con 1000 datos pueden utilizarse hasta 850 threads, mientras que con 50 millones, empieza a fallar esporádicamente a partir de 300. Esto debido a que la cantidad de datos asignados por thread es mayor, y los recursos se terminan más rápido, por lo que no se pueden crear más hilos.

Practica 4

¿Qué sucede si se utiliza la clausula reduction en la paralelizacion del ciclo?

La operación que se indique (suma en este caso) se aplica para 'reunir' todas las copias de la variable que se indique en la misma clausula.

Realiza una comparación entre la versión secuencial y paralela utilizando 2,4,8… dependiendo de la cantidad máxima de hilos disponibles en tu equipo de computo.

Threads	Tiempo de ejecución (promedio de 10 ejecuciones) con vector de 100 000 datos.
1	0.0046
4	0.0063

16	0.0109
300	0.0156

Practica 5

Prueba el ejercicio con al menos cinco tamaños de datos y diferente cantidad de threads. ¿Qué sucede? Realiza una comparativa también con la versión secuencial.

	Tiempo de Ejecucion				
Size / Threads	100	500	5000	10000	20000
1	0.0	0.0047	0.2028	0.8096	3.2307
4	0.0	0.0031	0.2046	0.62772	2.5053
16	0.0	0.0	0.2059	0.6303	2.5099
300	0.0061	0.0077	0.1591	0.6256	Fallo, no hay recursos suficientes.

El total de datos procesados en realidad es size+size*size. Asi que en el caso de 20 000, se están procesando 400 0020 000 datos (400 millones 20 mil).

Puede observarse que el número de threads mejora el rendimiento para tamaños grandes de datos, y que no hay mucha diferencia entre el número de threads utilizados, excepto en casos extremos cuando se terminan los recursos y no se puede ejecutar el programa.

Practica 6

Realiza una comparativa de la versión secuencial con la versión paralela utilizando diferente cantidad de threads.

Threads	Tiempo de ejecución (promedio de 10 ejecuciones) con matrices de 200x200 de datos.
1	0.0827

4	0.1263
16	0.1216
300	0.468

Contesta las siguientes preguntas:

1. A que se refiere la variable *shared* dentro de la zona paralelizable.
   Indica que las variables incluidas dentro de shared() serán compartidas entre todos los threads.
2. A que se refiere la variable *private* dentro de la zona paralelizable.
   Indica que las variables incluidas dentro de private() serán privadas, es decir, cada thread tendrá una copia propia de la variable.
3. Que hace la siguiente instrucción:
   a. #pragma omp parallel num_threads(4);

   Indica que el siguiente bloque de codigo entre { } se ejecutara en paralelo utilizando 4 threads.