Laboratorio 04

Instrucciones.

Esta actividad se realizará individualmente. Al finalizar los períodos de laboratorio o clase, deberá dejar constancia de sus avances en Canvas, según indique su catedrático. Cuidar la limpieza y orden de su código.

Parte 1 (15 puntos).

Explicar con sus propias palabras las siguientes cláusulas/directivas, qué tipo de datos devuelven y en qué casos recomienda utilizarlos:

1. **num_thread():** Esta función devuelve un entero que indica el número total de threads utilizados en una región paralela. Es útil cuando se necesita saber cuántos threads están activos en un bloque de código paralelo, lo que te permite ajustar dinámicamente la carga de trabajo o tomar decisiones basadas en el número de threads.
2. **omp_set_num_threads():** Es una función que no devuelve ningún valor y se utiliza para establecer manualmente el número de threads que se usarán en las regiones paralelas. Se recomienda utilizarla antes de entrar en una región paralela si se desea controlar específicamente cuántos threads se deben utilizar, en lugar de dejar que el entorno lo decida automáticamente.
3. **omp_get_thread_num():** Devuelve un entero que representa el identificador del thread actual que está ejecutando el código dentro de una región paralela. Es útil cuando se necesita identificar qué thread está ejecutando una parte específica del código, lo que puede ser importante para asignar tareas o para propósitos de depuración.
4. **omp_get_wtime():** Devuelve un valor de tipo double que representa el tiempo en segundos desde un punto en el pasado y se usa para medir el tiempo de ejecución de bloques de código. Es recomendable para medir y comparar el rendimiento de diferentes implementaciones de regiones de código paralelas, lo que ayuda en la optimización del programa.
5. **Nowait:** Es una cláusula que no devuelve ningún valor, sino que controla el flujo dentro de construcciones paralelas en OpenMP, permitiendo que los threads continúen con la ejecución del código sin esperar a que otros threads terminen su parte del trabajo. Se recomienda cuando no es necesario sincronizar threads al final de una construcción paralela, lo que puede mejorar la eficiencia en ciertos casos.

Parte 2 (18 puntos).

Desarrollar las respuestas con sus propias palabras:

1. ¿Qué es el modelo fork-join?
   E un esquema de paralelismo donde un programa comienza con un solo hilo que en algún punto del código se crean múltiples hilos de ejecución a partir del primer hilo creado, que pueden trabajar en paralelo. Cada uno de estos hilos realiza su tarea y al finalizar se reúnen de nuevo en un único hilo, regresando al primer hilo.

2. ¿Si en un equipo hay 64 hilos derivados, cuál sería el total de hilos? Por qué?
   El total de hilos sería 65. Esto se debe a que, además de los 64 hilos derivados que se crean para ejecutar tareas en paralelo, siempre existe un hilo padre que coordina y controla el proceso. Por lo tanto, sumando el hilo maestro a los 64 hilos derivados el total es 65.

3. ¿Por qué se dice que los hilos que se crean en OpenMP son dinámicos?
   Son dinámicos porque el número de hilos puede variar durante la ejecución del programa dependiendo de las condiciones del sistema y de las configuraciones que el programador establezca.

4. ¿Cuál es la función de un pragma?
   Un pragma es una directiva que se inserta en el código fuente para indicar cómo se debe paralelizar una determinada sección de código. Esto permite que el programador decida qué partes del código deben ejecutarse en paralelo, cuántos hilos se deben utilizar o cómo se deben gestionar las variables compartidas entre los hilos.

5. ¿Cuáles son las diferencias entre single y sections?
   Single se utiliza para que una sección específica del código sea ejecutada por un único hilo, mientras que el resto de los hilos esperan a que ese hilo termine su trabajo. En cambio, sections permite dividir el trabajo en diferentes bloques donde cada bloque es ejecutado por un hilo diferente.

6. Al utilizar sections, ¿qué ocurre si hay más secciones que hilos disponibles? ¿qué ocurre si hay más hilos disponibles que secciones?
   Si hay más secciones que hilos disponibles se distribuirá las secciones entre los hilos de manera que algunos hilos puedan ejecutar más de una sección en secuencia. Si hay más hilos disponibles que secciones, algunos hilos quedarán inactivos.

Parte 3 (15 puntos).

Analice el código dp_co.c:

- Resuelva los errores del código y explique cómo por qué se genera cada error y cómo lo corrigió.
  - Declaración incorrecta de los arrays A y B en la función dot_product. Los parámetros A y B están declarados como enteros pero deberían ser punteros a enteros porque representan arrays. La corrección es cambiar la firma de la función a void dot_product(int *A, int *B, int n).

  - Hay un carácter extra “&” al final de la línea donde se están creando tareas paralelas, lo cual es un error de sintaxis. La corrección consiste en eliminar el carácter “&” al final de la línea.

  - La directiva está mal escrita como #pragma omp paralel; debería ser #pragma omp parallel.

  - La función rec_dot_product necesita un parámetro adicional depth para la recursión pero en main la llamada inicial no lo está pasando. La corrección es modificar la llamada a rec_dot_product(a, b, N, 0); para iniciar con una profundidad de 0.

- Explique el funcionamiento del programa.
  El programa calcula el producto punto de dos vectores utilizando paralelismo con OpenMP. Primero divide recursivamente los vectores en subpartes más pequeñas. Si las subpartes son grandes crea

tareas paralelas para calcular su producto. Las subpartes pequeñas se calculan directamente sumando los productos de los elementos correspondientes y acumulando el resultado en una variable global. Al final el programa imprime el producto punto calculado.

- Indique, ¿qué sucede si modifica el valor de CUTOFF a 2, 3, y 4?
  Al cambiar el valor de CUTOFF se está ajustando cuándo el programa empieza a paralelizar el cálculo en lugar de continuar con la recursión secuencial. Un valor bajo de CUTOFF permite más paralelismo desde etapas tempranas lo que puede ser eficiente para aprovechar los núcleos disponibles, pero podría generar sobrecarga si hay demasiadas tareas pequeñas. Un valor más alto profundiza más la recursión antes de paralelizar lo que puede reducir la cantidad de tareas pero también aprovechar menos el paralelismo haciendo que el programa pueda ser menos eficiente en sistemas con muchos núcleos.

Parte 4 (42 puntos).

Escriba un código en C que realice lo siguiente:

1. Que se pida a usuario que indique si quiere trabajar con 8, 16 o 32 hilos.
2. Debe de crear un array llamado “x” del tamaño de hilos que se especificaron y debe de guardar números pares que inicien en el número 2 y vayan en aumento. Ej.: 2, 4, 6, 8, 10,12…
3. Para cada uno de los números, utilice la siguiente función para sustituir el valor de “x” y encontrar “y”:
   Y= (½)a+x^2 donde a=5.
   Este proceso debe de realizarse en paralelo.
4. Para cada resultado, guardar el valor en un array “y”. Este proceso debe de reallizarse en paralelo.
5. Imprimir una tabla de resultados con el siguiente esquema:

x	y
2	
4	
6	

. . .