Práctica 4. Introducción a OpenMP.

Objetivos de la práctica:

- Adquirir la capacidad de programar en lenguaje C con la librería OpenMP.
- Adquirir la capacidad de resolución de problemas paralelos sencillos.
- Comprender el uso y funcionamiento de las directivas de la librería OpenMP.
- Comprender cómo funciona un programa multihilo.
- Adquirir la capacidad de paralelizar un programa.

1 Introducción

Como hemos visto en clase, para poder paralelizar un programa con la librería OpenMP añadiremos la librería en el código fuente:

`#include<omp.h>`

Para compilar un programa que contenga directivas de OpenMP el comando es:

```
gcc -fopenmp -o salida fuente.c
```

Y luego se ejecuta normalmente:

```
./salida
```

En esta práctica vamos a repasar las que hemos visto en prácticas anteriores y vamos a utilizar también la clausula `schedule`. Las claúsulas vistas hasta el momento son:

- `num_threads(n)` especifica el grado de concurrencia (número de hilos en la región paralela)
- `if(expresion)` condiciona la paralelización. Si la expresión es falsa, crea el entorno para 1 hilo
- Manejo de datos:
  - `private(variables)` cada hilo tiene una copia de la variable
  - `firstprivate(variables)` cada hilo tiene una copia de la variable y se debe inicializar con el valor de la variable (existe antes de la región paralela)
  - `lastprivate(variables)` cada hilo tiene una copia de la variable y al final de la ejecución tiene el valor del hilo que ejecutó la última iteración
  - `shared(variables)` todos los hilos comparten la variable

  - `default(shared/private/none)` especifica el tipo por defecto de las variables. Si es none, el tipo de todas las variables se tiene que especificar. Por defecto son compartidas (shared)

- `reduction(operador:variables)` combina las variables de una lista usando los operadores (+, *, -, &, |, ^, &&, ||)

La clausula `schedule` va acompañada de las directivas `for` y nos permite especificar cómo se asignan las iteraciones de los bucles. Su formato es `schedule(type [,size])`. Y los tipos son:

- `static`. El trabajo de la iteracción se divide en bloques del tamaño especificado y se asignan los procesos usando round-robin.
  - Si no se especifica `size`, se dividen en tantos bloques como hilos (1 bloque, 1 hilo).
  - El comportamiento por defecto es: `schedule(static)`
- `dynamic`. El trabajo de la iteracción se divide en bloques del tamaño especificado y se asignan a los hilos según acaban su trabajo. Evita cargas no balanceadas de trabajo
  - Si no se especifica `size`, se dividen en tantos bloques como hilos (1 bloque, 1 hilo).
- `guided`. El tamaño del bloque se reduce según disminuyen las iteraciones que no se han distribuido.
  - El tamaño mínimo de bloque es `size`.
- `runtime`. La asignación se postpone hasta el momento de ejecución.
  - `OMP_SCHEDULE o omp_set_schedule()` establecen el tipo de asignación

Cuando un reparto de trabajo tiene la claúsula `nowait`, se elimina la barrera implítica al final del bucle

- Ejemplo de bucles independientes que se pueden solapar

```
#pragma omp for nowait
for (i=0;i<n;i++)
  v[i]=0;
#pragma omp for
for (i=0;i<n;i++)
  w[i]=0;
```

La directiva **collapse** se utiliza para combinar múltiples bucles anidados en uno solo para la paralelización. Los bucles tienen que ser independientes entre ellos. Esto es útil cuando se trabaja con bucles anidados y se desea paralelizarlos de manera conjunta para mejorar la eficiencia y reducir la sobrecarga de paralelización. Por ejemplo, si tienes dos bucles anidados y quieres paralelizarlos, puedes usar collapse para combinarlos en un solo bucle para una paralelización más efectiva.

– Ejemplo de bucles anidados que se pueden colapsar

```
#pragma omp parallel for collapse(2)
for (i = 0; i < n; ++i) {
  for (j = 0; j < m; ++j) {
  }
}
```

2 Ejercicios propuestos

2.1 Ejercicio con schedule y vectores

Escribe un programa en C que cree un vector de tamaño n, siendo n un argumento del main. Realice un bucle en paralelo que inicialice el vector con el identifcador del hilo que ha ejecutado su posición. Después imprime el vector para ver como ha repartido el trabajo. Después prueba las distintas configuraciones de schedule

2.2 Ejercicios con schedule y matrices

Escribe un programa en C que cree una matriz de tamaño n*m, siendo argumentos del main. Realice un bucle en paralelo que inicialice la matriz con el identifcador del hilo que ha ejecutado su posición. Después imprime la matriz para ver como ha repartido el trabajo. Prueba a hacerlo con y sin collapse. Después prueba las distintas configuraciones de schedule

2.3 Ejercicio de repaso

Realice un programa `primos.c` en lenguaje C que calcule cuántos primos hay en el intervalo 1 a n repartiendo las iteraciones entre distintos hilos de forma que se ejecuten en paralelo.

2.4 Ejercicio trabajos independientes

Realice un programa en C que solicite n, siendo n el número de alumnos por lo tanto tiene que ser un número positivo. Crea dos vectores de tamaño n que corresponden a las calificaciones de los alumnos en el primer parcial (vector1) y el segundo parcial (vector2) con número aleatorio entre 0 y 10 (incluidos). Calcula la media, mínimo y máximo de cada parcial. ¿Son los bucles dependientes o independientes? Utiliza nowait cuando sea posible