# Práctica 3. 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
```

De las directivas vistas en clase, en esta práctica vamos a trabajar con la directiva `for` y la directiva `parallel for` la cual distribuye las iteraciones de un bucle for entre los hilos existentes. Se puede emplear de dos formas:

- Lanzar la directiva `parallel` y dentro la directiva `for`
  - Ejemplo:

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

- Englobar la directiva `parallel` y la directiva `for` con la directiva `parallel for`:

- Ejemplo:

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

Las `clauses` que se pueden usar, entre otras, son `private`, `firstprivate`, `shared`.

Cuando tengamos instrucciones dentro del bucle que hagan operaciones como `total = total + v[i];` se puede indicar que se sumen todos los resultados calculados por cada hilo con `reduction(+: total)`. El primer valor es la operación que puede ser: `+, -, *, |, ||, &, &&, ^, min, max`. El compilador se encarga de crear una copia de la variable para cada thread y después de hacer la operación indicada en una variable compartida.

En caso de que todos los hilos necesiten acceder a la misma sección de código se necesitan utilizar secciones `atomic` o `critical`. La principal diferencia es que `atomic` se utiliza para serializar una única operación y es más rápido. Mientras que `critical` permite serializar varias instrucciones, pero es más lento.

Si se necesita paralelizar varios bucles anidados se utiliza `collapse`.

– Ejemplo

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

# 2. Ejercicios propuestos

## 2.1. Ejercicio 1

– Realice un programa `suma_elementos.c` en lenguaje C que lea por argumentos el tamaño de un vector y que reserve espacio utilizando punteros. Después inicializará el vector con números aleatorio entre 1 y 100 paralelizando el bucle. Después haz la media de los elementos del vector paralelizando el bucle. Mide los tiempos de paralelizar el bucle con `reduction` y con `atomic/critical`.

## 2.2. Ejercicio 2

– Realice un programa `suma.c` en lenguaje C que calcule la suma de todos los elementos de una matriz n*n paralelizando los bucles que se puedan paralelizar. Utiliza punteros. Compara el tiempo que tarda el programa en ejecutarse para distinto tamaño de la matriz (500x500, 100x100, etc) y distinto número de hilos. NOTA: la matriz estará inicializada con números aleatorios entre 1 y 10.

## 2.3. Ejercicio 3

– Realice un programa `producto.c` en lenguaje C que calcule el producto de dos matrices A y B paralelizando los bucles que se puedan paralelizar. Utiliza punteros. Compara el tiempo que tarda el programa en ejecutarse para distintos tamaños de la matriz (500x500, 100x100, etc) y distinto número de hilos. NOTA: las matrices estarán inicializadas con números aleatorios entre 1 y 10.