

# 1. Herramientas, estilos y estructuras de programación paralela

## 1. Problemas que plantea la programación paralela

Es evidente que, como mínimo, presenta los problemas de la programación secuencial a los que se añaden:

- **División** en unidades de trabajo independientes,
- **Agrupación** o asignación de **tareas** o cargas de trabajo en procesos libres.
- **Asignación** a **procedimientos** o **núcleos**.
- **Sincronización** y **cooperación**

## Estos problemas deben ser abordados por el programador si quiere tener una solución óptima o serán abordados en su defecto por la **herramienta de programación** o se occasionando que esta solución no sea la óptima.

## 2. Punto de partida

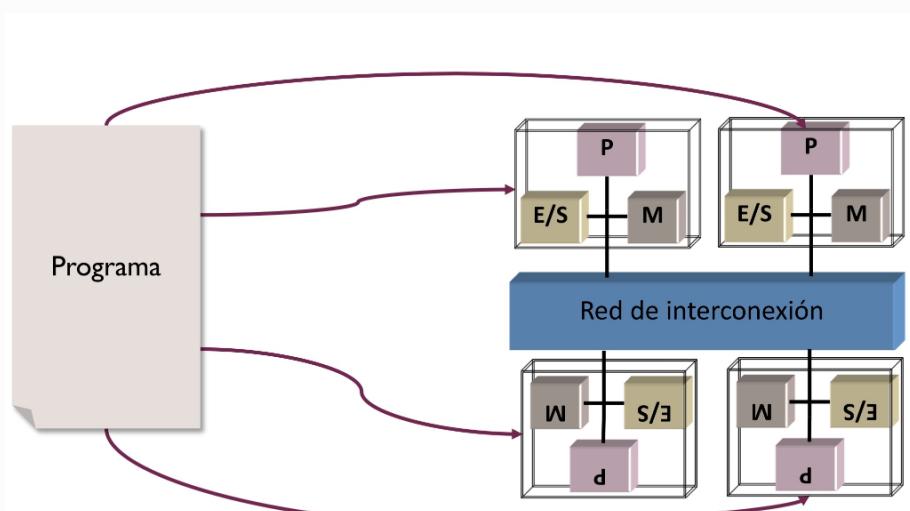
En ocasiones, partiremos de una **versión secuencial** teniendo ya una idea general del programa. O en ocasiones, comenzaremos simplemente con una **descripción del funcionamiento del mismo**.

Como apoyo podremos tener un **programa paralelo** que resuelva el problema sencillo. **Bibliotecas de funciones**

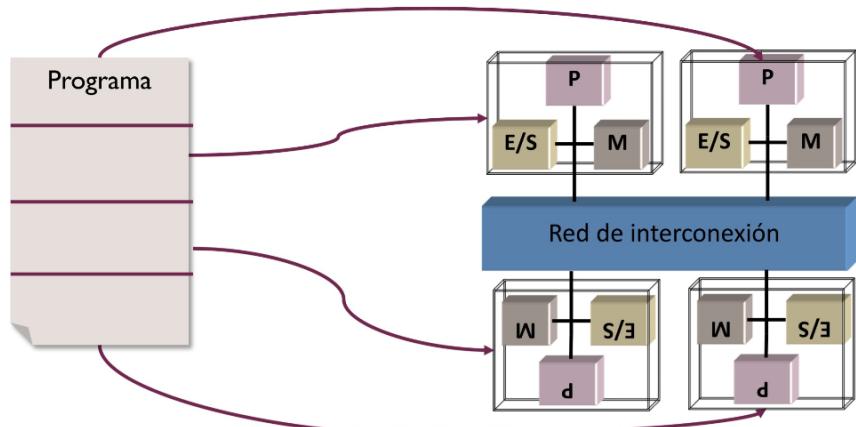
## 3. Modos de programación SIMD

Hay dos modos:

- **SPMD**: Consiste en ejecutar el mismo programa sobre varios procedimientos con datos distintos.



→ **MPMD**: Consiste en que el programa de partida se divide en subprogramas y que cada subprograma se ejecuta en su modo de ejecución distinto. Es posible conseguir, a base de modificaciones en el código, que se ejecute todo un SPMD.



#### 4. Herramientas para obtener código paralelo

Estas herramientas permiten de forma implícita o explícita una serie de cuestiones:

- Localizar paralelismo o descomponer en tareas independientes.
- Asignar las tareas creadas a procesos.
- Crear y terminar procesos o hilos.
- Coordinar y sincronizar procesos o hilos.

Por otra parte, el programador, el entorno o el Sistema Operativo se encarga de asignar procesos o hilos a unidades de procesamiento; es decir, de hacer el mapa.

#### 5. Comunicaciones colectivas

Uno a todos	Todos a uno	Todos a todos	Multiples uno a uno	Compartidos
<u>Difusión</u> 	<u>Reducción</u> 	<u>Todos a todos</u> 	<u>Permutaciones</u> 	<u>Todos combinando</u> 
<u>Dispersión</u> 	<u>Acumulación</u> 	<u>Todos dispersan</u> 	<u>Respalzamientos</u> 	<u>Barreras</u> 

## 6. Estilos o paradigmas de programación paralelo

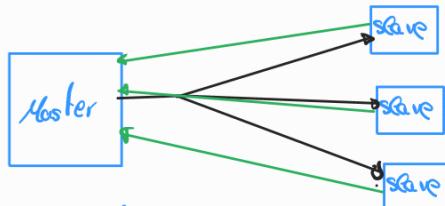
Hay tres estilos o herramientas de programación:

- 1) **Paso de mensajes** → lenguajes de programación o bibliotecas de funciones.
- 2) **Variables compartidas** → lenguajes de programación, bibliotecas de funciones o directivas del compilador.
- 3) **Paralelismo de datos** → lenguajes de programación o directivas del compilador.

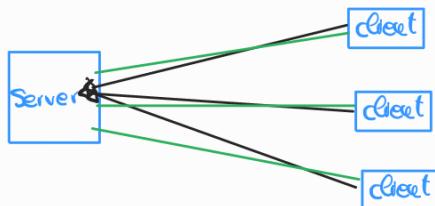
## 7. Estructuras típicas

Hay varias estructuras para la implementación de código paralelo, veremos tres de ellas:

1. Master-slave. Un proceso "maestro" crea y mantiene a su disposición varios procesos "esclavos"; los cuales ejecutan todos el mismo código impuesto por el maestro y le devuelven a este los resultados siendo el maestro el encargado de combinarlos.



- ) Servel-client. Es similar al anterior pero con el funcionamiento contrario, es decir, los clientes realizan una petición al servidor y este devuelve su respuesta.



- ) Divido y vencerás. Consiste en la división del dominio de datos entre los distintos programas que ejecutan el mismo código para después combinar las soluciones y obtener la solución al problema.

En este caso, la asignación de flujos de instrucciones debe ser estudiada en función de la red de interconexión del computador pues cuando tenemos flujos largos mejor limpieza habrá.

## 2. Proceso de paralelización

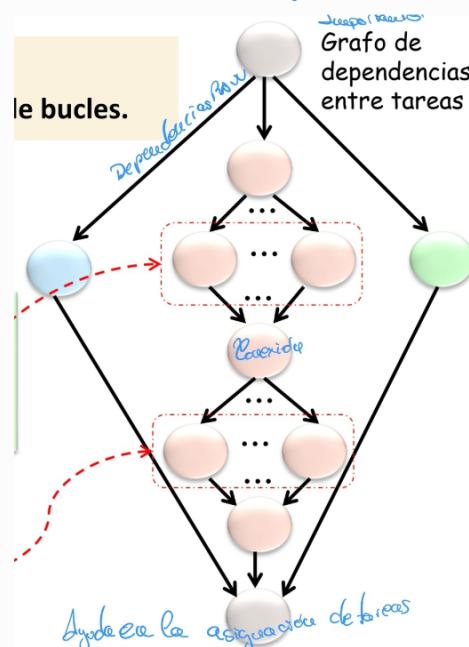
El proceso de paralelización se divide en cuatro partes:

1. Descomposición de tareas
2. Asignación de tareas
3. Reducción de código paralelo
4. Evaluación de prestaciones

### 1. Descomposición de tareas

Normalmente, este paso consiste en bulear las ideas principales del problema e implementarlas por separado de manera que sea más fácil encontrar las dependencias entre funciones y entre bucles.

Como resultado final se debe elaborar un grafo de dependencias entre tareas.



### 2. Asignación de tareas

Este paso consiste en agrupar tareas en procesos y maplar los procedimientos. Es decir, entre otras cosas, hacer que llega el mismo flujo de instrucciones en la red de interconexión.

Para ello, debemos tener en cuenta la granularidad de la carga de trabajo asignada a los procesos; es decir, cuanto de grande es el trabajo asignado. Esto depende del número de cores y de la proporción tiempo de comunicación entre bucle de cálculo.

Además, también tendremos en cuenta el equilibrio de la carga buscando que no haya espera entre procesos. Esto depende de la arquitectura, siendo mejor la homogénea, pues no necesita un gran control de la sincronización, y la uniforme frente a la no uniforme, y de la descomposición de tareas.

Hay varios tipos de asignación:

- . Estática. Sesale cuando y quién va a ejecutar la tarea. La puede hacer el programador o la hará el compilador.

Dinámica. No es posible saber con determinación cuándo y qué va a ejecutar una tarea pues dicha asignación se hace en tiempo de ejecución. Al igual que en la anterior, puede ser realizada por el programador o el computador.

Es importante aclarar que, dinámica significa "en tiempo de ejecución".

Por último, el S.O. es el encargado del manejo con ayuda del entorno e influencia del programador si se quiere.

### 3. Redacción de código paralelo

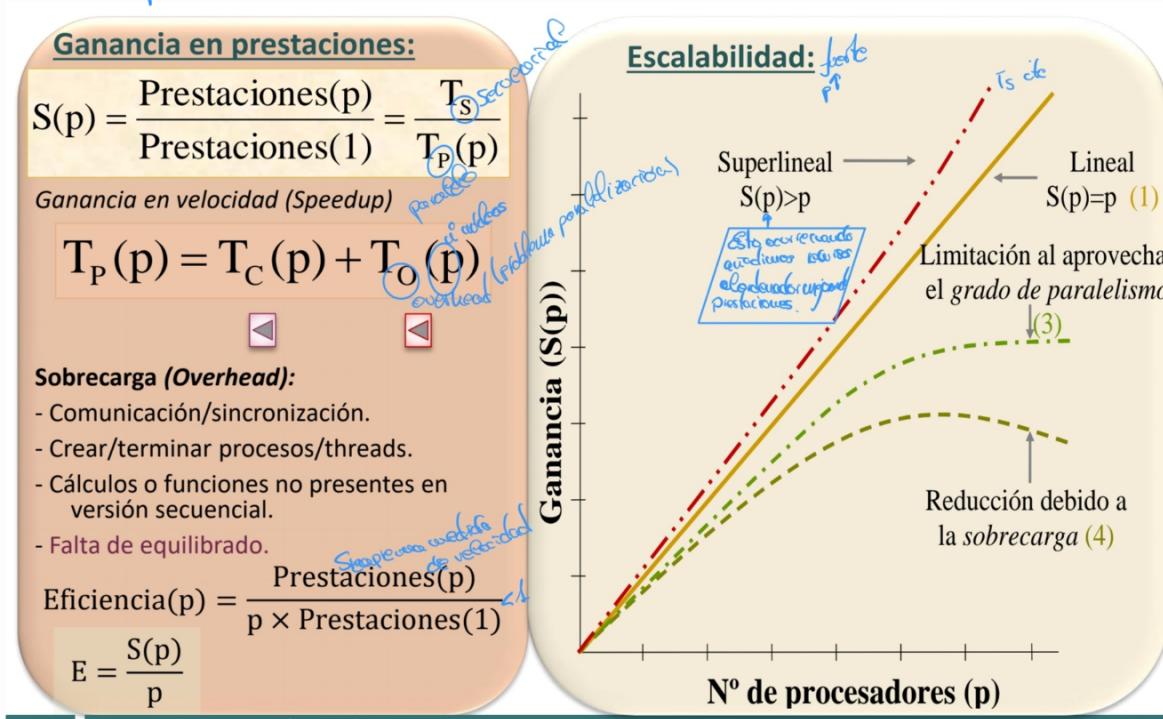
Simplemente consiste en aplicar los conceptos aprendidos en el Seminario 1 y 2 usando la herramienta OpenMP.

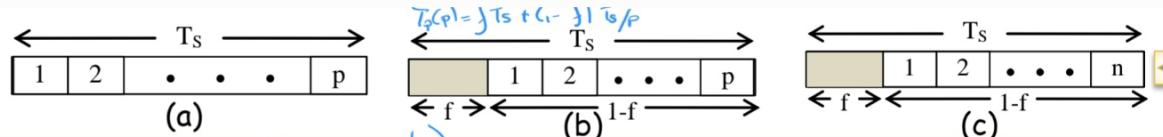
## 3. Evaluación de prestaciones (4)

### 1. Ganancia en prestaciones y escalabilidad

Consiste en estudiar la proporcionalidad entre el tiempo obtenido al paralelizar usando, usualmente, el tiempo de respuesta, ya sea real o del sistema con la orden time, y la productividad medida habitualmente por las Mflops.

Además, podemos medir la escalabilidad y la eficiencia siendo esta última la más intuitiva por su relación con el rendimiento.





Modelo código	Fracción no paralel. en $T_S$	Grado de paralelismo	Overhead	Ganancia en función del número de procesadores $p$ con $T_S$ constante
(a)	0	ilimitado	0	$S(p) = \frac{T_S}{T_p(p)} = p$ Ganancia lineal (1)
(b)	f	ilimitado	0	$S(p) = \frac{1}{f + \frac{(1-f)}{p}} \xrightarrow{p \rightarrow \infty} \frac{1}{f}$ (2)
(c)	f	n	0	$S(p) = \frac{1}{f + \frac{(1-f)}{p}} \xrightarrow{p=n} \frac{1}{f + \frac{(1-f)}{n}}$ (3)
(b)	f	ilimitado	Incrementa linealmente con $p$	$S(p) = \frac{1}{f + \frac{(1-f)}{p} + \frac{T_O(p)}{T_S}} \xrightarrow{p \rightarrow \infty} 0$ (4)

## 2. Ley de Amdahl's Law

Esta ley lo que nos dice es que dado un código con paralelización, la ganancia en prestaciones con  $p \in \mathbb{N}$  procesadores,  $S(p)$ , está limitada por la porción de código sin paralelizar,  $f$ . Por tanto:

$$S(p) = \frac{T_S}{T_p(p)} \leq \frac{T_S}{fT_S + \frac{(1-f)T_S}{p}} = \frac{p}{1+f(p-1)}$$

Entonces:

$$\lim_{p \rightarrow \infty} S(p) = \lim_{p \rightarrow \infty} \frac{p}{1+f(p-1)} \stackrel{(H)}{=} \lim_{p \rightarrow \infty} \frac{1}{\frac{1}{p} + f} = \frac{1}{f}$$

Donde en (H) se aplicó la regla de l'Hopital.

## 3. Ley de Gustafson-Barsis

A diferencia de la ley de Amdahl que mantiene constante  $T_S$ , Gustafson-Barsis mantiene constante  $T_p$ ; por tanto y bajo la misma situación y notación:

$$S(p) = \frac{T_S (1-fp)}{T_p} = \frac{fT_p + p(1-f)T_p}{T_p} = f + p(1-f)$$