

# Principios Generales de la Concurrencia

## Tema 1 - Programación Concurrente y de Tiempo Real

Antonio J. Tomeu<sup>1</sup>   Manuel Francisco<sup>2</sup>

<sup>1</sup>Departamento de Ingeniería Informática  
Universidad de Cádiz

<sup>2</sup>Departamento de CC. de la Computación e I.A.  
Universidad de Granada

PCTR, 2019

# Contenido

1. Concepto de Concurrencia.
2. Por Qué Utilizar la Concurrencia.
3. Exclusión Mutua y Sincronización.
4. Consideraciones Sobre el Hardware.
5. Sistemas Distribuidos.
6. Sistemas de Tiempo Real.

# ¿Por Qué Utilizar Concurrency?

- ▶ El hardware es cada vez más paralelo
  - ▶ Procesadores multinúcleo.
  - ▶ Uso de GPU para cálculos masivos
  - ▶ Uso de *Clusters* de procesadores

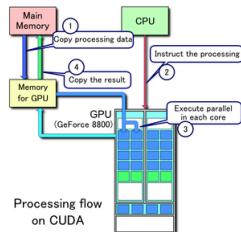
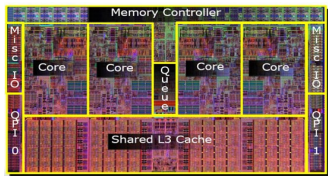
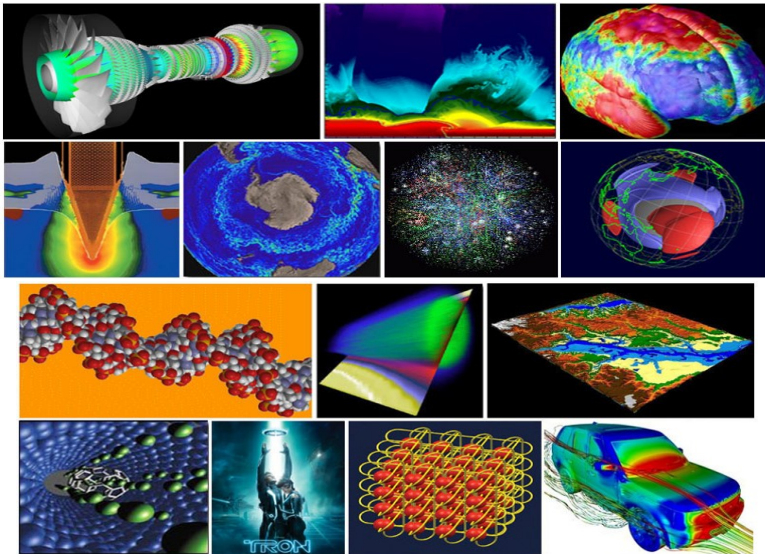


Figura: Intel i7 QuadCore (Nehalem) y Gráfica nVidia GPU (CUDA)

# ¿Cuándo Utilizar Concurrency?



## Definición de la RAE

Coincidencia, concurso simultáneo de varias circunstancias.

- ▶ En informática, se da cuando dos o más procesos **existen** simultáneamente.
- ▶ Distinguir entre existencia y ejecución simultánea.

## Definición

Un **proceso** es un programa en ejecución.

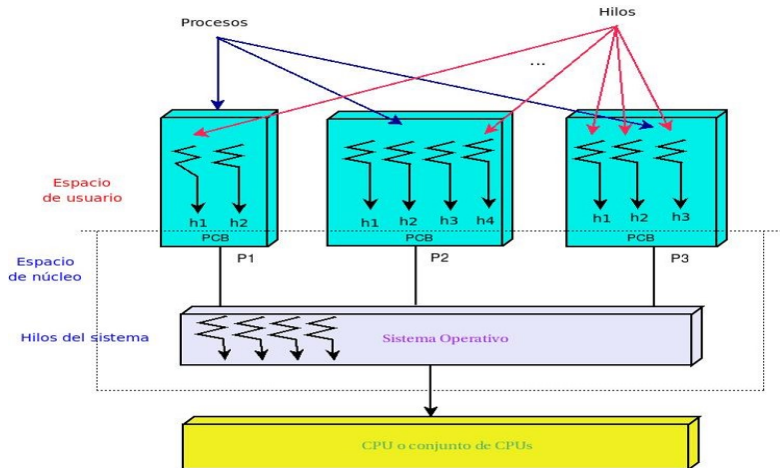
- ▶ Está formado por:
  - ▶ El código que ejecuta el procesador.
  - ▶ El estado (valores registros CPU).
  - ▶ La pila de llamadas.
  - ▶ La memoria de trabajo.
  - ▶ Información de la planificación.

## Definición

Un **hebra** es una secuencia de instrucciones dentro de un proceso, que ejecuta sus instrucciones de forma independiente.

- ▶ Un proceso puede tener varios hebras.
- ▶ Los hebras de un proceso no comparten
  - ▶ La pila (cada una posee la suya).
  - ▶ El contador de programa (ídem).
- ▶ Los hebras de un proceso comparten
  - ▶ Memoria de proceso (datos).
  - ▶ Recursos de sistema.

# hebra vs. Procesos: Idealización Gráfica





# Concurrencia vs. Paralelismo

- ▶ La concurrencia es más general que el paralelismo.
- ▶ Las soluciones que la P. Concurrente da para la sincronización y comunicación son válidas para programas paralelos con memoria común.
- ▶ Se tiene paralelismo cuando se produce la ejecución simultánea de dos procesos concurrentes.
  - ▶ Algoritmos paralelos.
  - ▶ Métodos de programación paralela.
  - ▶ Procesamiento paralelo.
  - ▶ Arquitectura paralela (procesador multinúcleo).

# Concurrencia vs. Paralelismo: Gráficamente

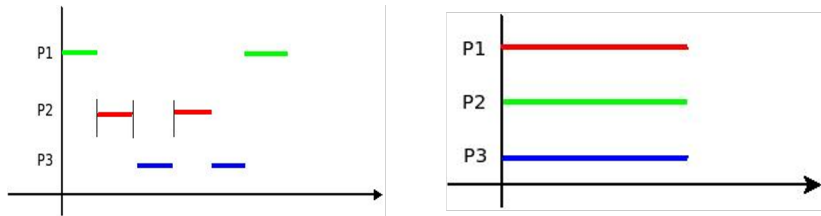


Figura: Discrimine entre concurrencia y paralelismo

- ▶ En el dominio de los problemas: casos reales que se pretenden modelar mediante programas.
- ▶ En el *hardware*:
  - ▶ Ejecución paralela.
  - ▶ Funcionamiento paralelo de periféricos.
  - ▶ Multiprocesadores y procesadores **multinúcleo**.
  - ▶ Sistemas distribuidos.
- ▶ En los sistemas operativos y el *software*:
  - ▶ Ejecución simultánea de procesos.
  - ▶ **Ejecución simultánea de *threads* dentro de un proceso.**

## Definición

Sistema informático donde la concurrencia desempeña un papel importante.

- ▶ Ejemplos:
  - ▶ Sistemas operativos.
  - ▶ Sistemas de gestión de bases de datos.
  - ▶ Sistemas en tiempo real.
  - ▶ Sistemas distribuidos.
  - ▶ Y prácticamente casi cualquier aplicación moderna con GUI decente.
- ▶ Distinguir entre:
  - ▶ Sistemas inherentemente concurrentes.
  - ▶ Sistemas potencialmente concurrentes.

## Definición

**Conjunto de notaciones y técnicas** utilizadas para describir mediante programas el paralelismo potencial de los problemas, resolviendo los problemas de sincronización y comunicación que pueden plantearse.

- ▶ Se ocupa: del análisis, diseño, implementación y depuración de programas concurrentes.
- ▶ No se ocupa: del *hardware* sobre el cual dichos programas concurrentes se ejecutan, ni del lenguaje concreto con el que implementar la concurrencia.

# Características de la Programación Concurrente I

- ▶ Mejora del rendimiento
- ▶ Indeterminación
  - ▶ Un programa secuencial es determinista: los mismos datos de entrada generan **siempre** la misma salida (orden total).
  - ▶ Un programa concurrente no es determinista: la misma entrada puede dar lugar a **diferentes** salidas (orden parcial).
  - ▶ Esto no significa que el programa sea incorrecto.
- ▶ No atomicidad.

# Características de la Programación Concurrente II

- ▶ Velocidad de ejecución de procesos desconocida.
- ▶ Incertidumbre sobre el resultado.
- ▶ Entrelazado
  - ▶ Un programa concurrente puede tener varias secuencias de ejecución diferentes.
  - ▶ Puede haber entrelazados patológicos que lleven a interbloqueo.
- ▶ Mayor dificultad de verificación de programas.
  - ▶ ¿Cuál es la definición de corrección de un programa?

## Ejemplo

Problema: encontrar el número de primos en un rango dado de números naturales. Por ejemplo, en el rango 2-10 hay cuatro primos: 2, 3, 5 y 7.

Rango de interés: 10000000.

- Descargar: `primosSecuencial.java`, `tareaPrimos.java` y `primosParalelos.java`



- ▶ Solución secuencial: 664579 primos en 52 segundos.
- ▶ Solución Concurrente/Paralela: 664579 primos en 33 segundos (con Intel Pentium Core2 Duo).
- ▶ Solución Concurrente/Paralela 2: 664579 primos en 13 segundos (con Intel Core i3 QuadCore).
- ▶ Mejora del rendimiento: 37 % y 75 % respectivamente.

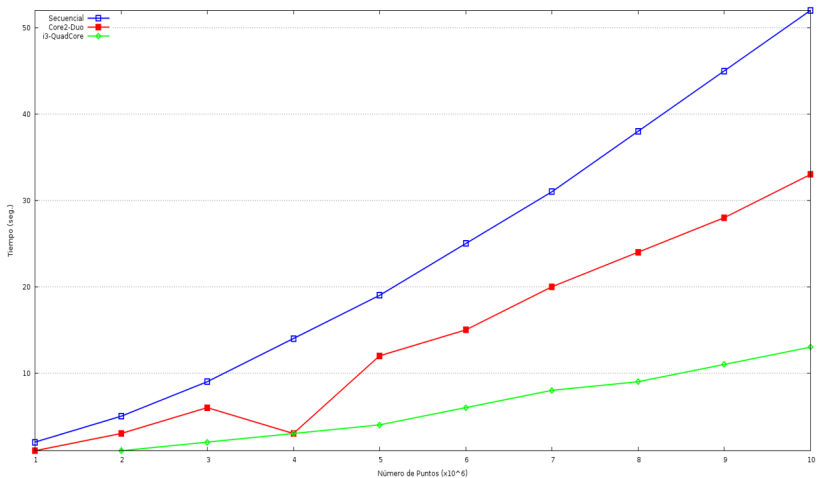


Figura: Mejora del rendimiento: secuencial vs. paralelo

## Definición

En paralelismo, podemos definir el concepto de *speedup* como el coeficiente entre el tiempo necesario para completar una tarea secuencialmente y el tiempo necesario para hacerlo en paralelo.

$$S = \frac{\text{tiempo en secuencial}}{\text{tiempo en paralelo}}$$

## Convolución de una imagen

Tamaño de imagen: 1000x1000 px

- ▶ Tiempos en un Intel i5 @1,9GHz (2 núcleos con HT)
  - ▶ Secuencial: 70 milisegundos.
  - ▶ Paralelo: 36 milisegundos.
- ▶ Speedup:  $\frac{70}{36} = 1,94$

## Convolución de una imagen

Tamaño de imagen: 1500x1500 px

- ▶ Tiempos en un Intel i5 @1,9GHz (2 núcleos con HT)
  - ▶ Secuencial: 153 milisegundos.
  - ▶ Paralelo: 81 milisegundos.
- ▶ Speedup:  $\frac{153}{81} = 1,89$

# Clasificación de Speedup

1. Si  $S < 1$ , la paralelización ha hecho que la solución empeore. En este caso, mejor no paralelizar.
2. Si  $1 < S \leq \text{número de núcleos}$ , hemos conseguido mejorar la solución en un rango normal. En tareas puramente de cómputo (coeficiente de bloqueo muy cercano a 0),  $S$  debe estar muy próximo al citado número de núcleos.
3. Si  $S > \text{número de núcleos}$ , la mejora obtenida ha sido *hiperlineal*. Son casos muy poco comunes y solo se dan para determinados problemas ejecutándose en determinadas arquitecturas de memoria caché.

## Números primos

Primos en el rango 0-10000000

- ▶ Tiempos en un Intel i5 @1,9GHz (2 núcleos con HT)
  - ▶ Secuencial: 18670 milisegundos.
  - ▶ Paralelo: 8968 milisegundos.
- ▶ Speedup:  $\frac{18670}{8968} = 2,08$

- ▶ Diferentes ejecuciones, diferentes *outputs*.
- ▶ Descargar `incConcurrente.java`
- ▶ Ejecutar varias veces.
- ▶ ¿Qué está pasando?
- ▶ ¿Tiene solución?

- ▶ Dada una línea de tiempo de la CPU, analizando el flujo de instrucciones que hay en ella:
  - ▶ Tiene diferentes grupos de instrucciones de distintos procesos.
  - ▶ Diferentes ejecuciones, diferentes flujos.
  - ▶ Descargar `ejEntrelazado.java`
  - ▶ ¿Qué está pasando?



# El Porqué de Todo lo Anterior

```
N: Integer := 0;
```

```
Task Body P1 is  
begin
```

```
  N := N+1;
```

```
end P1;
```

```
Task Body P2 is  
begin
```

```
  N := N+1
```

```
End P2;
```

```
Proceso 1;  
Load (x);  
Add (x,1);  
Store (x);
```

```
Proceso 2;  
Load (x);  
Add (x,1);  
Store (x);
```

```
Instrucciones de  
Incremento como  
Inc.
```

## Ejemplo de Entrelazado: Efectos Laterales Indeseables

PROC.	INST.	N	REG1	REG2
Inicial		0		
P1	Load(x)	0	0	
P2	Load(x)	0	0	0
P1	Add(x, 1)	0	1	0
P2	Add(x, 1)	0	1	1
P1	Store(x)	1	1	1
P2	Store(x)	1	1	1

# Condiciones de Bernstein (Prevención de Efectos Laterales)

Permiten determinar si dos conjuntos de instrucciones se pueden ejecutar de forma concurrente.

## Condiciones de Bernstein

- ▶  $L(S_k) = \{a_1, a_2, \dots, a_n\}$  es el **conjunto de lectura** del conjunto de instrucciones  $S_k$ .
- ▶  $E(S_k) = \{b_1, b_2, \dots, b_n\}$  es el **conjunto de escritura** del conjunto de instrucciones  $S_k$ .
- ▶ Dos conjuntos de instrucciones  $S_i$  y  $S_j$  se pueden ejecutar concurrentemente si:
  - ▶  $L(S_i) \cap E(S_j) = \emptyset$
  - ▶  $E(S_i) \cap L(S_j) = \emptyset$
  - ▶  $E(S_i) \cap E(S_j) = \emptyset$

# Condiciones de Bernstein (Ejemplo de Aplicación)

## Ejemplo

Dadas las siguientes instrucciones, ¿admiten ejecución concurrente?

$$S1 : a = x + y$$

$$S2 : b = z - 1$$

$$S3 : c = a - b$$

$$S4 : w = c + 1$$

## Definición

Situación en que todos los procesos quedan a la espera de una condición que nunca se dará.

- ▶ Normalmente asociados a condiciones de espera circular por recursos comunes.
- ▶ Descargar `Deadlock.java`
- ▶ ¿Qué está ocurriendo?

- ▶ Trabajar a nivel de instrucciones atómicas o de grupos de instrucciones cuya ejecución atómica es forzada por el programador.
- ▶ Considerar la **Hipótesis del Progreso Finito**.
- ▶ Razonar sobre la corrección a nivel de grupos de instrucciones.
- ▶ Considerar la incertidumbre sobre el resultado final.
- ▶ Considerar la corrección bajo **todas** las secuencias de entrelazado posible.
- ▶ Desarrollar un modelo **abstracto** de sistema concurrente.

# Hipótesis de Progreso Finito

- ▶ No se puede hacer ninguna suposición acerca de las velocidades absolutas o relativas de ejecución de los procesos o hebras, salvo que es mayor que cero.
- ▶ Un programa concurrente se entiende en base a sus componentes (procesos o hebras) y sus interacciones, sin considerar el entorno de ejecución.

# Problemas de la Concurrency I: Condiciones de Concurso

- ▶ Aparecen cuando varias entidades concurrentes comparten recursos comunes accediendo a ellos **simultáneamente**.
- ▶ Pueden dar lugar a problemas graves como sobreescritura de datos, interbloqueos, etc.
- ▶ Son propias de los sistemas concurrentes.
- ▶ Se caracterizan, a nivel de programación, por zonas de código de las entidades concurrentes desde las que se accede a recursos comunes. Se las llama **secciones críticas**.
- ▶ Dado el indeterminismo de los programas concurrentes, la única forma de evitar una condición de concurso será forzar la ejecución **aislada** de cada sección crítica.



## Concepto

Consiste en evitar la condición de concurso sobre un recurso compartido **forzando la ejecución atómica** de las secciones críticas de las entidades concurrentes que lo usan. **Elimina el entrelazado.**

- ▶ Esto implica detener la ejecución de una entidad concurrente hasta que se produzcan determinadas circunstancias: **sincronización**.
- ▶ También implica el poder comunicar a otras entidades el estado de una dada: **comunicación**.
- ▶ Los lenguajes concurrentes deben permitir ambas.

# Beneficios del Uso de Exclusión Mutua

- ▶ Dentro de las secciones críticas no hay entrelazado.
- ▶ Se incrementa el determinismo, ya que se garantiza la ejecución secuencial (“atómica”) de las secciones críticas.
- ▶ Permite comunicar a los procesos a través de las secciones críticas.
- ▶ Acota a nivel de código (sintácticamente) las secciones críticas mediante el uso de **protocolos de entrada y salida** de las mismas.
- ▶ Pueden generar sobrecargas de ejecución.

- ▶ Descargar `incConcurrenteSeguro.java`.
- ▶ ¿Qué diferencias sintácticas tiene con `incConcurrente.java`?
- ▶ ¿Hay diferencias en la ejecución?
- ▶ ¿Qué cree que ha pasado?

# Modelo de Entidad Concurrente

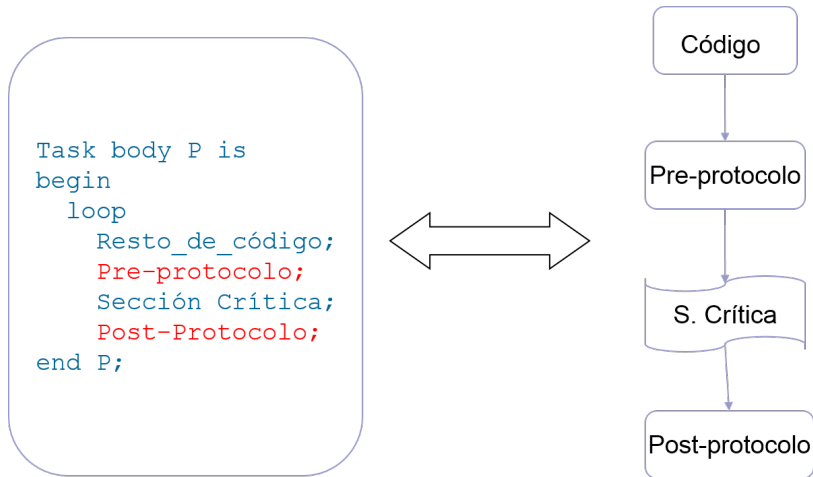


Figura: Modelo de entidad concurrente

## Ejemplo clásico: Problema del Productor-Consumidor

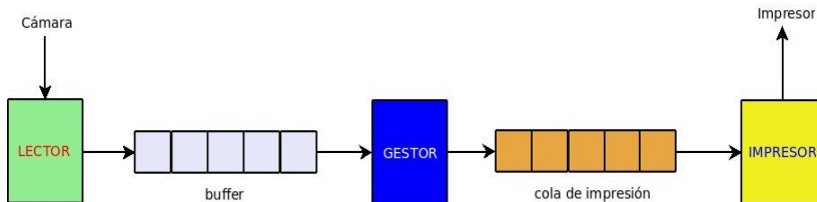


Figura: Problema del productor-consumidor

- ▶ Propiedades de Corrección
  - ▶ Propiedades de Seguridad
    - ▶ Exclusión mutua de las secciones críticas
    - ▶ No existen interbloqueos (deadlock)
  - ▶ Propiedades de Vivacidad
    - ▶ No existen interbloqueos activos (livelock)
    - ▶ No existe inanición de procesos
- ▶ Especificación y Verificación Formal de Sistemas Concurrentes
  - ▶ Lógica temporal.
  - ▶ CSP (Hoare).
  - ▶ Redes de Petri.
    - ▶ Metodología gráfica de representación.
    - ▶ Admite modelado dinámico de sistemas concurrentes.
    - ▶ Permite una verificación matemáticamente sencilla de las propiedades de corrección de un sistema concurrente.

## Definición

Se consideran lenguajes de programación concurrentes a aquellos que permiten expresar la concurrencia **directamente**, no siendo necesario el recurso al sistema operativo o a bibliotecas específicas.

- ▶ Incluyen herramientas para sincronizar y comunicar a entidades concurrentes.
- ▶ C no es un lenguaje concurrente.
- ▶ C++ incorpora (¡por fin!) concurrencia nativa desde la revisión C++11 (Agosto de 2012).
- ▶ Ada, Java, Occam son lenguajes concurrentes.
- ▶ Puede incluir OO como Java/C++, o no incluirla.

- ▶ Manuales
  - ▶ Accediendo directamente al hardware.
  - ▶ Usando bibliotecas software (pthread en C, herencia de la clase Thread o implementación de la interfaz Runnable en Java.
- ▶ Automáticas
  - ▶ Bajo la responsabilidad del Sistema Operativo (multiprogramación).
  - ▶ Detección del compilador de la concurrencia implícita en un programa secuencial y generación automática de los procesos concurrentes que correspondan.



# Modelos de Creación de Entidades Concurrentes

- ▶ Modelo de Creación **Estático**.
  - ▶ Número de procesos concurrentes fijado en tiempo de compilación.
  - ▶ No es un método flexible.
  - ▶ Es un método seguro, eficaz y limitado
- ▶ Modelo de Creación **Dinámico (estándar actual)**.
  - ▶ Procesos creados y destruidos en tiempo de ejecución.
  - ▶ Es un método flexible.
  - ▶ Es menos seguro que los métodos estáticos.
  - ▶ Es utilizado por el sistema operativo Unix o el lenguaje Java.
  - ▶ Es menos estructurado y más difícil de interpretar.

- ▶ Sistemas monoprocesador
  - ▶ Modelo de concurrencia simulado (procesadores virtuales).
  - ▶ Existe interfoliación de instrucciones.
  - ▶ Arquitectura de memoria común.
  - ▶ Existe planificación en el acceso al procesador.
- ▶ Sistemas multiprocesador
  - ▶ Acoplamiento fuerte (Multiprocesadores y **núcleos multicore**)
    - ▶ Arquitectura de memoria común (UMA).
    - ▶ Soluciones propuestas para monoprocesadores admisibles.
    - ▶ Existe planificación en el acceso a los núcleos.
  - ▶ Acoplamiento débil (Sistemas distribuidos).
    - ▶ Arquitectura de comunicaciones.
    - ▶ Arquitectura de memoria no común (NUMA).
    - ▶ Necesidad de soluciones *ad-hoc*.

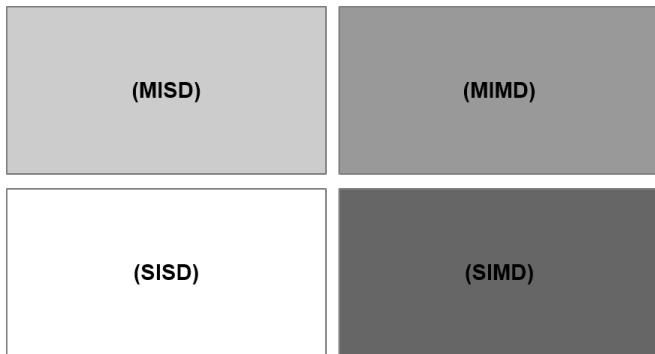


Figura: Taxonomía de Flynn

# Taxonomía de Flynn: SISD

- ▶ SI: una instrucción por cada ciclo de reloj.
- ▶ SD: un segmento de datos por cada ciclo de reloj.
- ▶ Ejecución determinista.
- ▶ Antiguo modelo de computación

UNIVAC 1



IBM 360



CRAY 1



Dell Laptop

# Taxonomía de Flynn: SIMD

- ▶ SI: todas las unidades de ejecución ejecutan la misma.
- ▶ SD: cada unidad la aplica sobre datos distintos.
- ▶ Ejecución determinista y síncrona (con cerrojo).
- ▶ Muy útil en nubes de datos reticuladas.

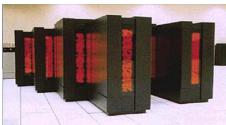
ILLIAC IV



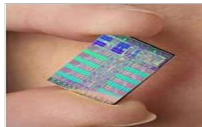
Cray Y-MP



Thinking Machine



Cell Processor  
(GPU)



# Taxonomía de Flynn: MIMD

- ▶ MI: cada procesador ejecuta distintas instrucciones
- ▶ MD: sobre datos diferentes
- ▶ Ejecución determinista/no determinista y síncrona/asíncrona.
- ▶ Multiprocesadores, clusters, multicores...

Cluster  
IBM  
Power 5



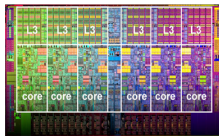
Cluster  
Cray XT3



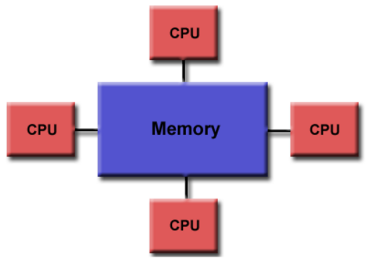
Cluster  
AMD  
Opteron



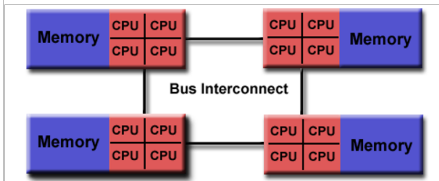
Procesador  
Multicore  
Xeon 5600



# Consideraciones Sobre el Hardware (Memoria)



MODELO UMA  
(Multiprocesadores y Mutinúcleos)



MODELO NUMA  
(Sist. Distribuidos y *Clusters*)

# Implementación de Programas Distribuidos

- ▶ Modelo de paso de mensajes.
  - ▶ Primitivas de comunicación send/receive.
  - ▶ Carácter de las primitivas.
  - ▶ Protocolo de solicitud/respuesta de pocas capas.
- ▶ Modelo RPC/RMI
  - ▶ Permite al programador invocar procedimientos remotos de forma (idealmente) transparente.
  - ▶ Requiere procedimientos de resguardo (stubs).
- ▶ Modelo de Objetos Distribuidos
  - ▶ DCOM (Microsoft).
  - ▶ CORBA (OMG).
  - ▶ EJB (Sun).



## Definicion

Son sistemas concurrentes caracterizados por la existencia de una **ligadura temporal** para completar una tarea.

- ▶ Control robótico, teledirección, telemedicina, multimedia...
- ▶ El programador tiene acceso al reloj y planificador.
- ▶ Pueden ser:
  - ▶ Críticos: las tareas deben, necesariamente, satisfacer las restricciones impuestas por la ligadura temporal.
  - ▶ No criticos: las tareas intentan satisfacer tales restricciones, pero el diseño no garantiza la consecución de las mismas con total garantía.
- ▶ Linux RT y JRTS.

# Sistemas de Tiempo Real: Características

- ▶ Predictibilidad.
- ▶ Determinismo.
- ▶ Certidumbre.
- ▶ Control.

# En el Próximo Tema...

- ▶ Programación en Lenguaje Java.
- ▶ Elementos básicos.
- ▶ Clases y objetos.
- ▶ E/S básica.
- ▶ Modelo de herencia.



Ben-Ari, M.

*Concurrent and Distributed Programming*

Prentice Hall, 2006

[Capítulos 1 y 2]



Bollela, G. y Bruno, E.

*Real Time Java Programming with Java RTS*

Sun Microsystems, 2009.

[Capítulo 1, págs. 3-13]



Palma, J.

*Programación concurrente*

2ª edición, [Capítulos 1 y 2]