

2º curso / 2º cuatr.

Grado en  
Ing. Informática

# Arquitectura de Computadores

## Tema 1

# Arquitecturas Paralelas: Clasificación y Prestaciones

Material elaborado por los profesores responsables de la asignatura:

Mancia Anguita – Julio Ortega

Licencia Creative Commons



ugr

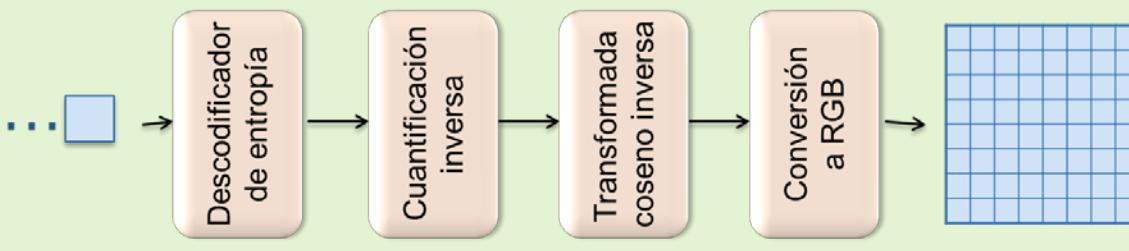
Universidad  
de Granada



# Lecciones

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones

Estructura de funciones de un decodificador JPEG



Suma de vectores  
 $\mathbf{c} = \mathbf{a} + \mathbf{b}$

Producto de matrices  
 $\mathbf{C} = \mathbf{A} * \mathbf{B}$

```
Func1() { ... }
Func2() { ...
    for (i=0;i<N;i++) {
        código para i
    }
}
Func3() { ... }
Main () {
...
    Func1(...);
    Func2(...);
    Func3(...);
...
}
```

# Objetivos Lección 1

- Conocer las clasificaciones usuales del paralelismo implícito en una aplicación. Distinguir entre paralelismo de tareas y paralelismo de datos.
- Distinguir entre dependencias RAW, WAW, WAR.
- Distinguir entre *thread* y proceso.
- Relacionar el paralelismo implícito en una aplicación con el nivel en el que se hace explícito para que se pueda utilizar (instrucción, thread, proceso) y con las arquitecturas paralelas que lo aprovechan.

# Bibliografía

## ➤ Fundamental

- Sección 7.1. J. Ortega, M. Anguita, A. Prieto. *Arquitectura de Computadores*, Thomson, 2005. ESIIT/C.1 ORT arq

## ➤ Complementaria

- Secciones 3.7.1, 3.7.2. T. Rauber, G. Ründer. *Parallel Programming: for Multicore and Cluster Systems*. Springer 2010. Disponible en línea (biblioteca UGR):  
<http://dx.doi.org/10.1007/978-3-642-04818-0>

# Criterios de clasificaciones del paralelismo implícito en una aplicación

Niveles de  
paralelismo implícito  
en un código que  
resuelve la aplicación

Clasificación  
del  
paralelismo  
implícito

Paralelismo de  
tareas –  
paralelismo de  
datos

Granularidad

# Niveles de paralelismo implícito en una aplicación

Niveles:

Programas

Programa1

Programa2

...



Funciones

Func1() {  
...  
}

Func2() {  
...  
}

Func3() {  
...  
}

Bucle (bloques)

for ( ){  
...  
}

while( ){  
...  
}

Operaciones

\*

+

/

Granularidad:

Grano Grueso

Grano  
Medio

Grano  
Medio-Fino

Grano Fino

# Dependencias de datos

- Condiciones que se deben cumplir para que el bloque de código  $B_2$  presente dependencia de datos con respecto a  $B_1$ :
  - Deben hacer referencia a una misma posición de memoria M (variable).
  - $B_1$  aparece en la secuencia de código antes que  $B_2$
- Tipos de dependencias de datos (de  $B_2$  respecto a  $B_1$ ):
  - RAW (*Read After Write*) o dependencia verdadera
  - WAW (*Write After Write*) o dependencia de salida
  - WAR (*Write After Read*) o antidependencia

$B_1$   
 $B_2$



...  
 $a=b+c$   
... //código que no usa a  
 $d=a+c$   
...

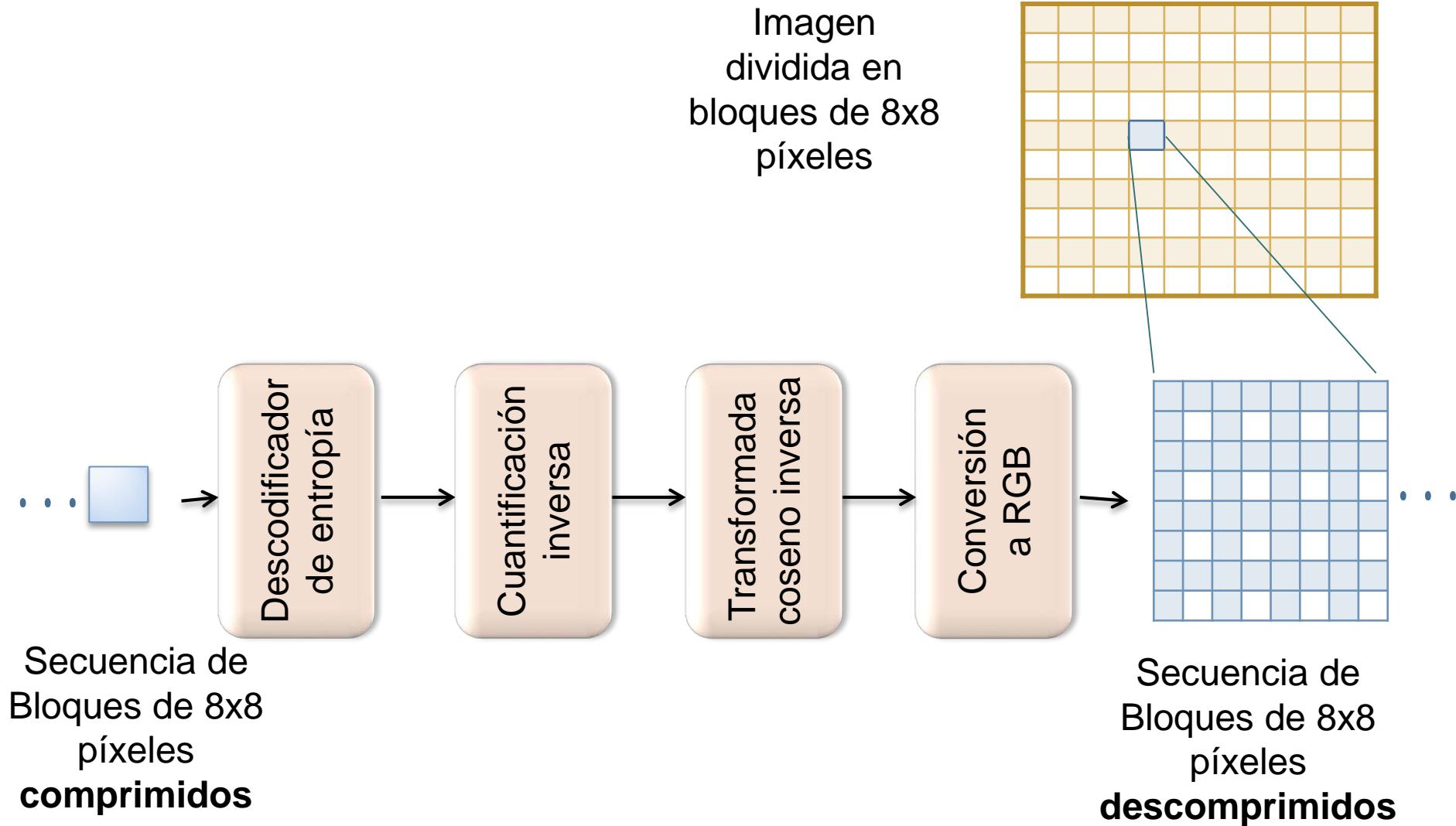
...  
 $a=b+c$   
... //se lee a  
 $a=d+e$   
... //se lee a

...  
 $b=a+1$   
...  
 $a=d+e$   
... //se lee a

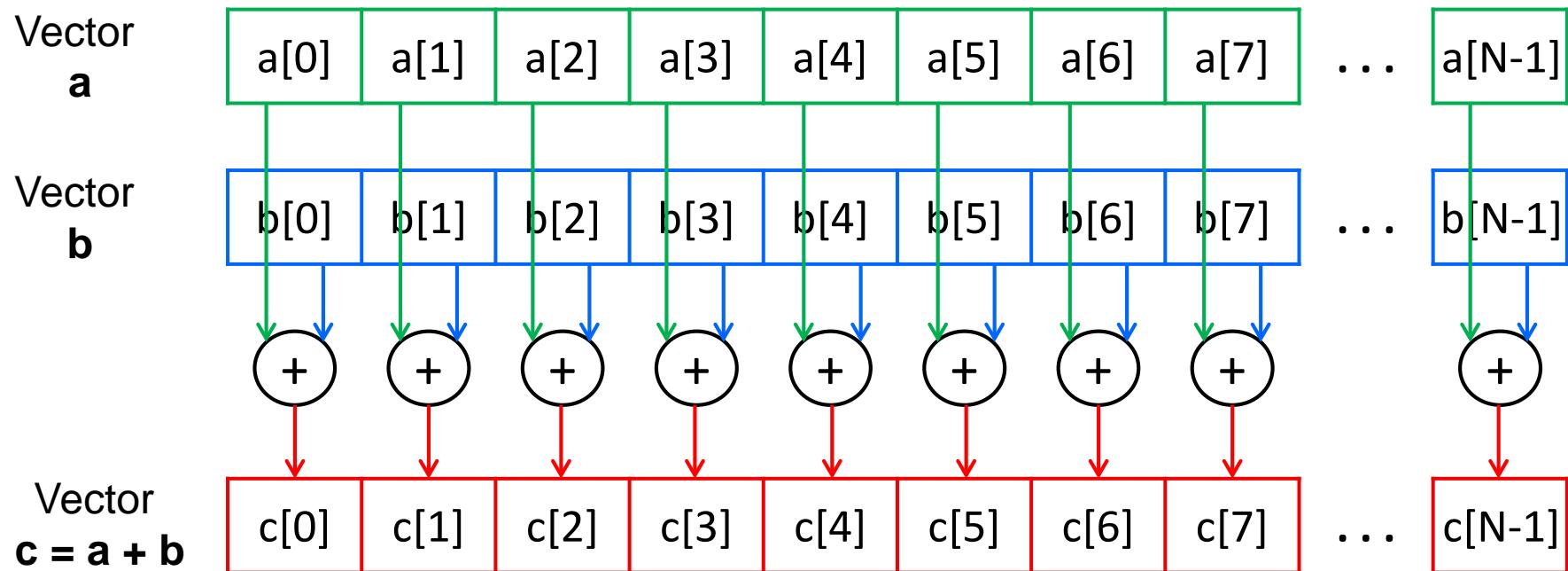
# Paralelismo implícito en una aplicación

- Paralelismo de tareas (*task parallelism o TLP -Task Level Par.*)
  - Se encuentra extrayendo la estructura lógica de funciones de una aplicación.
  - Está relacionado con el paralelismo a **nivel de función**
- Paralelismo de datos (*data parallelism o DLP-Data Level Par.*)
  - Se encuentra implícito en las operaciones con estructuras de datos (vectores y matrices)
    - Por ejemplo, la operación vectorial  $V1=V2+V3$  engloba múltiples sumas de escalares.
  - Se puede extraer de la representación matemática de la aplicación.
  - Está relacionado principalmente con el paralelismo a **nivel de bucle**

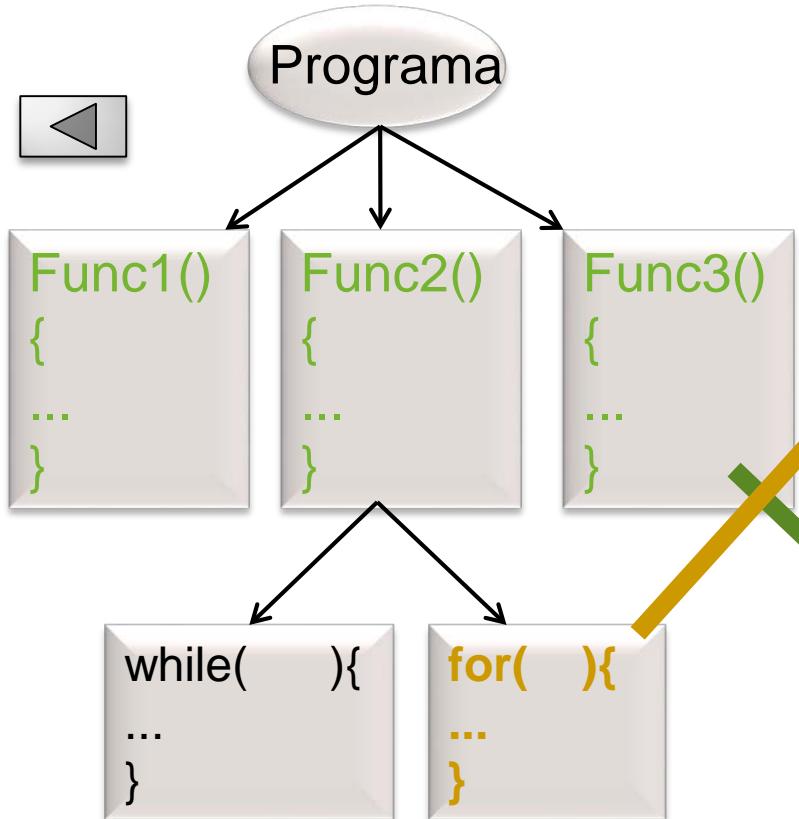
# Estructura de funciones lógica de una aplicación. Ej.: decodificador JPEG



# Paralelismo de datos. Ej: suma de dos vectores

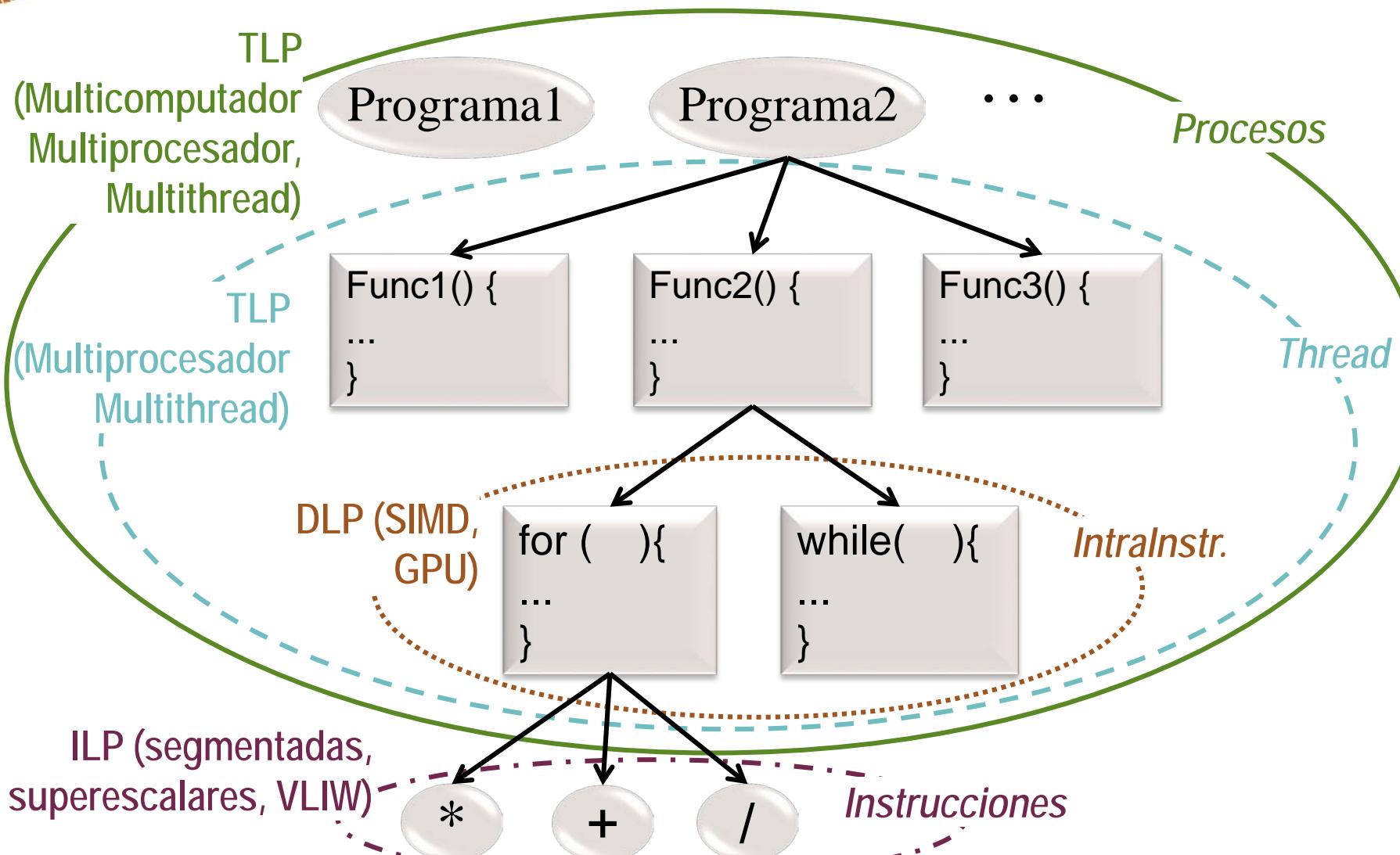


# Paralelismo de datos y paralelismo de tareas en OpenMP (Prácticas 1, 2 y 3)



```
Func1()\n{...}\nFunc2()\n{...}\n#pragma omp parallel for\nfor (i=0;i<N;i++){\n    código para i\n}\n...\nFunc3()\n{...}\nMain () {\n...\n#\n#pragma omp parallel sections\n{\n    #pragma omp section\n    Func1();\n    #pragma omp section\n    Func2();\n    #pragma omp section\n    Func3();\n}\n...\n}
```

# Paralelismo implícito (nivel de detección), explícito y arquitecturas paralelas



# Nivel de paralelismo explícito.

## Unidades en ejecución en un computador

### ➤ Instrucciones

- La unidad de control de un core o procesador gestiona la ejecución de instrucciones por la unidad de procesamiento

### ➤ *Thread o light process* (concepto del SO)

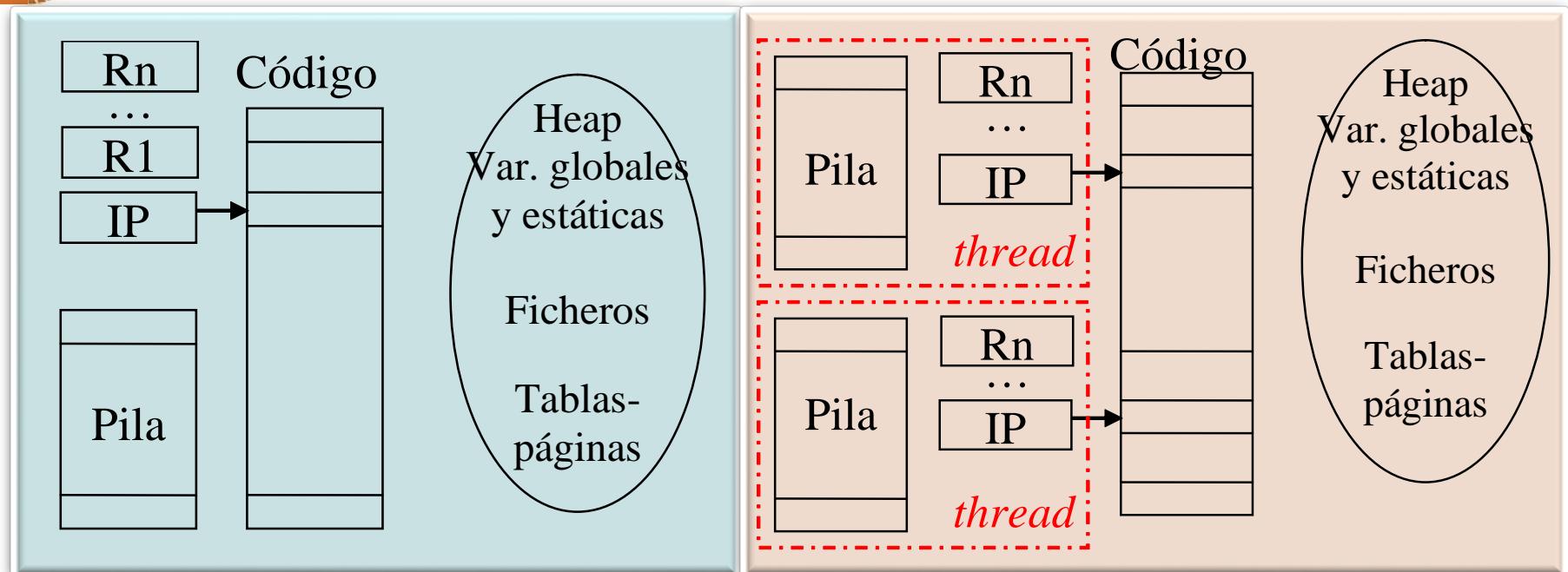
- Es la menor unidad de ejecución que gestiona el SO
- Menor secuencia de instrucciones que se pueden ejecutar en paralelo o concurrentemente

### ➤ Proceso o *process* (concepto del SO)

- Mayor unidad de ejecución que gestiona el SO
- Un proceso consta de uno o varios thread

# Nivel de parallelismo explícito.

## Threads versus procesos I



- **Proceso:** comprende el código del programa y todo lo que hace falta para su ejecución:
  - Datos en pila, segmentos (variables globales y estáticas) y en heap
  - Contenido de los registros
  - Tabla de páginas
  - Tabla de ficheros abiertos
- Para comunicar procesos hay que usar llamadas al SO
- Un proceso puede constar de múltiples flujos de instrucciones, llamados **threads** o procesos ligeros. Cada thread tiene:
  - Su propia pila
  - Contenido de los registros, en particular el contador de programa o *instruction pointer* y el puntero de pila o *stack pointer*
- Para comunicar threads de un proceso se usa la memoria que comparten

grano

# Nivel de parallelismo explícito.

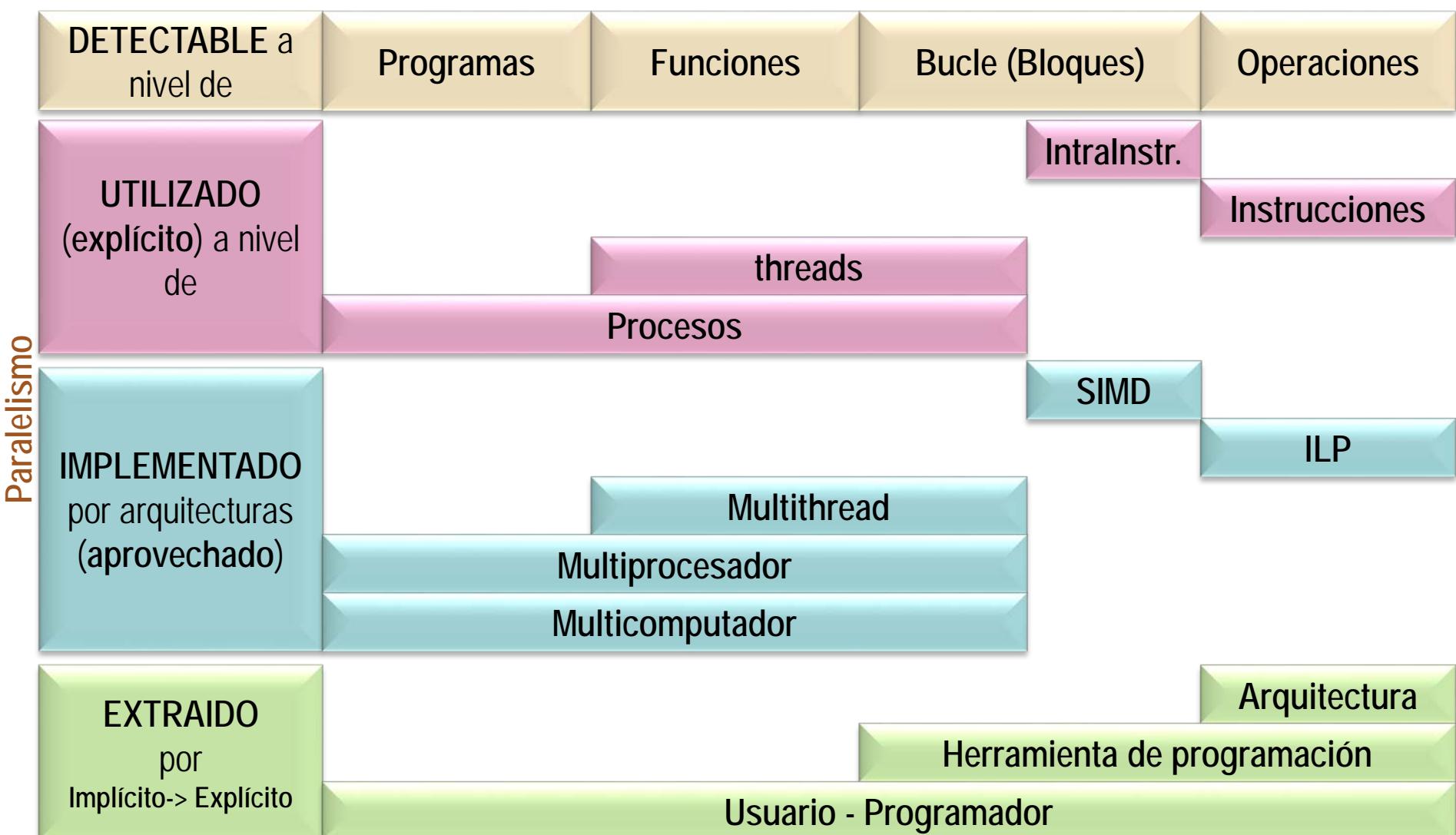
## Threads versus procesos II



# Detección, utilización, implementación y extracción del paralelismo

niveles

AC ATC



# Para ampliar .....

## ➤ Páginas Web:

- [http://en.wikipedia.org/wiki/Instruction-level\\_parallelism](http://en.wikipedia.org/wiki/Instruction-level_parallelism)
- [http://en.wikipedia.org/wiki/Task\\_parallelism](http://en.wikipedia.org/wiki/Task_parallelism)
- [http://en.wikipedia.org/wiki/Data\\_parallelism](http://en.wikipedia.org/wiki/Data_parallelism)

2º curso / 2º cuatr.

Grado en  
Ing. Informática

# Arquitectura de Computadores

## Tema 1

# Arquitecturas Paralelas: Clasificación y Prestaciones

Material elaborado por los profesores responsables de la asignatura:

Mancia Anguita – Julio Ortega

Licencia Creative Commons



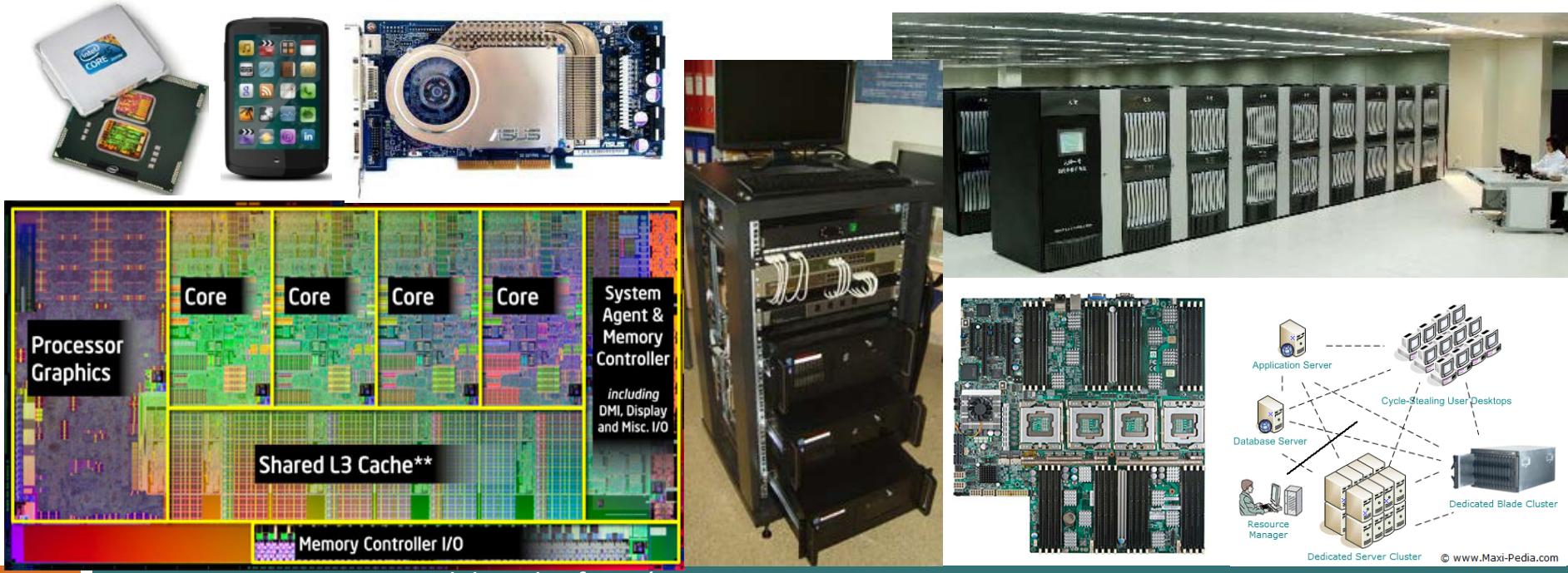
ugr

Universidad  
de Granada



# Lecciones

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones



# Objetivos Lección 2

- Distinguir entre procesamiento o computación paralela y distribuida.
- Clasificar los computadores según segmento del mercado.
- Distinguir entre las diferentes clases de arquitecturas de la clasificación de Flynn.
- Diferenciar un multiprocesador de un multicomputador.
- Distinguir entre NUMA y SMP.
- Distinguir entre arquitecturas DLP, ILP y TLP.
- Distinguir entre arquitecturas TLP con una instancia de SO y TLP con varias instancias de SO.

# Bibliografía

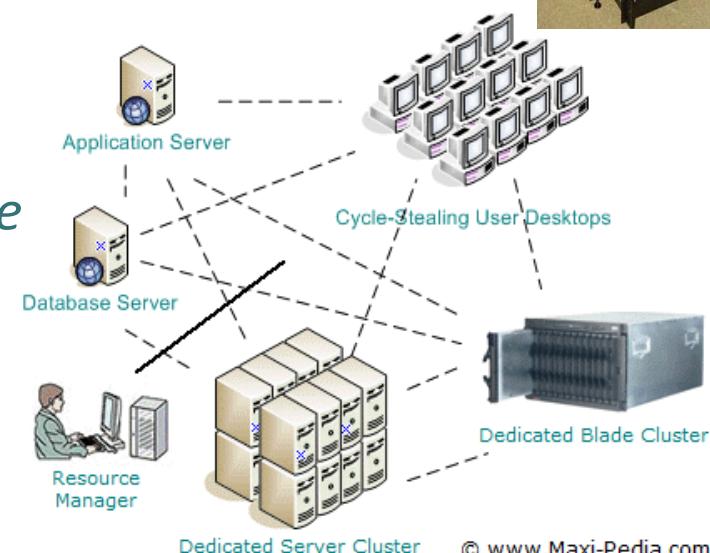
- Fundamental
  - Secciones 1.3, 7.3 y 7.2.2. J. Ortega, M. Anguita, A. Prieto. *Arquitectura de Computadores*, Thomson, 2005. ESIIT/C.1 ORT arq
- Complementaria
  - T. Rauber, G. Ründer. *Parallel Programming: for Multicore and Cluster Systems*. Springer 2010. Disponible en línea (biblioteca UGR): <http://dx.doi.org/10.1007/978-3-642-04818-0>

# Contenidos

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
  - **Computación paralela y computación distribuida**
  - Clasificaciones de arquitecturas y sistemas paralelos
  - Nota histórica
- Lección 3. Evaluación de prestaciones

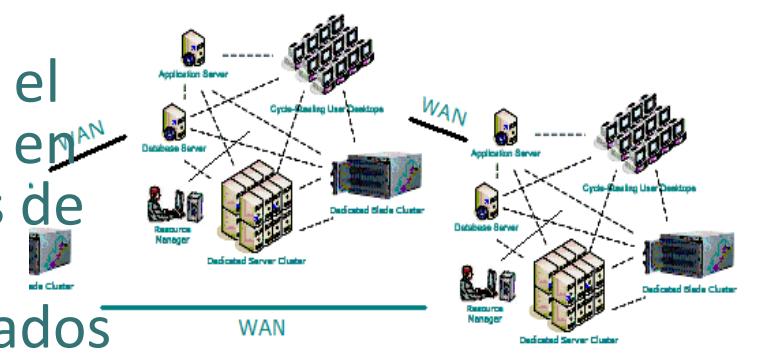
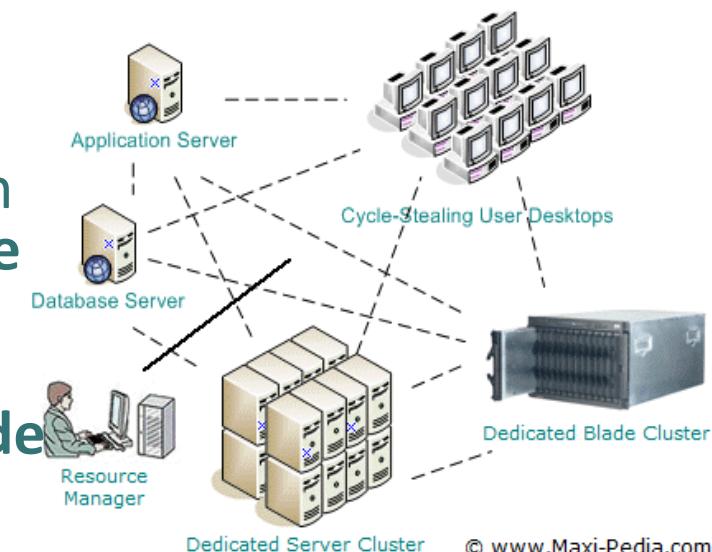
# Computación paralela – Computación distribuida

- Computación paralela (se estudia en AC)
  - Estudia los aspectos hardware y software relacionados con el *desarrollo y ejecución de aplicaciones* en un sistema de cómputo compuesto por **múltiples cores/procesadores/computadores** que es visto externamente como una **unidad autónoma** (multicores, multiprocesadores, multicamputadores, cluster)
- Computación distribuida
  - Estudia los aspectos hardware y software relacionados con el *desarrollo y ejecución de aplicaciones* en un **sistema distribuido**; es decir, en una **colección de recursos autónomos** (PC, servidores -de datos, software, ...-, supercomputadores ...) situados en **distintas localizaciones físicas**



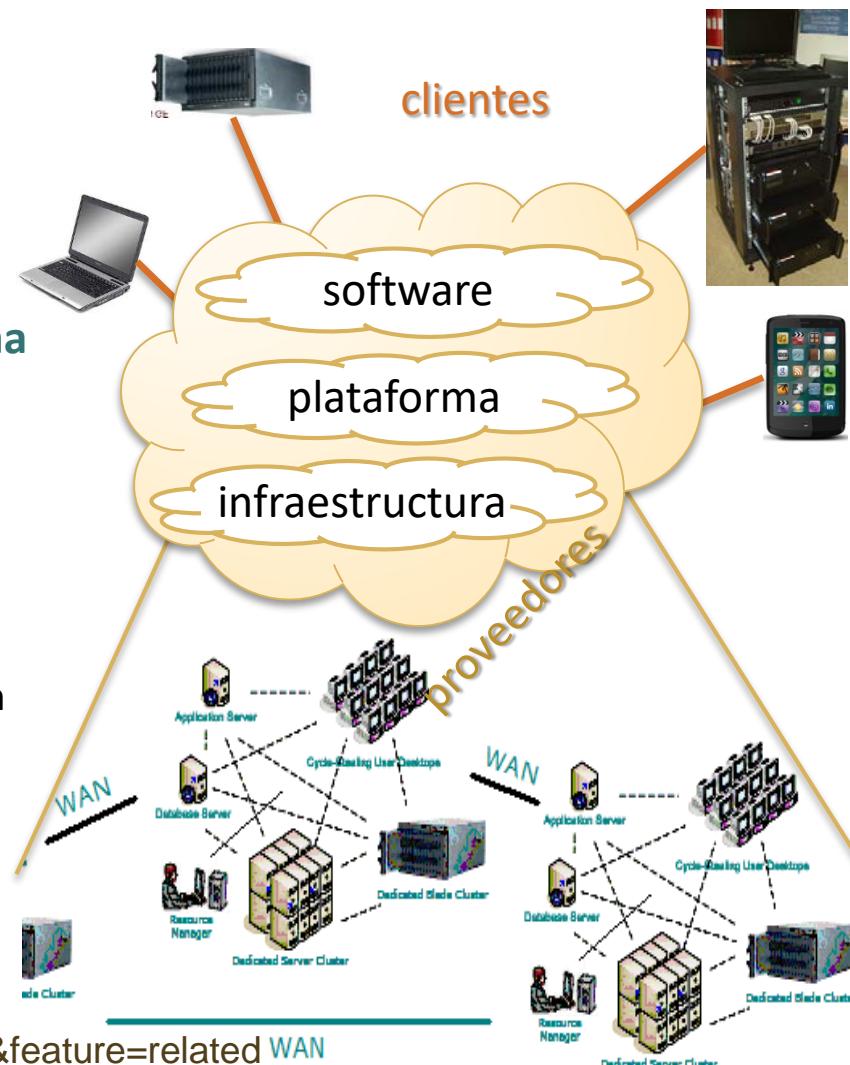
# Computación distribuida a gran escala: Computación grid

- Computación distribuida baja escala
  - Estudia los aspectos relacionados con el *desarrollo y ejecución de aplicaciones* en una colección de recursos autónomos **de un dominio administrativo** situados en **distintas localizaciones físicas** conectados a través de **infraestructura de red local**
- Computación grid
  - Estudia los aspectos relacionados con el *desarrollo y ejecución de aplicaciones* en una colección de recursos autónomos **de múltiples dominios administrativos geográficamente distribuidos** conectados con **infraestructura de telecomunicaciones**



# Computación distribuida a gran escala: Computación nube o *cloud*

- Computación *cloud*
  - Comprende los aspectos relacionados con el desarrollo y ejecución de aplicaciones en un **sistema cloud**
- Sistema *cloud*
  - Ofrece **servicios de infraestructura, plataforma y/o software**, por los que se paga cuando se necesitan (*pay-per-use*) y a los que se accede típicamente a través de una **interfaz (web) de auto-servicio**
  - Consta de **recursos virtuales** que
    - son una **abstracción** de los recursos físicos
    - parecen **ilimitados en número y capacidad** y son reclutados/liberados de forma **inmediata** sin interacción con el proveedor
    - soportan el acceso de múltiples clientes (**multi-tenant**)
    - están conectados con métodos **estándar independientes de la plataforma de acceso.**



Para ampliar:<http://www.youtube.com/watch?v=SgujalzkwrE&feature=related> [WAN](http://soa-eda.blogspot.com/2010/05/cloud-computing-explained.html)  
<http://soa-eda.blogspot.com/2010/05/cloud-computing-explained.html>

# Contenidos

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
  - Computación paralela y computación distribuida
  - **Clasificaciones de arquitecturas y sistemas paralelos**
  - Nota histórica
- Lección 3. Evaluación de prestaciones

# Criterios de clasificación de computadores

## → Comercial

- Segmento del mercado
  - embebidos, servidores gama baja ...
- Educación, investigación (también usados por fabricantes y vendedores)
- flujos de instrucciones y flujos de datos: clasificación de Flynn (1972)
- Sistema de memoria
- flujos de instrucciones (propuesta de clasificación de arquitecturas con múltiples flujos de instrucciones)
- Nivel del paralelismo aprovechado (propuesta de clasif.)

# Segmento del mercado



## MATERIAS DEL GRADO

Supercomputadores

Servidores de gama alta  
 $(500.000\$ < \text{high-end})$

Servidores de gama media  
 $(25.000\$ < \text{mid-range} < 500.000\$)$

Servidores de gama baja  
 $(\text{volume}/\text{entry-level} < 25.000\$)$

PC/WS

Mercado de computadores  
empotrados

Sistemas de Cómputo  
de Altas Prestaciones  
(IC.SCAP)

Estructura y  
Arquitectura de  
Computadores  
(R.EAC)

Sistemas de Cómputo  
para Aplicaciones  
Específicas (IC.SCAE)

# Clasificación de Computadores según segmento

Externo (*desktop, laptop, server, cluster ...*)  
- R.EAC, IC.SCAP

Para todo tipo de aplicaciones:

- Oficina, entretenimiento, ...
- Procesamiento de transacciones o OLTP, sistemas de soporte de decisiones o DSS, e-commerce, ...
- Científicas ( medicina, biología, predicción del tiempo, etc.) y animación (películas animadas, efectos especiales, etc.), ...

Empotrado (oculto) - IC.SCAE

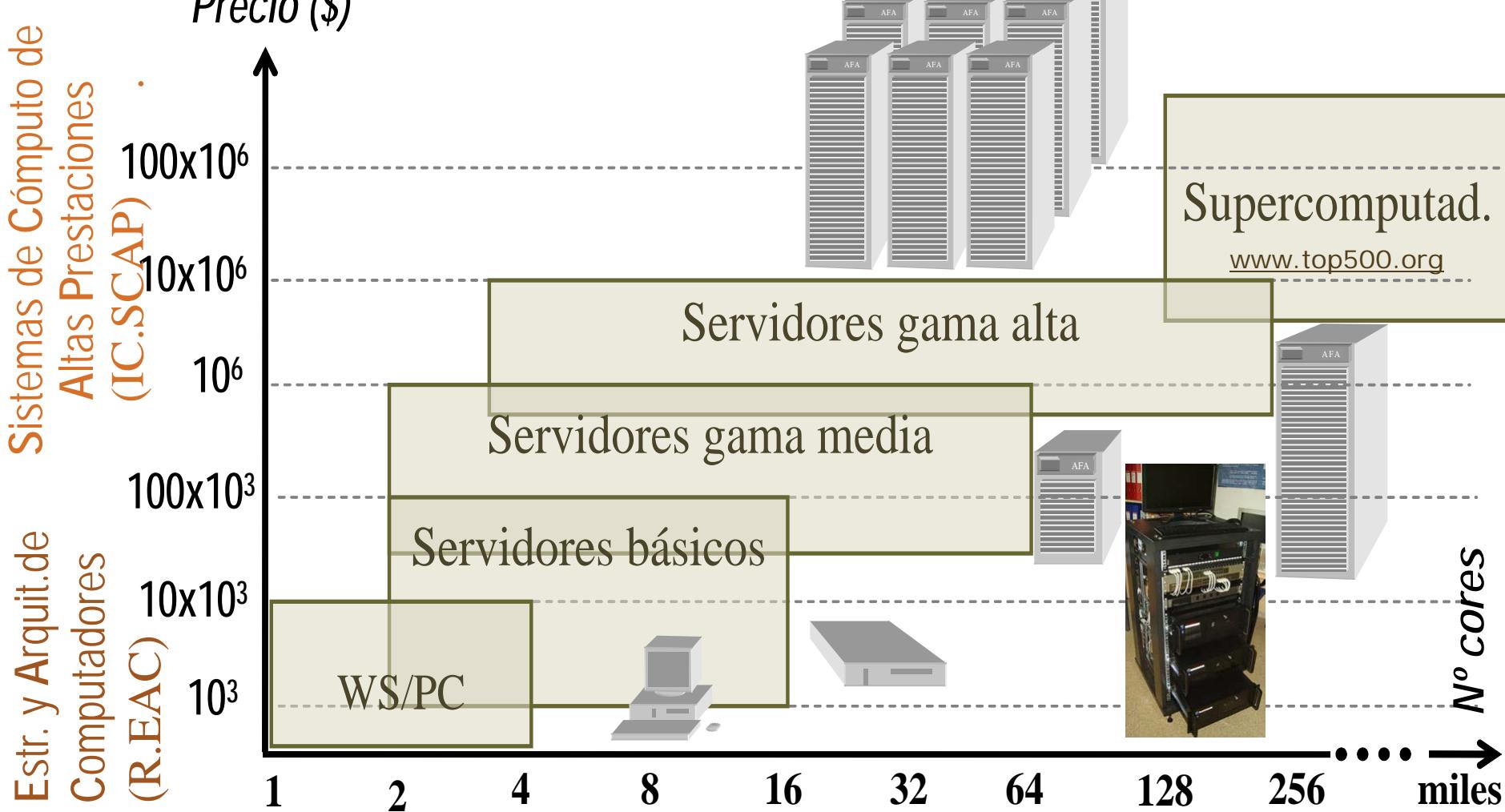
Aplicaciones de propósito específico

- Videojuegos, teléfonos, coches, electrodomésticos, ...

Restricciones típicas:

- Consumo de potencia, precio, tamaño reducidos
- Tiempo real

# Clasificación de Comp. (externos) según segmento del mercado



# Criterios de clasificación de computadores

## ➤ Comercial

### ➤ Segmento del mercado

- embebidos, servidores gama baja ...

→ Educación, investigación (también usados por fabricantes y vendedores)

- Flujos de instrucciones y flujos de datos (clasificación de Flynn 1972)
- Sistema de memoria
- flujos de instrucciones (propuesta de clasificación de arquitecturas con múltiples flujos de instrucciones)
- Nivel del paralelismo aprovechado (propuesta de clasif.)

# Clasificación de Flynn de arquitecturas, 1972 (Flujo instr./flujo de datos)

Clasifica-  
ción de  
Flynn

flujo de Instrucciones

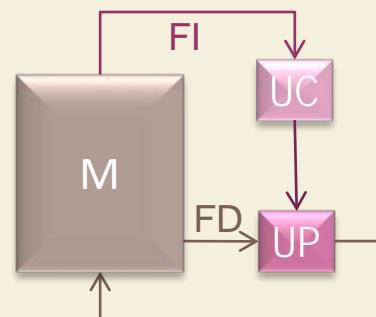
Único (*Single*)

Múltiple

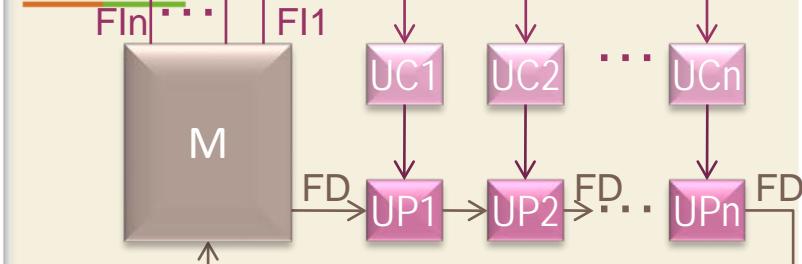
Único(*Sing*)

**SISD**

(Un núcleo  
procesador)



**MISD**

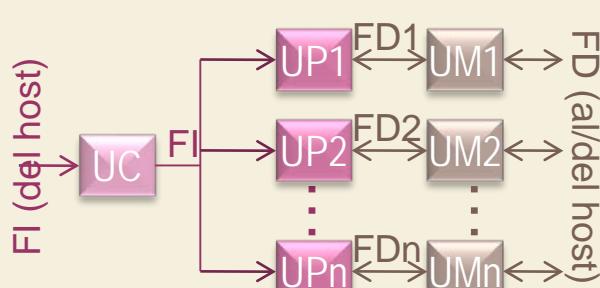


flujo de Datos

Múltiple

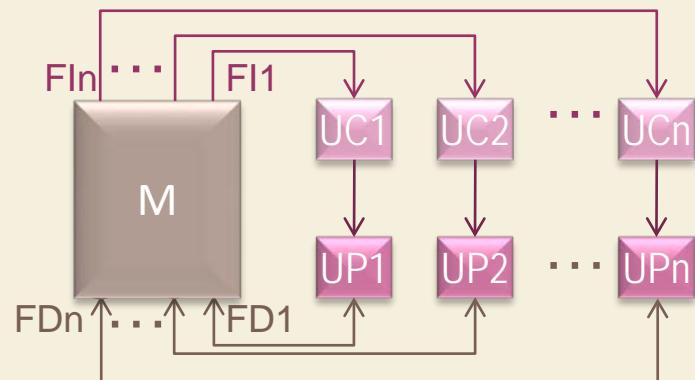
**SIMD**

(GPU, Procesadores matriciales)



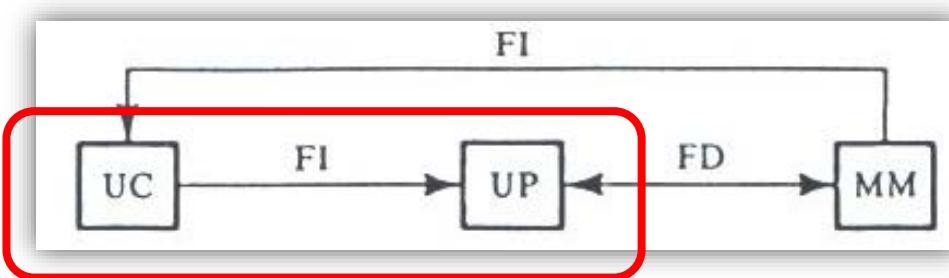
vector

**MIMD** (multicores, multipro-  
cesadores, multicamputadores)



# Arquitecturas SISD

## Descripción Estructural

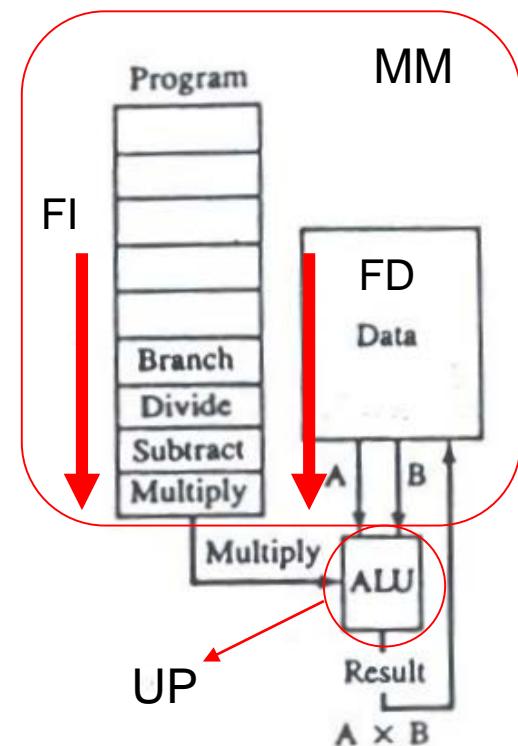


CPU, núcleo, procesador

Corresponde a los computadores uni-procesador  
(un núcleo o core procesador)

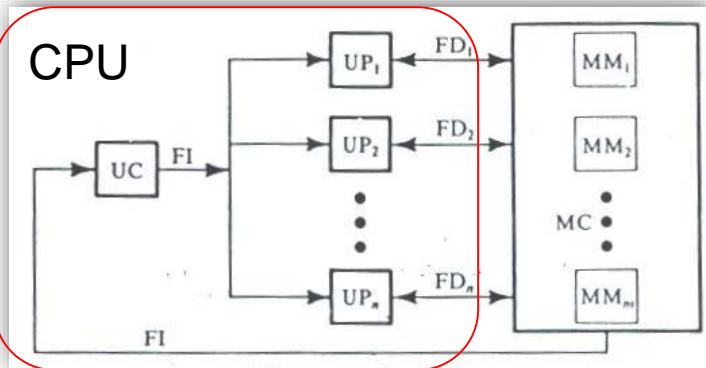
```
for i:=1 to 4 do
begin
  C[i]:=A[i]+B[i];
  F[i]:=D[i]-E[i];
  G[i]:=K[i]*H[i];
end;
```

## Descripción Funcional

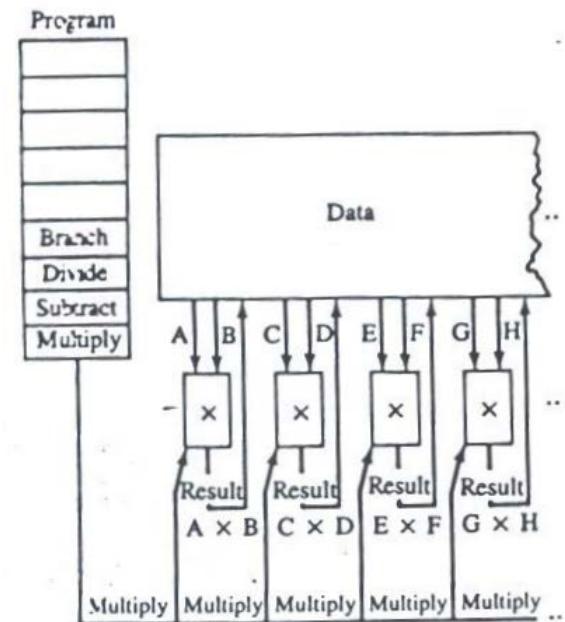


# Arquitecturas SIMD

## Descripción Estructural



## Descripción Funcional



Aprovechan paralelismo de datos

```
for all EPi(i:=1 to 4) do
begin
    C[i]:=A[i]+B[i];
    F[i]:=D[i]-E[i];
    G[i]:=K[i]*H[i];
end;
```

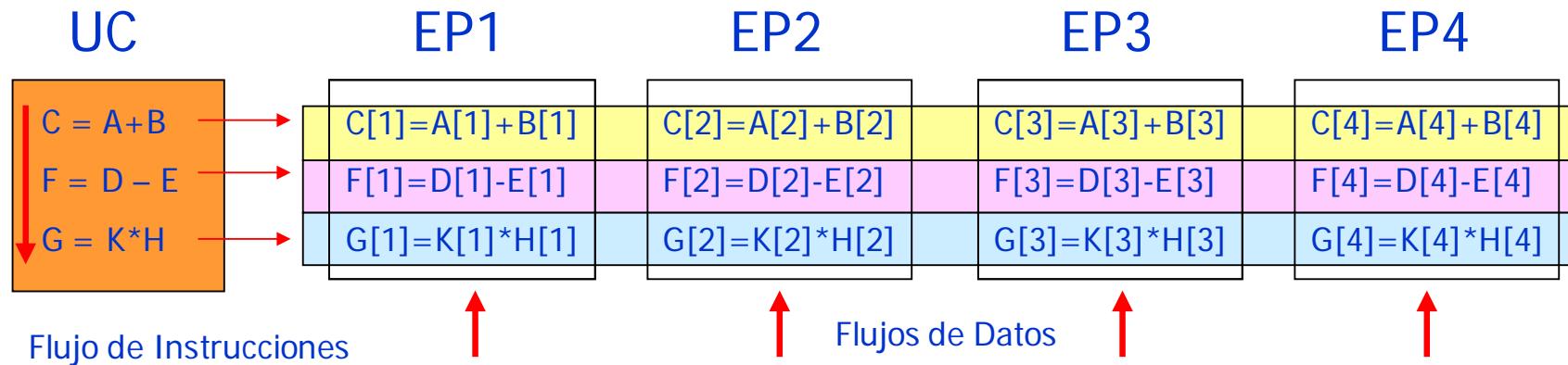
Procesadores  
Matriciales

Procesadores  
Vectoriales

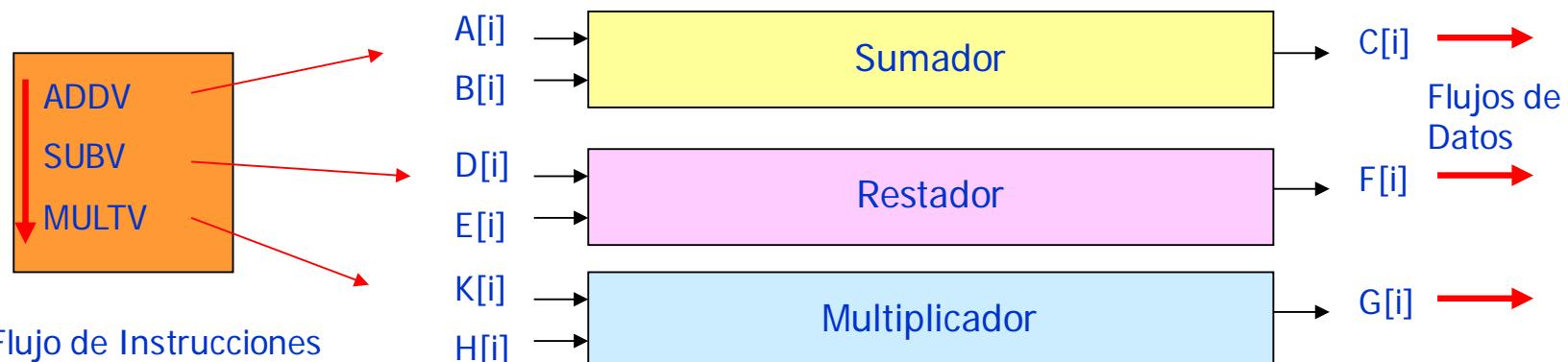
**ADDV C,A,B  
SUBV F,D,E  
MULV G,K,H**

# Arquitecturas SIMD: Ejemplo

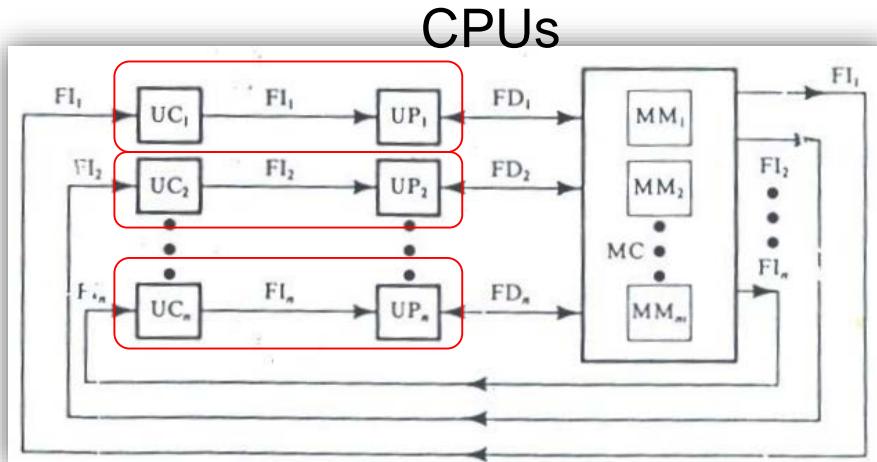
## Procesador Matricial



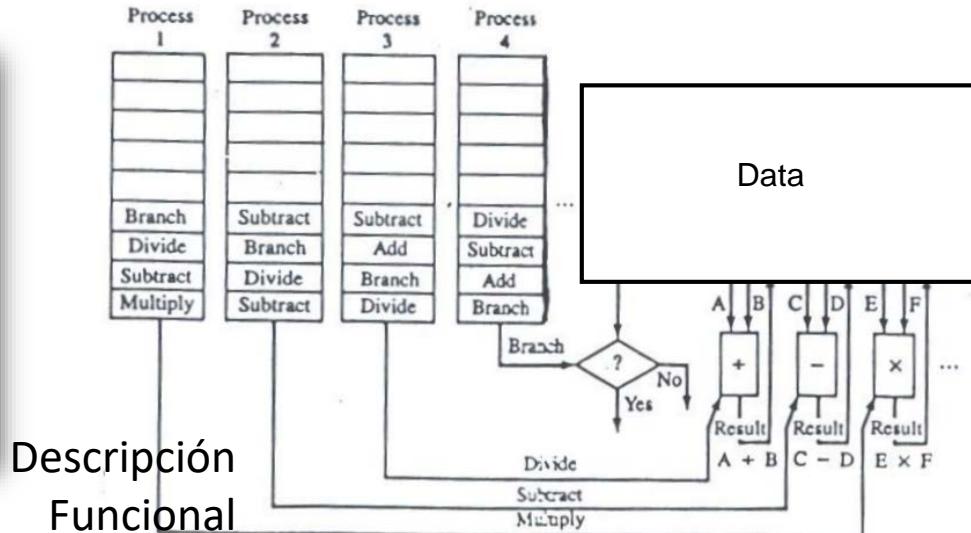
## Procesador Vectorial



# Arquitecturas MIMD



Descripción Estructural



Descripción Funcional

Corresponde con Multinúcleos, Multiprocesadores y Multicomputadores: Puede aprovechar, además, **paralelismo funcional**

```
for i:=1 to 4 do
begin
  C[i]:=A[i]+B[i];
end;
```

Proc 1

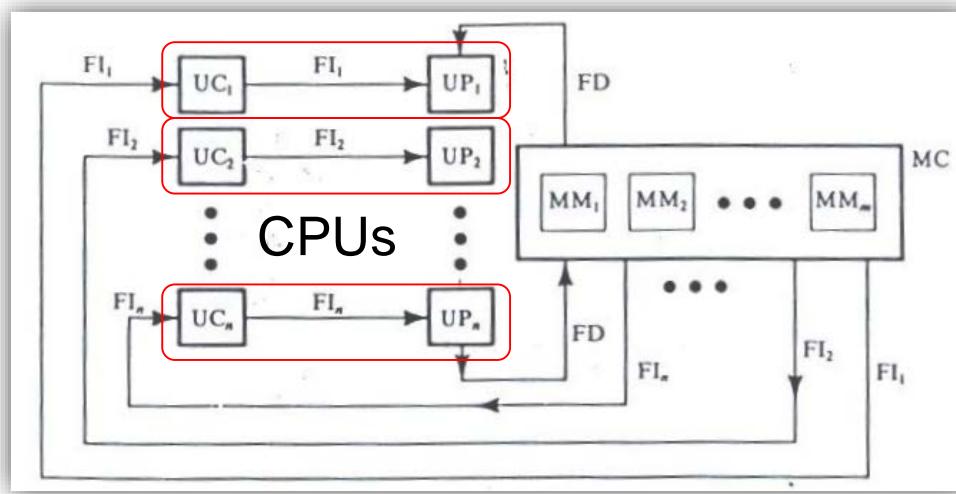
```
for i:=1 to 4 do
begin
  F[i]:=D[i]-E[i];
end;
```

Proc 2

```
for i:=1 to 4 do
begin
  G[i]:=K[i]*H[i];
end;
```

Proc 3

# Arquitecturas MISD



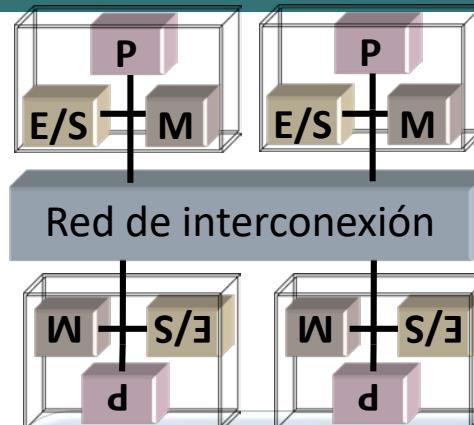
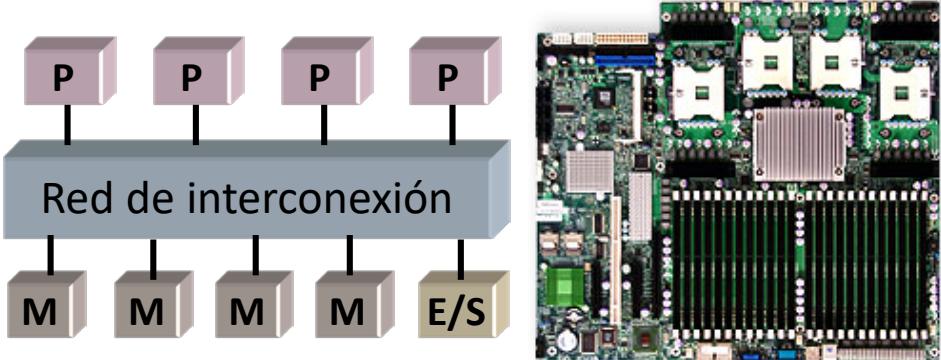
- No existen computadores que funcionen según este modelo
- Se puede simular en un código este modelo para aplicaciones que procesan una secuencia o flujo de datos

# Criterios de clasificación de computadores

- Comercial
    - Segmento del mercado
      - embebidos, servidores gama baja ...
  - Educación, investigación: también usados por fabricantes y vendedores
    - Flujos de instrucciones y flujos de datos (clasificación de Flynn 1972)
- Sistema de memoria
- Flujos de instrucciones (propuesta de clasificación de arquitecturas con múltiples flujos de instrucciones)
  - Nivel del paralelismo aprovechado (propuesta de clasif.)

# Clasificación de Computadores Paralelos

## MIMD según el sistema de memoria



### Multiprocesadores

Todos los procesadores  
comparten el **mismo espacio**  
de direcciones

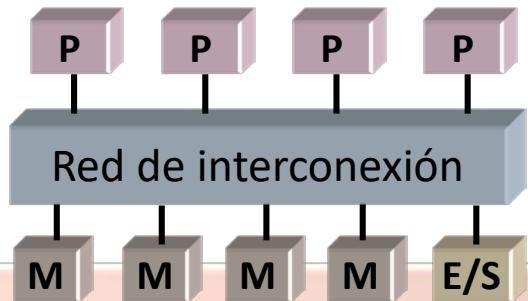
El programador NO necesita  
conocer dónde están  
almacenados los datos

### Multicomputadores

Cada procesador tiene su  
**espacio de direcciones**  
propio

El programador necesita  
conocer dónde están  
almacenados los datos

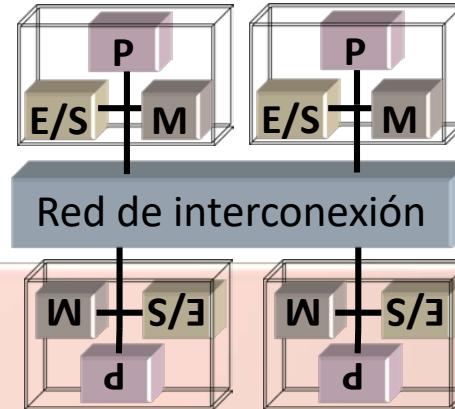
# Comparativa SMP (Symmetric Multi-Processor) y multicomputadores I



Multiprocesador con memoria centralizada (SMP)

Mayor latencia - Poco escalable

Comunicación implícita mediante variables compartidas. Datos no duplicados en memoria principal

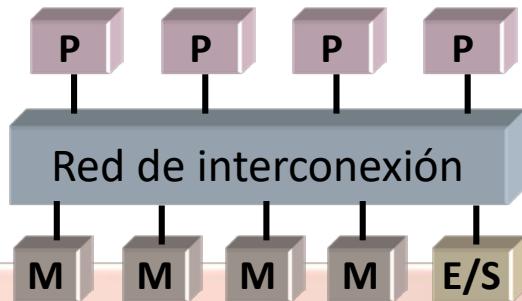


Multicomputador

Menor latencia – Más escalable

Comunicación explícita mediante software para paso de mensajes (*send/receive*). Datos duplicados en memoria principal, copia datos

# Comparativa SMP (Symmetric Multi-Processor) y multicomputadores II

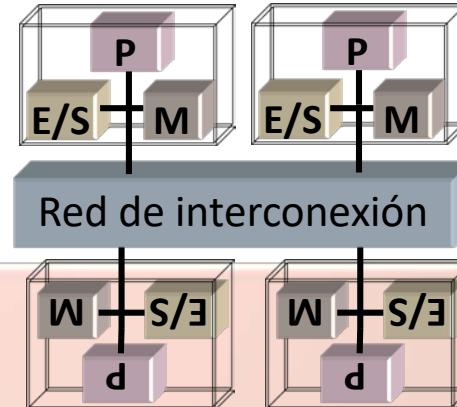


Multiprocesador con memoria centralizada (SMP)

Necesita implementar primitivas de sincronización

Distribución código y datos entre procesadores: no necesaria

Programación, generalmente, más sencilla



Multicomputador

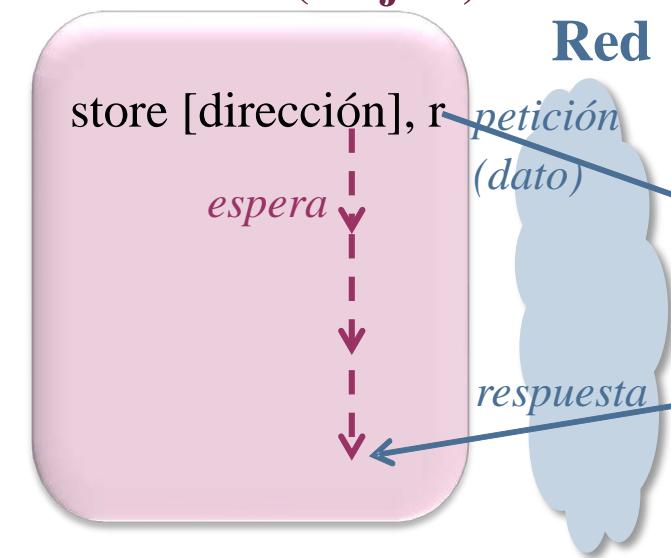
Sincronización mediante software de comunicación

Distribución código y datos entre procesadores: necesaria=> herramientas program. más sofisticadas

Programación generalmente más difícil

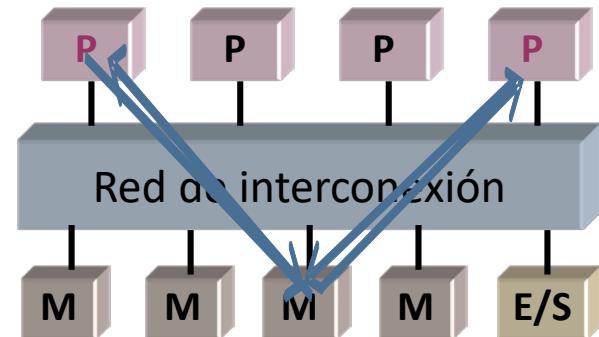
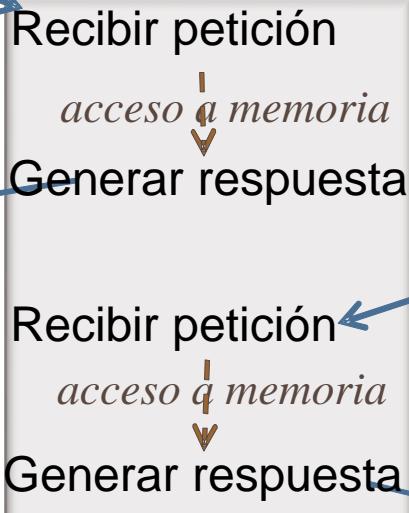
# Comunicación uno-a-uno en un multiprocesador

## Nodo fuente (Flujo 1)

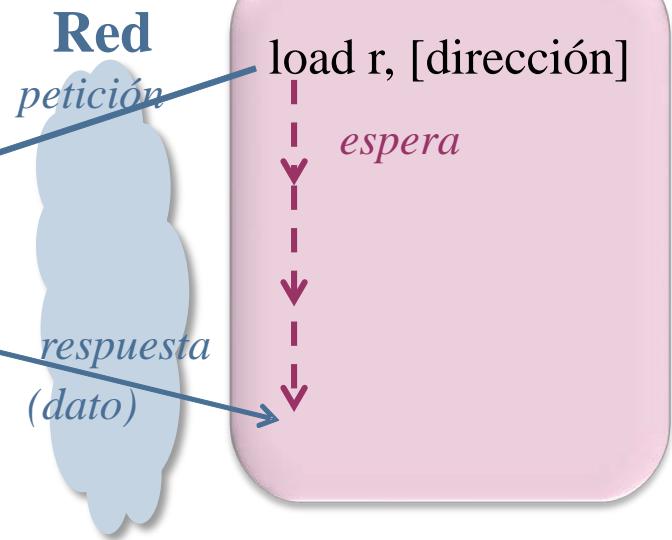


Red

Memoria



## Nodo destino (Flujo 2)



# Comunicación uno-a-uno en un multiprocesador

Secuencial	Paralelo	
... <b>A=valor;</b> ... copia= <b>A</b> ; ...	<b>F1</b> ... <b>A=valor;</b> ...	<b>F2</b> ... copia= <b>A</b> ; ...

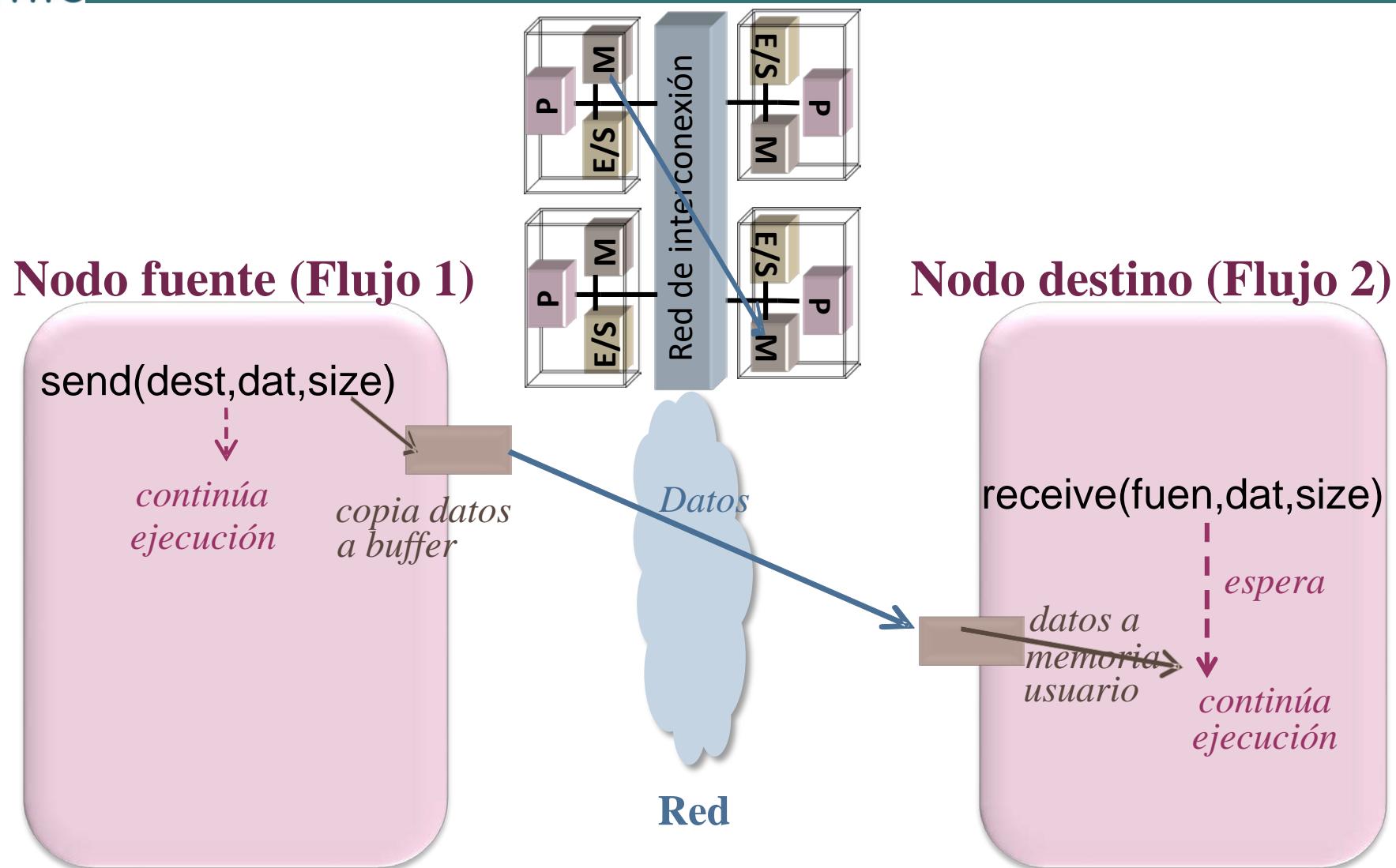
**F1** es el flujo de instrucciones productor del dato (envía el dato)  
**F2** es el flujo de instr. consumidor del dato (recibe el dato)

- Se debe garantizar que el flujo de instrucciones consumidor del dato lea la variable compartida (A) cuando el productor haya escrito en la variable el dato

Paralelo multiproc. ( $K=0$ )	
<b>F1</b> ... <b>A=valor;</b> <b>K=1;</b> ...	<b>F2</b> ... <b>while (K==0) { };</b> copia= <b>A</b> ; ...

NOTA: La programación paralela de multiprocesadores se estudia en AC

# Comunicación uno-a-uno en multicomputador (*receive* bloqueante)



# Comunicación uno-a-uno en un multicomputador

Secuencial	Paralelo	
... <b>A=valor;</b> ... <b>copia=A;</b> ...	<u>F1</u> ... <b>A=valor;</b> ...	<u>F2</u> ... <b>copia=A;</b> ...

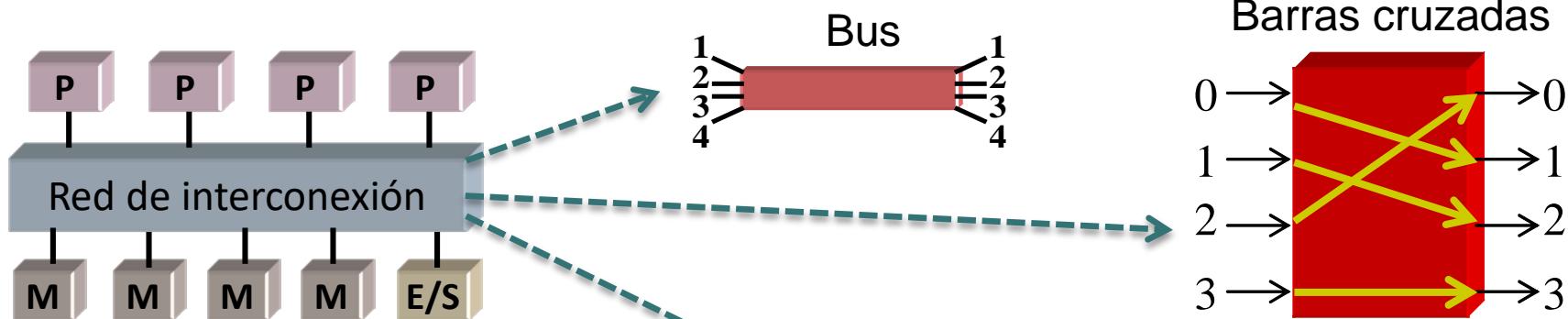
**F1** es el flujo de instrucciones productor del dato (*envía el dato*)  
**F2** es el flujo de instr. consumidor del dato (*recibe el dato*)

Paralelo multicomputador (size = 4 byte)	
<u>F1</u>	<u>F2</u>
... send(F2, valor, 4); ...	... receive(F1,copia,4); ...

NOTA: La programación paralela de multicomputadores se estudia en la asignatura: Arquitecturas y Computación de Altas Prestaciones (IC.SCAP.ACAP – Especialidad (IC), Materia (SCAP), Asignatura (ACAP))

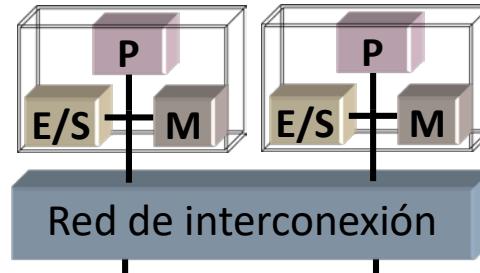
# Incremento de escalabilidad en multiprocesadores y red de interconexión

AC ATC

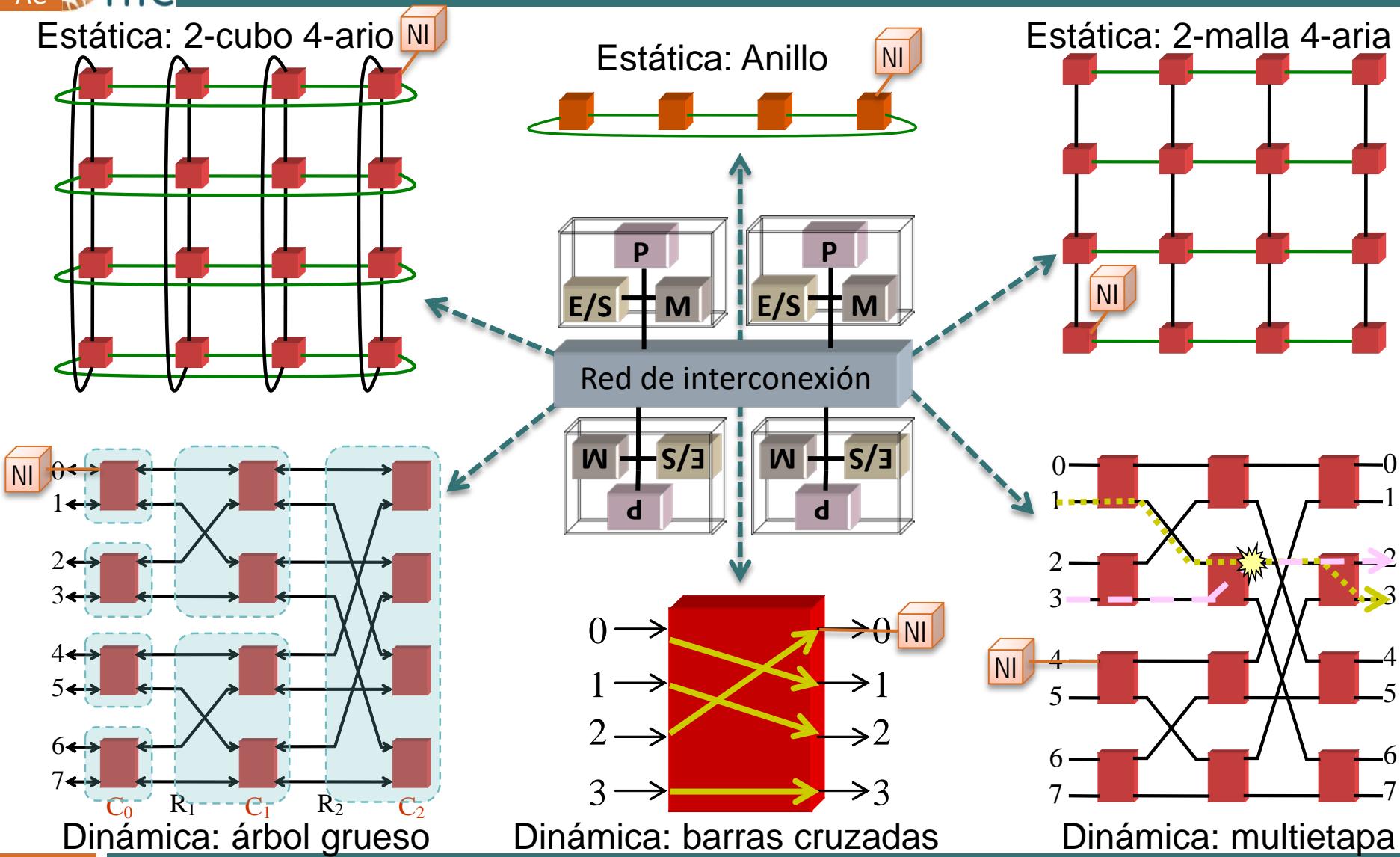


- Incremento escalabilidad multiprocesadores:
  - Aumentar cache del procesador
  - Usar redes de menor latencia y mayor ancho de banda que un bus (jerarquía de buses, barras cruzadas, multietapa)
  - Distribuir físicamente los módulos de memoria entre los procesadores (pero se sigue compartiendo espacio de direcciones)

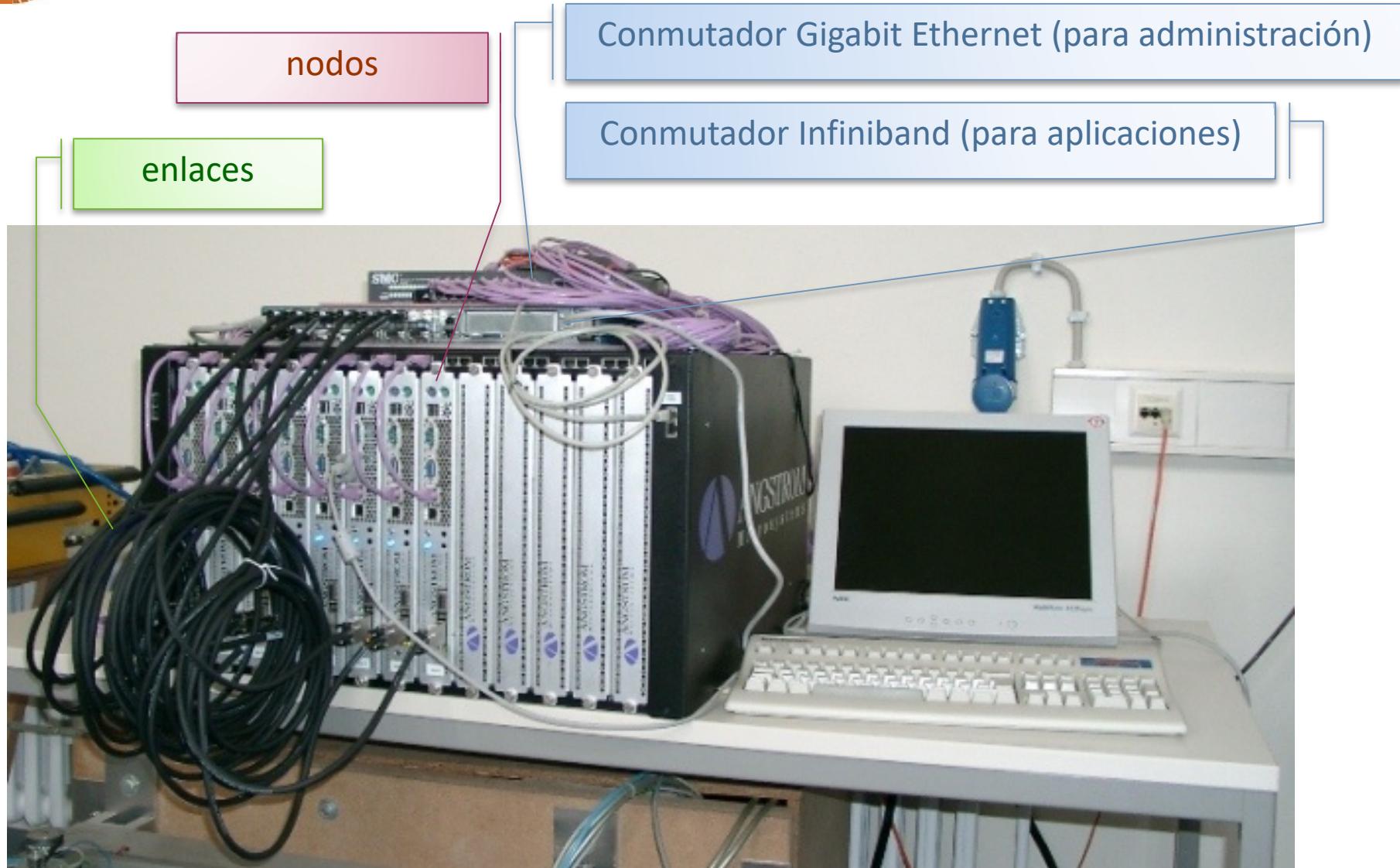
# Clasificación completa de computadores según el sistema de memoria

<b>Multi-computadores</b> Memoria no compartida	<b>NORMA</b> <i>No Remote Memory Access</i>	<i>ej. cluster, red de computadores</i>	Memoria físicamente distribuida	+ + Nivel de empaquetamiento y conexión	
	<b>NUMA</b> <i>Non-Uniform Memory Access</i>	<b>NUMA</b>			
	<b>CC-NUMA</b> <i>Non-Uniform Memory Access</i>	<b>CC-NUMA</b>			
		<b>COMA</b>			
<b>Multi-procesadores</b> Memoria compartida Un único espacio de direcciones	<b>UMA</b> <i>Uniform Memory Access</i>	<b>SMP Symmetric MultiProcessor</b>	Memoria físicamente centralizada	- - Escalabilidad	
	<b>NUMA</b> <i>Non-Uniform Memory Access</i>	<b>Red de interconexión</b>			

# Red en sistemas con memoria físicamente distribuida (NI: Network Interface)

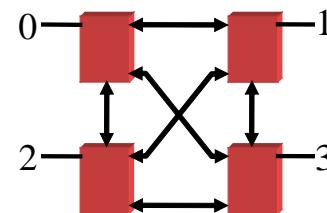
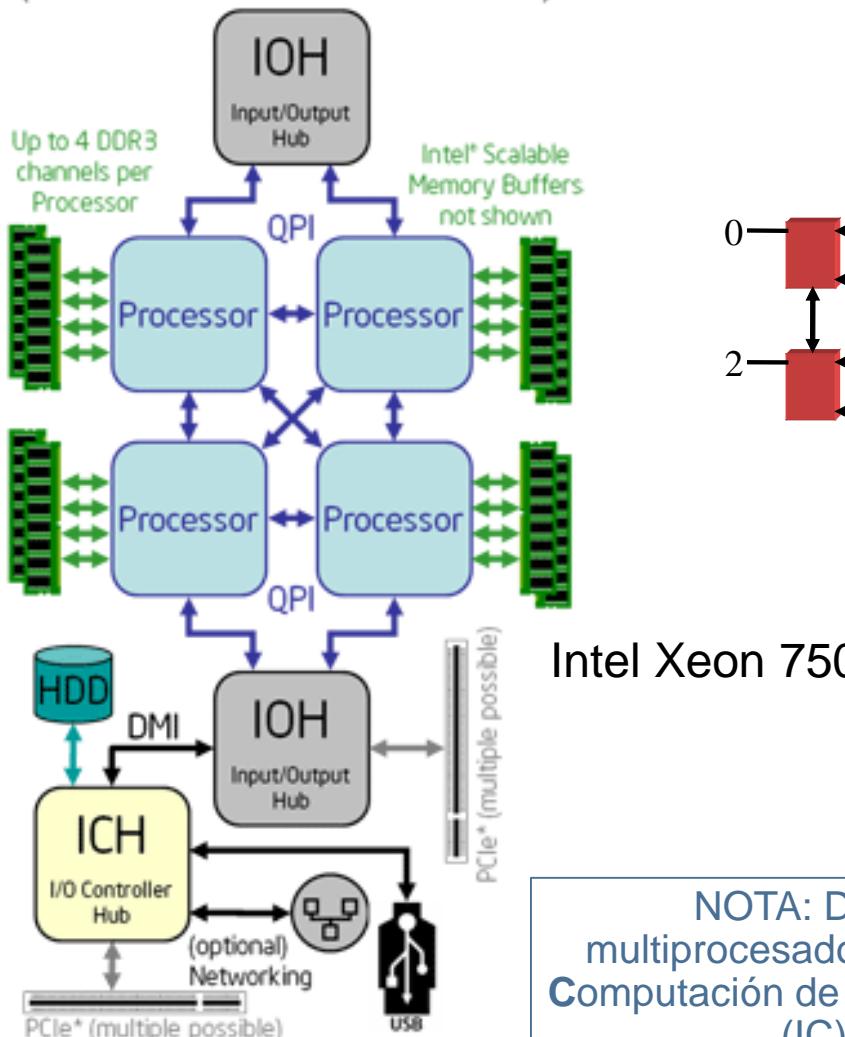


# Ejemplo: Red (con conmutador o *switch*) de barras cruzadas



# Ejemplo: Placa CC-NUMA con red estática

(ICH & other connects same as bottom if needed)



Intel Xeon 7500

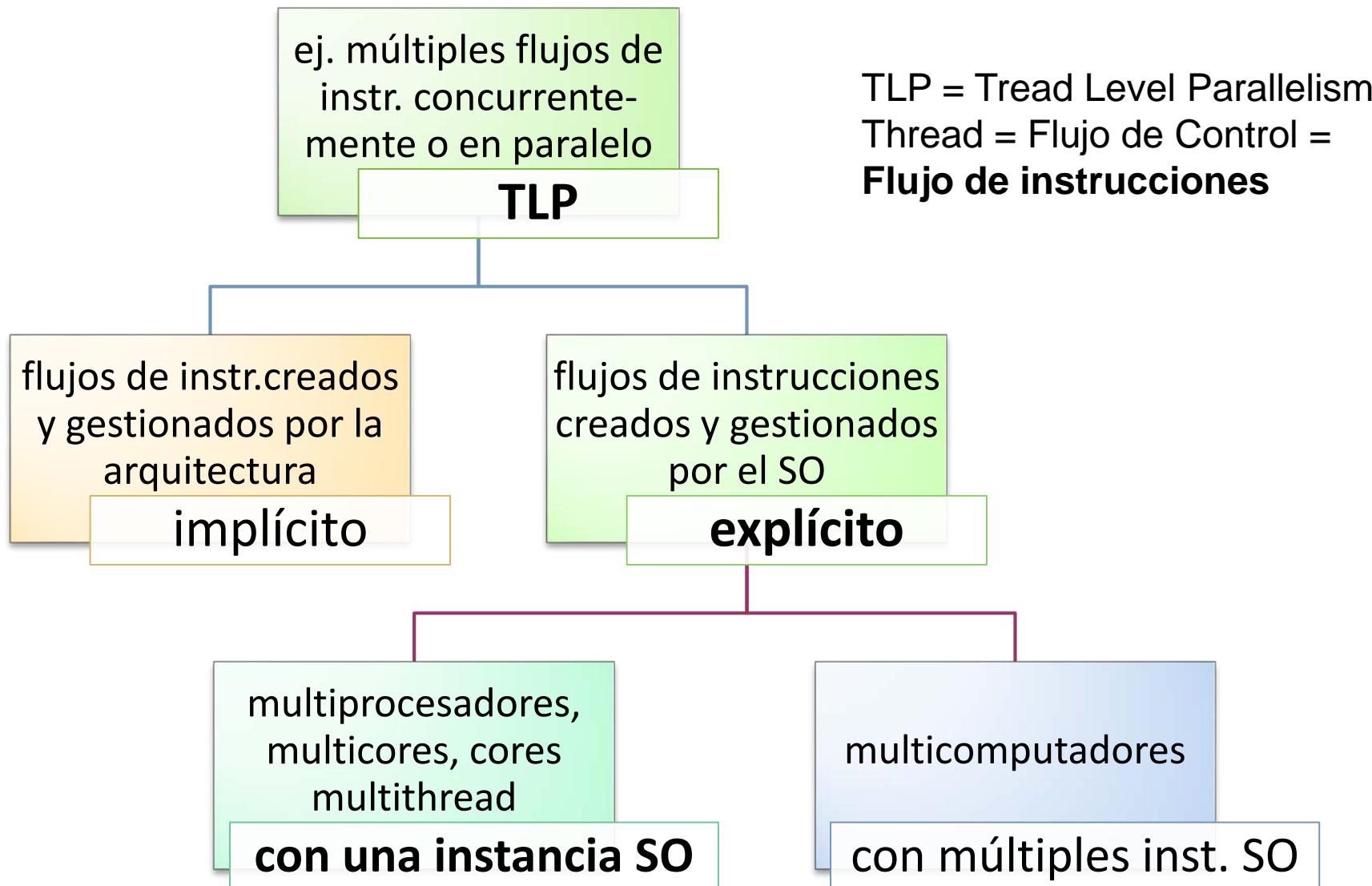


NOTA: Detalles de redes para multicomputadores y multiprocesadores se estudian en la asignatura: **Arquitecturas y Computación de Altas Prestaciones (IC.SCAP.ACAP – Especialidad (IC), Materia (SCAP), Asignatura (ACAP))**

# Criterios de clasificación de computadores

- Comercial
  - Segmento del mercado
    - embebidos, servidores gama baja ...
- Educación, investigación: también usados por fabricantes y vendedores
  - Flujos de instrucciones y flujos de datos (clasificación de Flynn 1972)
  - Sistema de memoria
  - ➡ Flujos de instrucciones (propuesta de clasificación de arquitecturas con múltiples flujos de instrucciones)
  - Nivel del paralelismo aprovechado (propuesta de clasif.)

# Propuesta clasificación computadores con múltiples flujos de instrucciones o threads



# Criterios de clasificación de computadores

- Comercial
    - Segmento del mercado
      - embebidos, servidores gama baja ...
  - Educación, investigación: también usados por fabricantes y vendedores
    - Flujos de instrucciones y flujos de datos (clasificación de Flynn 1972)
    - Sistema de memoria
    - Flujos de instrucciones (propuesta de clasificación de arquitecturas con múltiples flujos de instrucciones)
- Nivel del paralelismo aprovechado (propuesta de clasif.)

# Arquitecturas con DLP, ILP y TLP (thread=flujo de control o de instrucciones)

AC ATC

Arq. con **DLP**  
(*Data Level Parallelism*)

Tema 5  
Ejecutan las **operaciones** de una instrucción **concurr.** o en **paralelo**

Unidades funcionales vectoriales o SIMD

Arq. con **ILP**  
(*Instruction Level Parallelism*)

Tema 4  
Ejecutan múltiples **instrucciones concurr.** o en **paralelo**

Cores escalares segmentados, superescalares o VLIW/EPIC

Arq. con **TLP** (*Thread Level Parallelism*) explícito y **una** instancia de SO

Temas 3

Ejecutan múltiples **flujos de instrucciones** concurr. o en **paralelo**

Cores que modifican la arquit. escalar segmentada, superescalar o VLIW/EPIC para ejecutar threads *concurr.* o en *paralelo*

Multi-procesadores: ejecutan threads en *paralelo* en un computador con múltiples cores (incluye **multicores**)

Arq. con **TLP** explícito y **múltiples** instancias SO

IC.SCAP

Ejec. múltiples **flujos de instr.** en **paralelo**

Multi-computadores : ejecutan threads en *paralelo* en un sistema con **múltiples** computadores

# Contenidos

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
  - Computación paralela y computación distribuida
  - Clasificaciones de arquitecturas y sistemas paralelos
  - Nota histórica
- Lección 3. Evaluación de prestaciones

# Nota histórica. DLP y ILP

## ➤ DLP (*Data Level Parallelism*)

TEMA 5

### ➤ Unidades funcionales (o de ejecución) SIMD (o multimedia)

- **1989** (**Intel i860**). **1991** (motorola M88110). **1993** (repertorio MAX en HP PA : PA7100LC). **1995** (repertorio VIS en Sun Sparc: Ultra I). **1997** (repertorio MMX en Intel x86: Pentium MMX). **1999** (repertorio SSE en Intel x86: Pentium III; repertorio Altivec en IBM Power: PowerPC 8000)

## ➤ ILP (*Instruction Level Parallelism*) :

TEMA 4

### ➤ Procesadores/cores segmentados

- **1961** (**IBM 7030**). **1982** (chip Intel i286, Motorola 68020). **1986** (chip MIPS R2000). **1987** (chip AMD Am29000). **1988** (chip Sun Sparc)...

### ➤ Procesadores con múltiples unidades funcionales

- **1967** (IBM 360/91) ...

### ➤ Procesadores/cores superescalares

- **1989** (chip Intel 960CA (3)). **1990**: (chip IBM Power1 (4)). **1992**: (chips DEC  $\alpha$ 21064 (2/4), HP PA 7100 (2/2), Sun SuperSparc (3/5)). **1993**: (Pentium(2/3)) **1995**: (Pentium Pro (3/7)) ...

### ➤ Procesadores/cores VLIW

- **1990** (**chip Intel i860 (2)**). **1997** (**chip DSP TMS320C6x (8)**). **2001** (**chip Intel Itanium**)...

NOTA: Destacado en *cursiva* las primeras implementaciones y en **color** más claro los chip de propósito específico

# Nota histórica. TLP (*Thread Level Parallelism*)

- TLP explícito con una instancia de SO:
  - Multithread grano fino (FGMT)
    - 1975 (Denelcor HEP). **2005** (chip Sun UltraSPARC T1) ...
  - Multithread grano grueso (CGMT)
    - 1990 (MIT Alewife). **2000** (chip IBM PowerPC RS64 IV (2)). **2006** (chip Intel Itanium Montecito (2)) ...
  - Multithread simultánea (SMT)
    - **2002** (chip Intel Pentium 4/Xeon Hyper-Threading). **2004** (chip IBM Power5) ...
  - Multiprocesadores en un chip (CMP) o multicores
    - **2001** (chip IBM Power4). **2004** (chip Sun UltraSPARC IV). **2006** (chip Intel Core Duo).  
**2008** (chip Intel Celeron Dual-core) ...
  - Multiprocesadores
    - 1962 (Burroughs D825 - red barras cruzadas). 1966 (UNIVAC 1108 - red bus). 1985 (IBM RP3 - red multietapa). 1996 (SGI Origin 2000 -CC-NUMA, red estática+multietapa) .**2006** (SGI Altix 4000) ...
- TLP explícito con múltiples instancias del SO (multicomputadores) IC.SCAP
  - 1985 (Intel iPSC1 - i286+red estática con Ethernet) ... cualquier cluster

NOTA: Destacado en *cursiva* las primeras implementaciones y en **color** más claro los chip de propósito específico

# Para ampliar .....

- Páginas Web:
  - “Embedded Processors. 2010-14 Global Market Demand Analysis.” VDC Research Group.  
[http://www.vdcresearch.com/\\_Documents/proposal/pro-attachment-2637.pdf](http://www.vdcresearch.com/_Documents/proposal/pro-attachment-2637.pdf)
- Artículos de Revistas:
  - Ranakrishna Rau, B.; Schlansker, M.S.: “Embedded Computer Architecture and Automation”. IEEE Computer, pp.75-82. Abril, 2001.

2º curso / 2º cuatr.

Grado en  
Ing. Informática

# Arquitectura de Computadores

## Tema 1

# Arquitecturas Paralelas: Clasificación y Prestaciones

Material elaborado por los profesores responsables de la asignatura:

Mancia Anguita – Julio Ortega

Licencia Creative Commons



ugr

Universidad  
de Granada



# Lecciones

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones de una arquitectura
  - Medidas usuales para evaluar prestaciones
  - Conjunto de programas de prueba (*Benchmark*)
  - Ganancia en prestaciones

# Objetivos Lección 3

- Distinguir entre tiempo de CPU (sistema y usuario) de unix y tiempo de respuesta
- Distinguir entre productividad y tiempo de respuesta
- Obtener, de forma aproximada mediante cálculos, el tiempo de CPU, GFLOPS y los MIPS del código ejecutado en un núcleo de procesamiento
- Explicar el concepto de ganancia en prestaciones
- Aplicar la ley de Amdahl

# Bibliografía

## ➤ Fundamental

- Capítulo 1, M. Anguita, J. Ortega. Fundamentos y problemas de Arquitectura de Computadores, Editorial Técnica Avicam. ESIIT/C.1 ANG fun
- Secciones 1.2, 1.4, 7.5.1. J. Ortega, M. Anguita, A. Prieto. *Arquitectura de Computadores*, Thomson, 2005. ESIIT/C.1 ORT arq

## ➤ Complementaria

- T. Rauber, G. Ründer. *Parallel Programming: for Multicore and Cluster Systems*. Springer 2010. Disponible en línea (biblioteca UGR): <http://dx.doi.org/10.1007/978-3-642-04818-0>

# Contenido

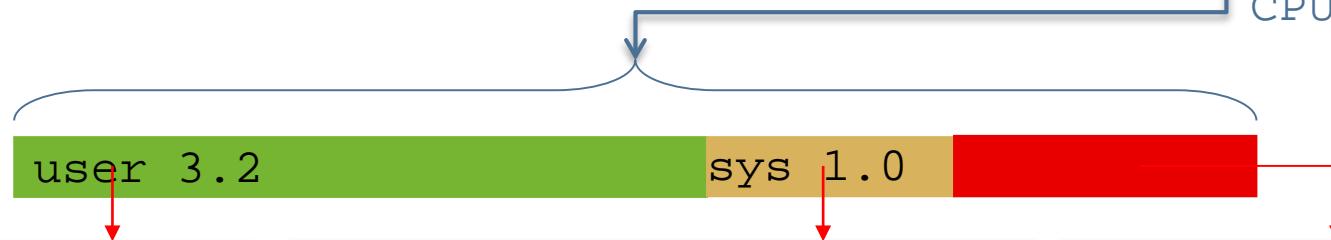
- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones de una arquitectura
  - **Medidas usuales para evaluar prestaciones**
    - Tiempo de respuesta
    - Productividad
  - Ganancia en prestaciones al realizar una mejora
  - Conjunto de programas de prueba (*Benchmark*)

# Tiempo de respuesta de un programa en una arquitectura

- Real (*wall-clock time, elapsed time, real time*)
- CPU time = user + sys (no incluye todo el tiempo)
- Con un flujo de instrucciones
  - elapsed  $\geq$  CPU time
- Con múltiples flujos de instrucciones
  - elapsed < CPU time, elapsed  $\geq$  CPU time/nº flujos control

```
$ time ./program.exe
elapsed 5.4
user 3.2
sys 1.0
```

Elapsed  $\geq$  CPU time



**Tiempo de CPU de usuario** (Tiempo en ejecución en espacio de usuario)

**Tiempo de CPU de sistema**  
(Tiempo en el nivel del kernel del SO)

**Tiempo** asociado a las esperas debidas a I/O o asociados a la ejecución de otros programas.

Comando **time** en Unix:

3.2u 1.0s 5.4

3.2+1.0 es el **78%** del tiempo transcurrido (5.4)

# Algunas alternativas para obtener tiempos

Función	Fuente	Tipo	Resolución aprox. (microsegundos)
<code>time</code>	SO (/usr/bin/time)	<i>elapsed, user, system</i>	10000
<code>clock() /CLOCKS_PER_SEC</code>	SO (time.h)	<i>CPU</i>	10000
<code>gettimeofday()</code>	SO (sys/time.h)	<i>elapsed</i>	1
<code>clock_gettime() /clock_getres()</code>	SO (time.h)	<i>elapsed</i>	0.001
<code>omp_get_wtime() /omp_get_wtick()</code>	OpenMP (omp.h)	<i>elapsed</i>	0.001
<code>SYSTEM_CLOCK()</code>	Fortran	<i>elapsed</i>	1

La resolución depende de la plataforma

# Tiempo de CPU

```
...
for (i=0; i<N; i++) {
    v3[i]=v1[i]+v2[i];
}
...
```

$$T_{CPU} = NI \times CPI \times T_{ciclo}$$

$$T_{CPU} = NI \times \frac{1}{IPC} \times T_{ciclo}$$

$$T_{CPU} = \frac{\text{Nº ciclos}}{\text{código}} \times T_{ciclo}$$

$$T_{CPU} = \left[ \sum_i NI_i \times CPI_i \right] \times T_{ciclo}$$

$$T_{CPU} = NI \times \left( \frac{\sum_i NI_i \times CPI_i}{NI} \right) \times T_{ciclo}$$

$$T_{CPU} = NI \times CPI \times T_{ciclo}$$

.L7:

... ; rax=0, rbx=8N  
 movsd v1(%rax), %xmm0  
 addsd v2(%rax), %xmm0  
 movsd %xmm0, v3(%rax)  
 addq \$8, %rax  
 cmpq %rbx, %rax  
 jne .L7  
 ...

$i$	$NI_i$	$CPI_i$
movsd m,r	2N	4
movsd r,m	N	5
addsd m,r	N	1
addq i,r	N	1
cmp r,r	N	1
jne	N	1
	<b>6N</b>	

$$T_{ciclo} = 1/F$$

Para  $N = 10^3$  y  $F = 100\text{MHz}$  ( $\Rightarrow T_{ciclo} = 10^{-8}\text{seg./ciclo}$ ):

$$\begin{aligned} T_{CPU} &\approx \left[ 6N \times \left( \frac{2N \times 4 + N \times 5 + 3N \times 1}{6N} \right) \right] \times T_{ciclo} \\ &= 10^3 \times 16 \text{ ciclos/código} \times 10^{-8}\text{seg./ciclo} \\ &= 16 \times 10^{-5}\text{seg./código} \end{aligned}$$

# Tiempo de CPU

$$T_{CPU} = NI \times CPI \times T_{ciclo}$$

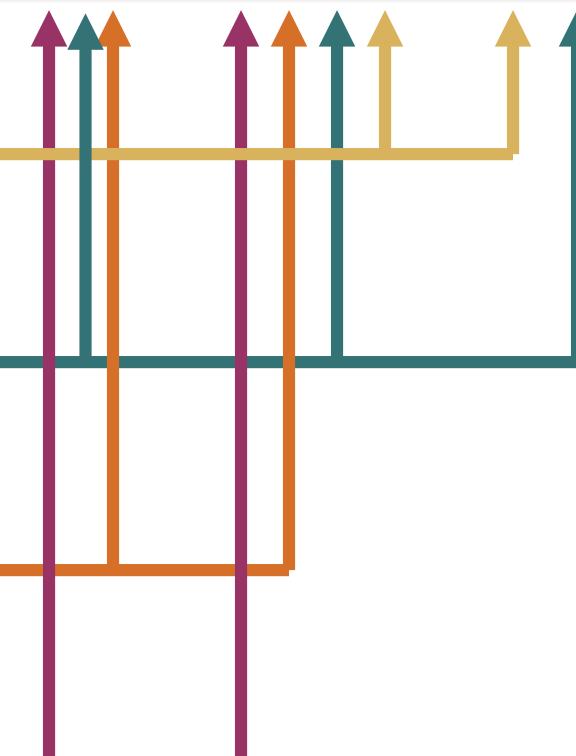


Tecnología

Estructura y  
Organización

Repertorio de  
Instrucciones

Compilador



# MIPS y MFLOPS

- Millones de instrucciones por segundo (MIPS):

$$MIPS = \frac{NI}{T_{CPU} \times 10^6}$$

$$MIPS = \frac{NI}{NI \times CPI \times T_{ciclo} \times 10^6} = \frac{F}{CPI \times 10^6}$$

- Depende del repertorio de instrucciones (difícil la comparación de máquinas con repertorios distintos)
- Puede variar inversamente con las prestaciones (mayor valor de MIPS corresponde a peores prestaciones)
- Millones de operaciones punto flotante por segundo (MFLOPS):

$$MFLOPS = \frac{nº FP}{T_{CPU} \times 10^6}$$

# MIPS y FLOPS

```
...
for (i=0; i<N; i++) {
    y[i]=a*x[i]+y[i];
}
...
```

```
;r12=&x,r13=&y,rax=0,rbp=N,xmm1=a
.L6:
    movsd (%r12,%rax,8), %xmm0
    mulsd %xmm1, %xmm0
    addsd (%r13,%rax,8), %xmm0
    movsd %xmm0, (%r13,%rax,8)
    addq $1, %rax
    cmpl %eax, %ebp
    jg .L6
```

$$T(N=2^{26}) = 0.182 \text{ seg.}$$

$$\begin{aligned} GIPS &= \frac{NI}{T_{CPU} \times 10^9} = \frac{N \times 7}{0.182 \times 10^9} \\ &= \frac{2^{26} \times 7}{0.182 \times 10^9} \approx 2.58 \text{ GIPS} \end{aligned}$$

$$\begin{aligned} GFLOPS &= \frac{n^o FP}{T_{CPU} \times 10^9} = \frac{N \times 2}{0.182 \times 10^9} \\ &= \frac{2^{26} \times 2}{0.182 \times 10^9} \approx 0.737 \text{ GFLOPS} \end{aligned}$$

```
;r12=&x,r13=&y,rax=0,rbp=N/2,xmm1=a
.L7:
```

```
    movapd (%r12), %xmm0
    addq $1, %rax
    addq $16, %r12
    addq $16, %r13
    mulpd %xmm1, %xmm0
    addpd -16(%r13), %xmm0
    movaps %xmm0, -16(%r13)
    cmpl %ebp, %eax
    jb .L7
```

$$T(N=2^{26}) = 0.178 \text{ seg.}$$

$$\begin{aligned} GIPS &= \frac{NI}{T_{CPU} \times 10^9} = \frac{(N/2) \times 9}{0.178 \times 10^9} \\ &= \frac{2^{25} \times 9}{0.178 \times 10^9} \approx 1.7 \text{ GIPS} \end{aligned}$$

$$GFLOPS = \frac{2^{26} \times 2}{0.178 \times 10^9} \approx 0.754 \text{ GFLOPS}$$

# Lecciones

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones de una arquitectura
  - Medidas usuales para evaluar prestaciones
  - **Ganancia en prestaciones al realizar una mejora**
  - Conjunto de programas de prueba (*Benchmark*)

# Mejora o Ganancia en Prestaciones (*Speed-up* o ganancia en velocidad)

Si se incrementan las prestaciones de un sistema, el incremento en prestaciones (velocidad) que se consigue en la nueva situación,  $p$ , con respecto a la previa (**sistema base**,  $b$ ) se expresa mediante la ganancia en prestaciones o *speed-up*,  $S$

$$S = \frac{V_p}{V_b} = \frac{T_b}{T_p}$$

$$S = \frac{T_{CPU}^b}{T_{CPU}^p} = \frac{NI^b \times CPI^b \times T_{ciclo}^b}{NI^p \times CPI^p \times T_{ciclo}^p}$$

$V_b$  Velocidad de la máquina base

$V_p$  Velocidad de la máquina mejorada (un factor  $p$  en uno de sus componentes)

$T_b$  Tiempo de ejecución en la máquina base

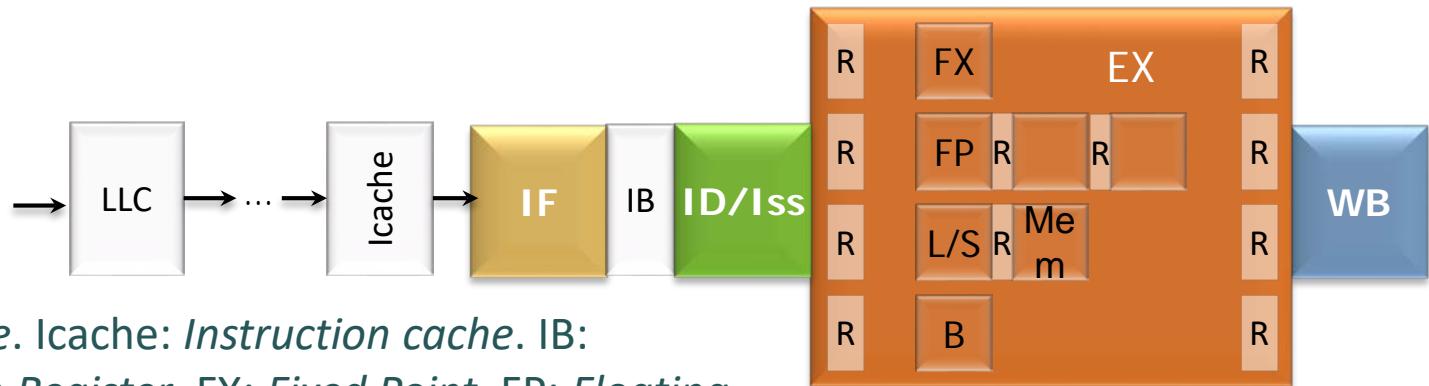
$T_p$  Tiempo de ejecución en la máquina mejorada

# Arquitecturas con paralelismo a nivel de instrucción (ILP)

Escalar  
segmentada



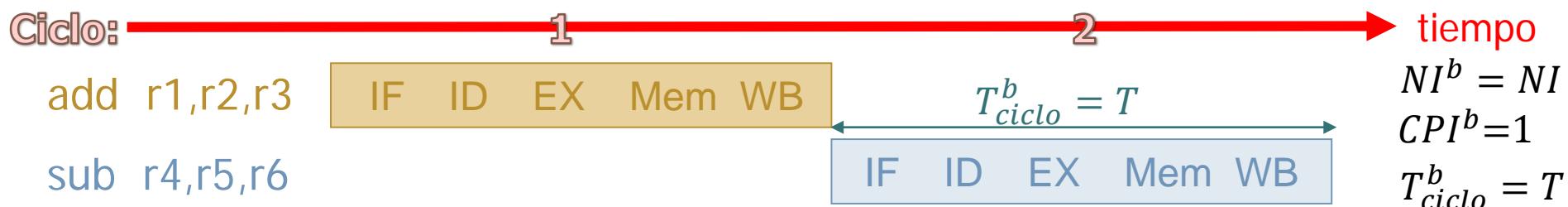
VLIW y  
superescalar



LLC: *Last Level Cache*. Icache: *Instruction cache*. IB:  
*Instruction Buffer*. R: *Register*. FX: *Fixed Point*. FP: *Floating Point*. L/S: *(memory) Load/Store*. B: *Branch*

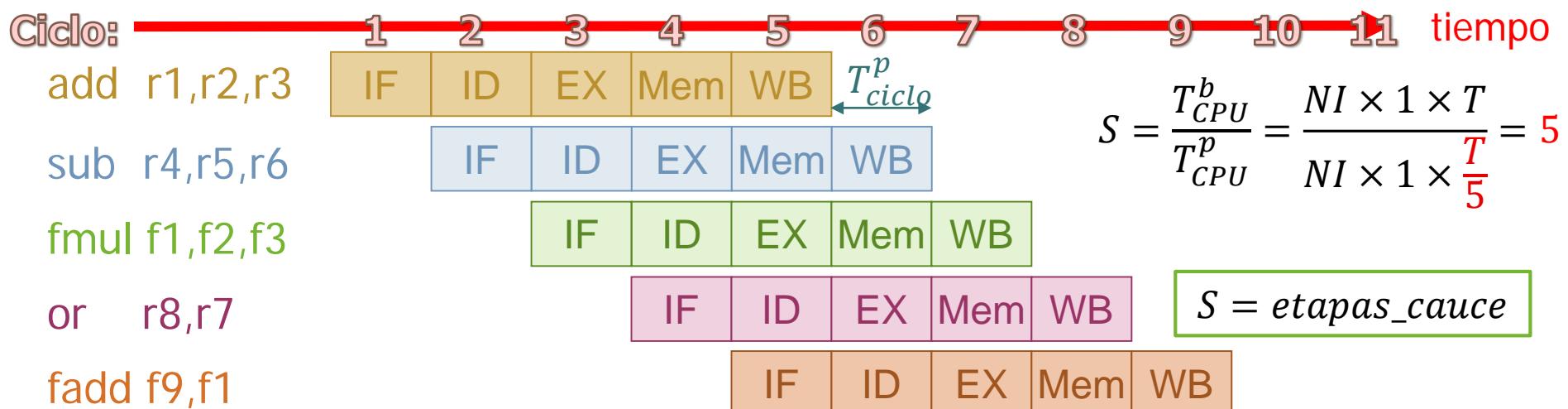
- Etapa de captación de instrucciones (*Instruction Fetch*)
- Etapa de decodificación de instrucciones y emisión a unidades funcionales (*Instruction Decode/Instruction Issue*) (incluye captura de "operандos" de los registros de la arquitectura)
- Etapas de ejecución (*Execution*). Etapa de acceso a memoria (*Memory*)
- Etapa de almacenamiento de resultados en registros de la arquitectura (*Write-Back*)

# Mejora en un núcleo de procesamiento: segmentación

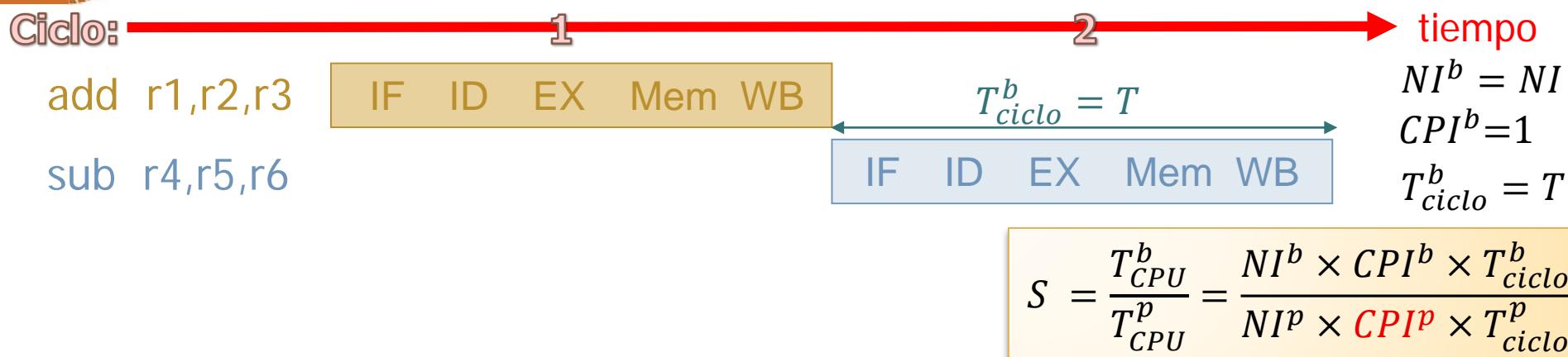


$$S = \frac{T_{CPU}^b}{T_{CPU}^p} = \frac{NI^b \times CPI^b \times T_{ciclo}^b}{NI^p \times CPI^p \times T_{ciclo}^p}$$

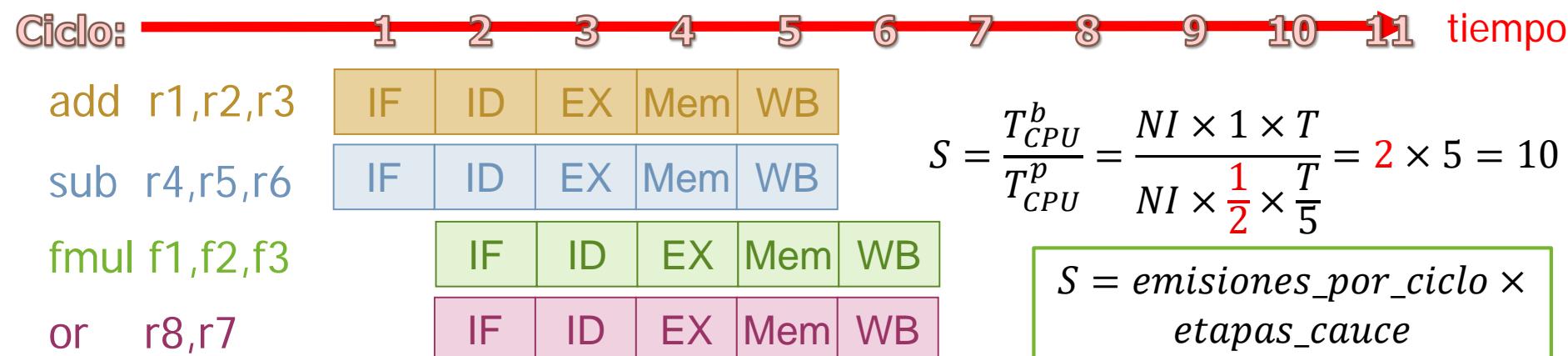
Núcleo segmentado en 5 etapas:



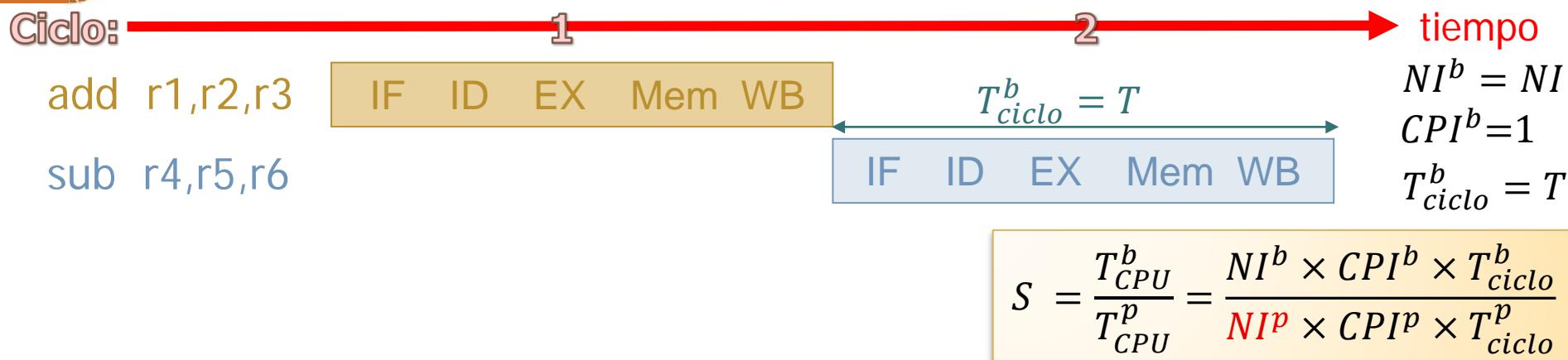
# Mejora en un núcleo de procesamiento: operación superscalar



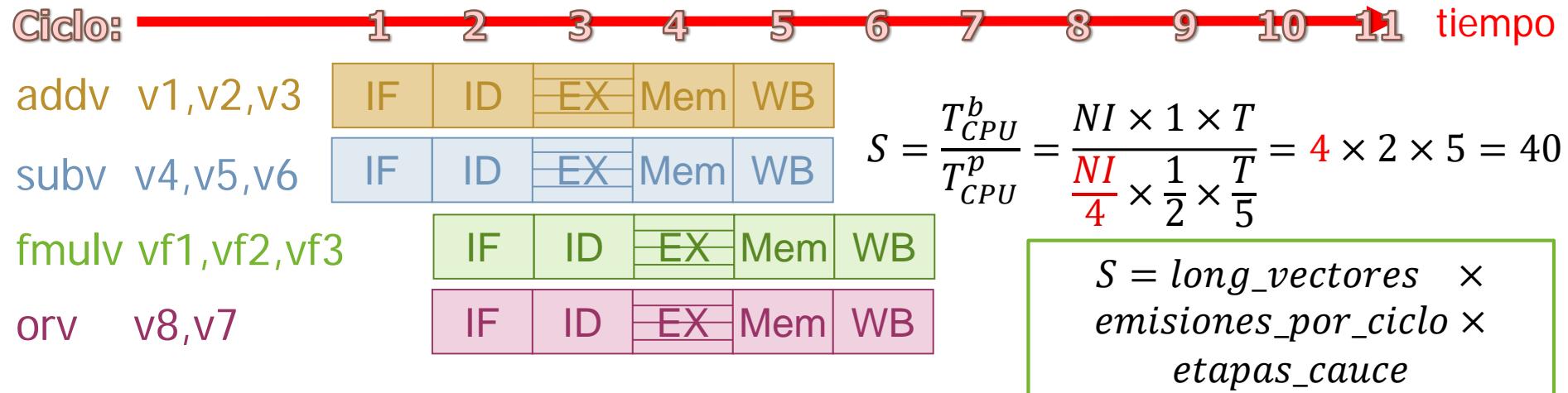
Núcleo superescalar con 2 emisiones por ciclo y 5 etapas:



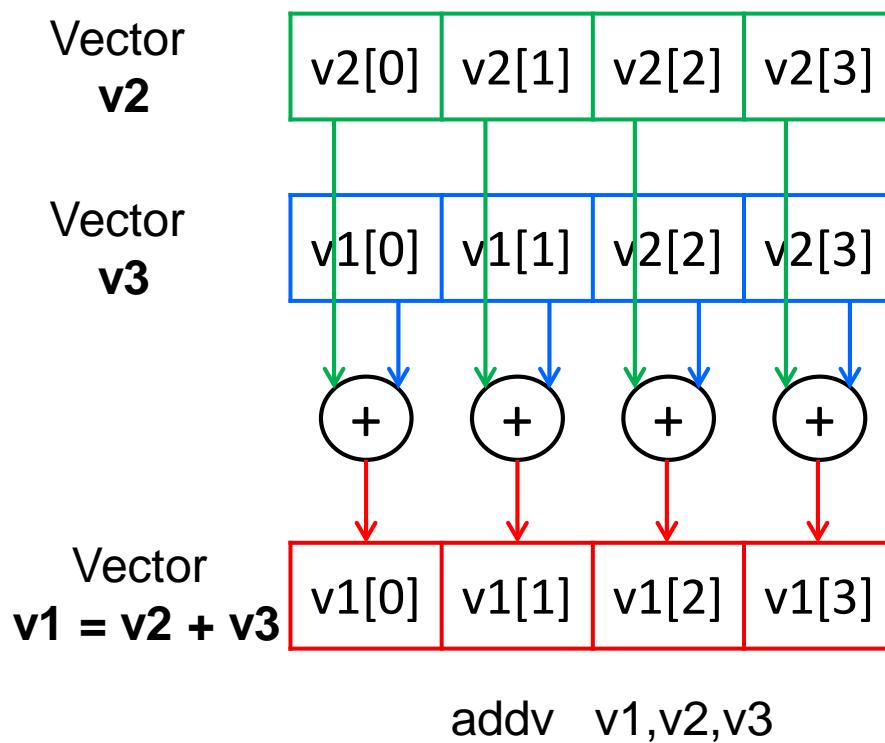
# Mejora en un núcleo de procesamiento: unidades funcionales SIMD



Núcleo **superescalar** con **2 emisiones por ciclo** y **5 etapas**, y  
unidades funcionales SIMD (vectoriales) que procesan **vectores de 4 componentes**:



# Paralelismo de datos. Ej: suma de dos vectores

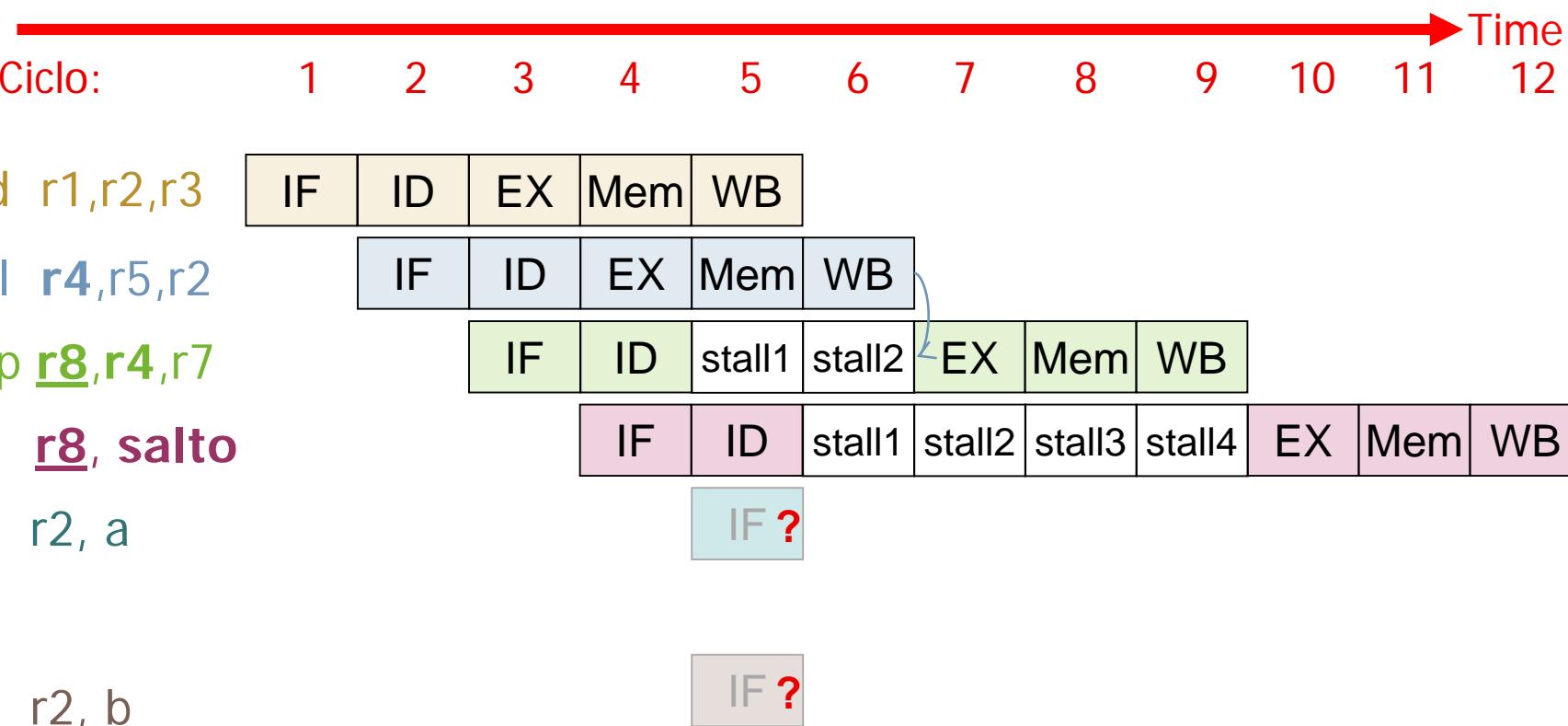


# ¿Qué impide que se pueda obtener la ganancia en velocidad pico?

- Riesgos:
  - Datos
  - Control
  - Estructurales
- Accesos a memoria (debido a la jerarquía)



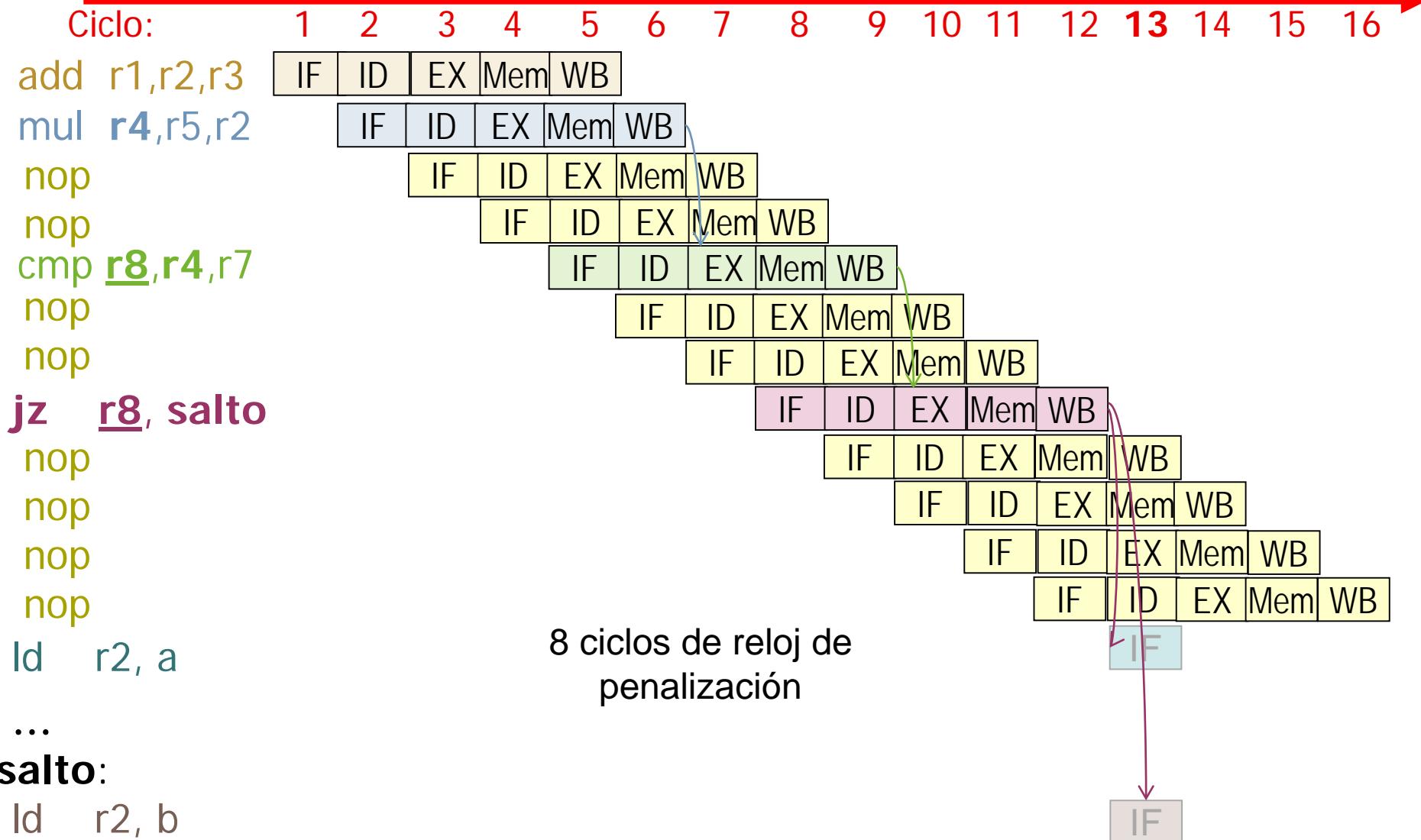
# Riesgos (*hazards*): de datos, de control



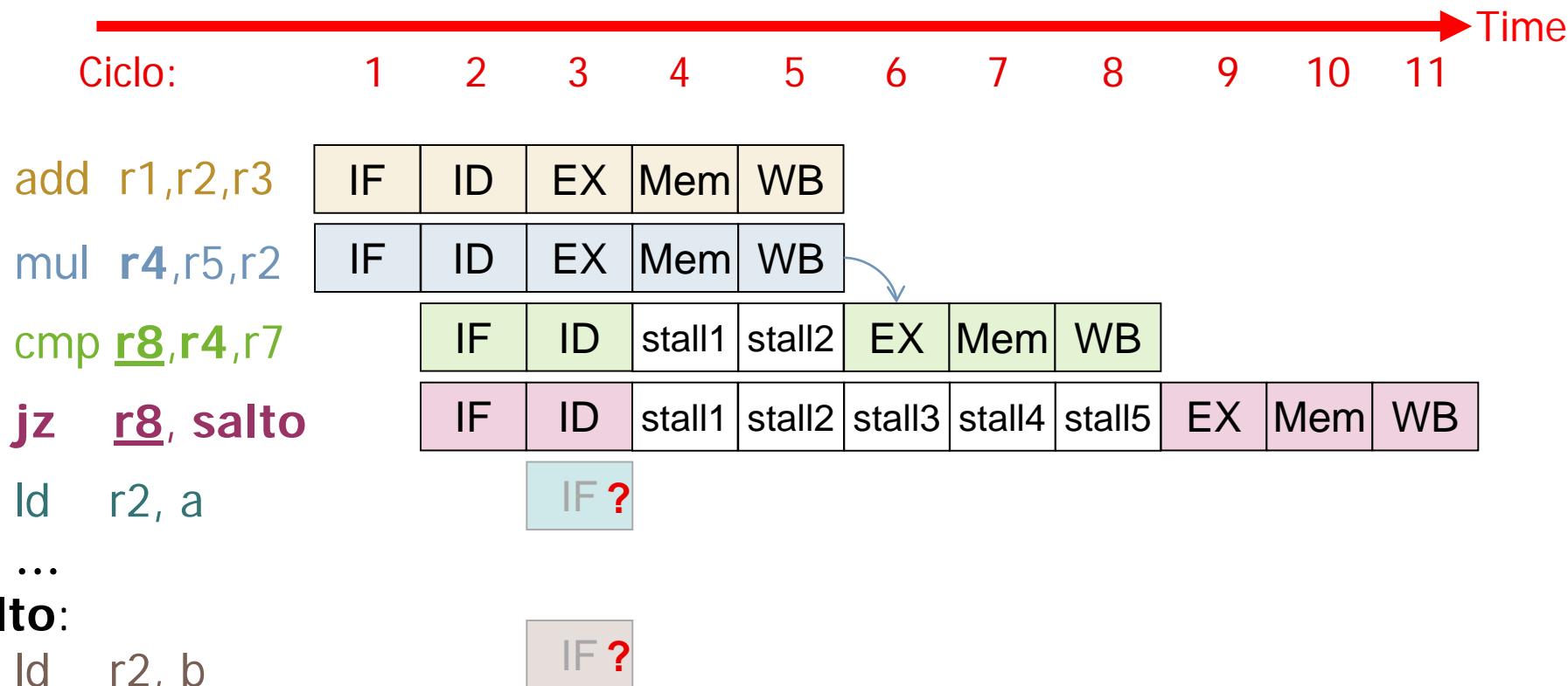
8 ciclos de reloj de  
penalización

$$T_{CPU} = NI \times CPI \uparrow \times T_{ciclo}$$

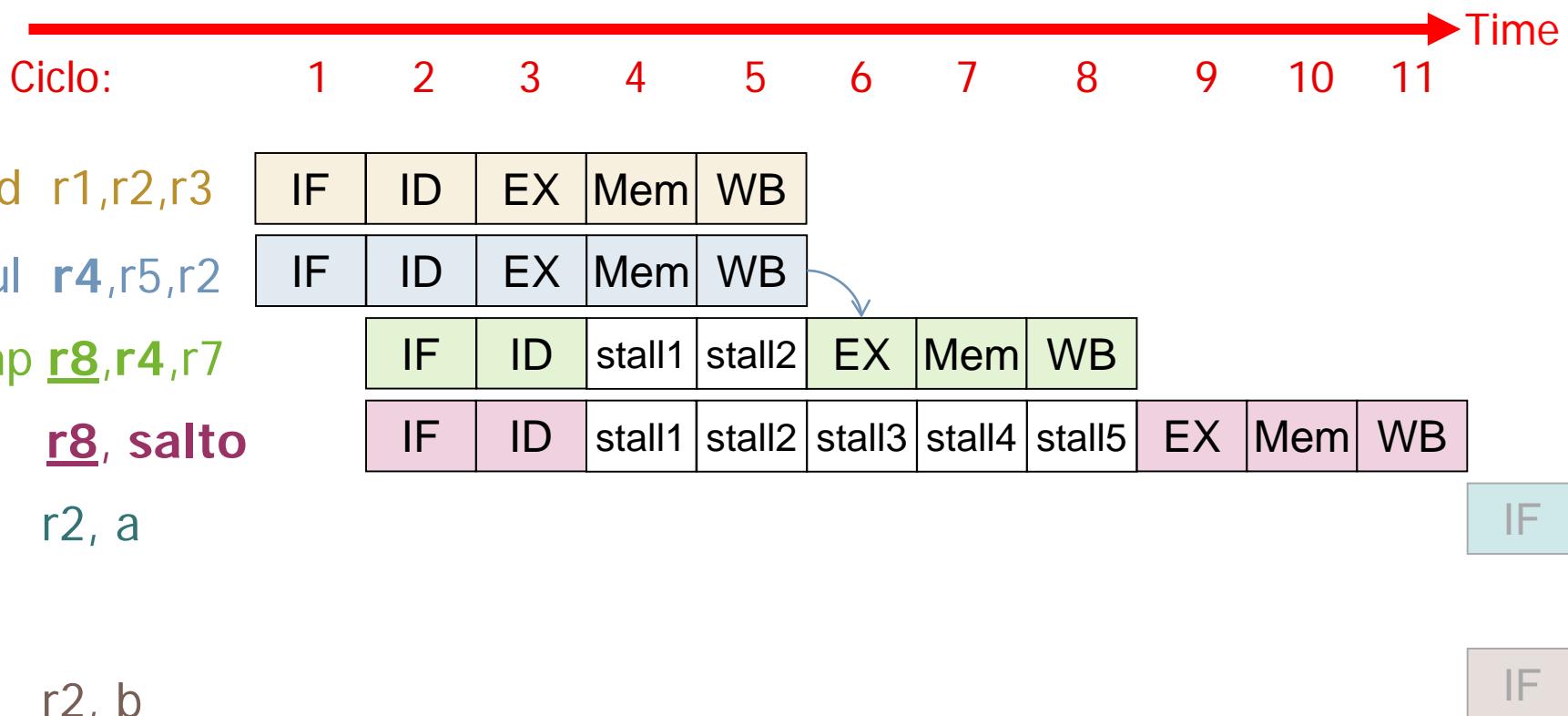
# Riesgos (*hazards*): de datos, de control



# Riesgos (*hazards*): de datos, de control



# Riesgos (*hazards*): de datos, de control



9 ciclos de reloj de penalización

Modificación del contador de programa

# Riesgos (*hazards*): de datos, de control, estructural



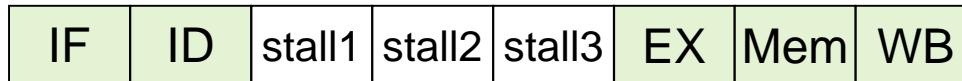
**add r1,r2,r3**



**mul r4,r5,r2**



**cmp r8,r4,r7**



**jz r8, salto**



**ld r2, a**

IF ?

...

**salto:**

**ld r2, b**

IF ?

add y mul usan la misma unidad funcional

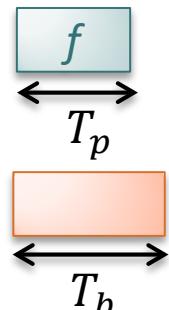
10 ciclos de reloj de  
penalización

# Ley de Amdahl

La mejora de velocidad,  $S$ , que se puede obtener cuando se mejora un recurso de una máquina en un factor  $p$  está limitada por:

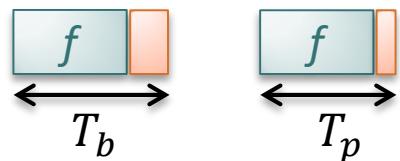
$$S = \frac{V_p}{V_b} = \frac{T_b}{T_p} \leq \frac{1}{f + \frac{1-f}{p}} = \frac{p}{1+f(p-1)}$$

$\xrightarrow{p \rightarrow \infty} \frac{1}{f}$   
 $\xrightarrow{f \rightarrow 0} p$



donde  $f$  es la fracción del tiempo de ejecución del sistema base (i.e. de la máquina sin la mejora) durante el que no se usa dicha mejora.

**Ejemplo:** Si un programa pasa un 25% de su tiempo de ejecución en una máquina realizando instrucciones de coma flotante, y se mejora la máquina haciendo que estas instrucciones se ejecuten en la mitad de tiempo, entonces  $p=2$ ,  $f=0.75$  y



$$S = \frac{T_b}{T_p} = \frac{1}{0.75 + \frac{0.25}{2}} \approx 1.14$$

**Habría que mejorar el caso más frecuente (lo que más se usa)**

# Lecciones

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones de una arquitectura
  - Medidas usuales para evaluar prestaciones
  - Ganancia en prestaciones al realizar una mejora
  - **Conjunto de programas de prueba (*Benchmark*)**

# Benchmarks

- Propiedades exigidas a medidas de prestaciones:
  - **Fiabilidad => Representativas, evaluar diferentes componentes del sistema y reproducibles**
  - Permitir comparar diferentes realizaciones de un sistema o diferentes sistemas => Aceptadas por todos los interesados (usuarios, fabricantes, vendedores)
- Interesados:
  - Vendedores y fabricantes de hardware o software.
  - Investigadores de hardware o software.
  - Compradores de hardware o software.

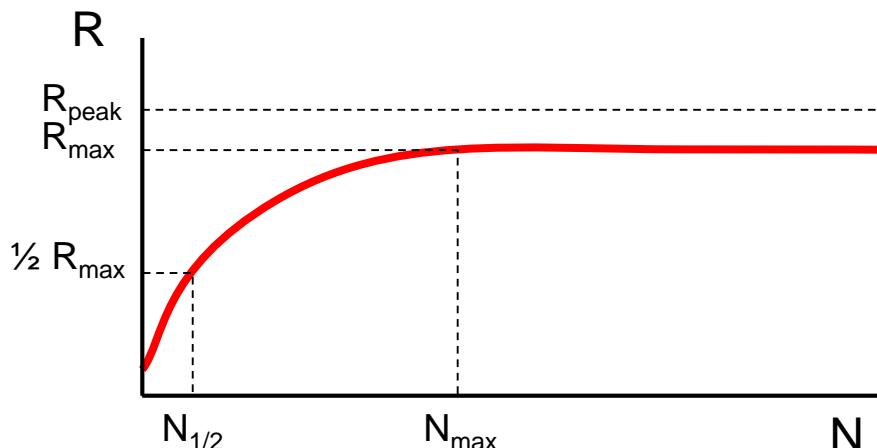
# Tipos de *Benchmarks*

- Tipos de Benchmark:
  - De bajo nivel o microbenchmark
    - test ping-pong, evaluación de las operaciones con enteros o con flotantes
  - Kernels
    - resolución de sistemas de ecuaciones, multiplicación de matrices, FFT, descomposición LU
  - Sintéticos
    - Dhrystone, Whetstone
  - Programas reales
    - SPEC CPU2017: enteros (gcc, perlbench),
  - Aplicaciones diseñadas
    - Predicción de tiempo, dinámica de fluidos, animación etc. (p. ej. SPEC2017).

# LINPACK

El núcleo de este programa es una rutina denominada DAXPY (*Double precision- real Alpha X Plus Y*) que multiplica un vector por una constante y los suma a otro vector. Las prestaciones obtenidas se escalan con el valor de **N** (tamaño del vector):

```
for (i=0 ; i<N ; i++)
    y[i] = alpha*x[i] + y[i];
```



<https://www.top500.org/project/linpack/>  
**TOP500**

$R_{\max}$ : Número máximo de Gflops alcanzados

$R_{\text{peak}}$ : Límite teórico del sistema (en Gflops)

Rank	Site Country/Year	Computer / Processors Manufacturer	$R_{\max}$ $R_{\text{peak}}$
1	<u>DOE/NNSA/LLNL</u> United States/2005	<u>BlueGene/L</u> <u>eServer Blue Gene Solution</u> / 65536 IBM	136800 183500
2	<u>IBM Thomas J. Watson</u> <u>Research Center</u> United States/2005	<u>BGW</u> <u>eServer Blue Gene Solution</u> / 40960 IBM	91290 114688
3	<u>NASA/Ames Research</u> <u>Center/NAS</u> United States/2004	<u>Columbia</u> <u>SGI Altix 1.5 GHz, Voltaire</u> <u>Infiniband</u> / 10160 SGI	51870 60960
4	<u>The Earth Simulator Center</u> Japan/2002	<u>Earth-Simulator</u> / 5120 NEC	35860 40960
5	<u>Barcelona Supercomputer</u> <u>Center</u> Spain/2005	<u>MareNostrum</u> <u>JS20 Cluster, PPC 970, 2.2 GHz,</u> <u>Myrinet</u> / 4800 IBM	27910 42144
6	<u>ASTRON/University</u> <u>Groningen</u> Netherlands/2005	<u>eServer Blue Gene Solution</u> / 12288 IBM	27450 34406.4

# Para ampliar .....

- Páginas Web:
  - <http://www.top500.org>
  - <http://en.wikipedia.org/wiki/LINPACK>
- Artículos de Revistas:
  - Henning, J.L.: “SPEC CPU2000: Measuring CPU Performance in the New Millennium”. IEEE Computer. Julio, 2000.
  - O’Neal, D.: “On Microprocessors, Memory Hierarchies, and Amdahl’s Law”.