

LECCIÓN 1: Clasificación del paralelismo implícito en una aplicación

Criterios de clasificaciones del paralelismo implícito en una aplicación

A la hora de clasificar el paralelismo implícito de una aplicación hay que tener en cuenta que, los niveles de paralelismo implícito en un código que resuelven la aplicación, el paralelismo de tareas, el de datos, y la granulidad.

Niveles de paralelismo implícito en una aplicación

Niveles y granulidad correspondiente:

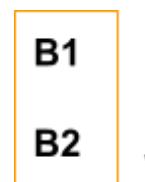
- Programas: Programa1, Programa2,... ➔ **Grano Grueso**
- Funciones: Func1, Func2, Func3, ➔ **Grano Medio**
- Bucle (bloques): for, while,... ➔ **Grano Medio-Fino**
- Operaciones: * + / ➔ **Grano Fino**

En los bucles, sí no hay dependencias **RAW**, entre las iteraciones, sí que se pueden ejecutar en paralelo. Lo mismo a nivel de funciones, operaciones y a nivel de programas.

Dependencias de datos

Condiciones que se deben cumplir para que el bloque de código **B2** presente dependencia de datos con respecto a **B1**:

- Deben hacer referencia a una misma posición de memoria M.
- **B1** aparece en la secuencia de código antes que **B2**



Tipos de dependencias de datos:

- **RAW (Read After Write)** o dependencia verdadera ➔ Las operaciones con este tipo de dependencias no puede hacer paralelismo.
- **WAW (Write After Write)** o dependencia de salida.
- **WAR (Write After Read)** o antidependencia.

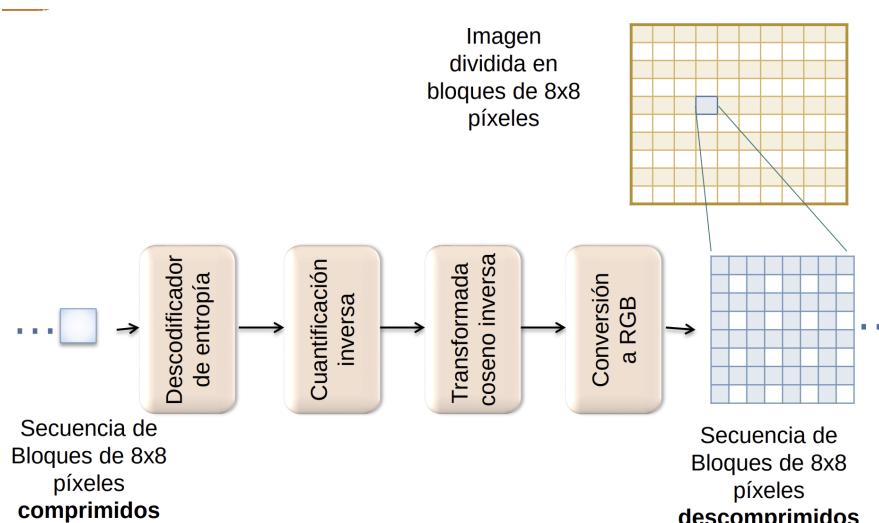
Estas dos últimas dependencias (WAW y WAR) se pueden eliminar fácilmente. De hecho, el propio hardware las quita.

Paralelismo implícito en una aplicación

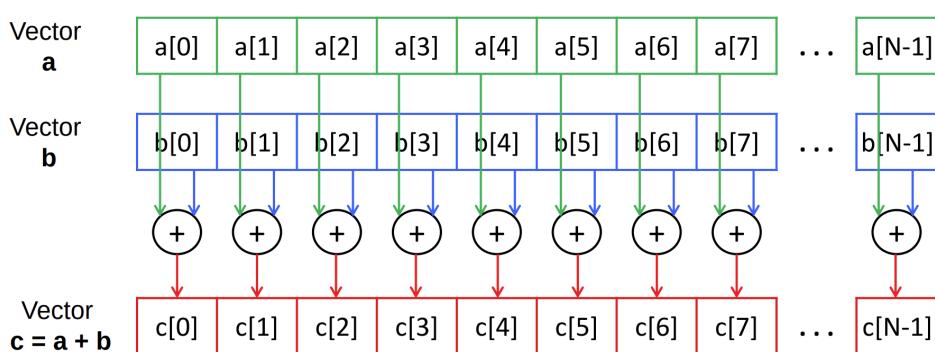
Paralelismo de tareas (task parallelism o TLP -Task Level Par)(paralelismo a nivel de función): Se encuentra extrayendo la estructura lógica de funciones de una aplicación. Está relacionado con el paralelismo a **nivel de función**.

Paralelismo de datos (data parallelism o DLP-Data Level Par) (paralelismo a nivel de bucles): Se encuentra implícito en las operaciones con estructuras de datos (vectores y matrices). Se puede extraer de la representación matemática de la aplicación. Está relacionado principalmente con el paralelismo a **nivel de bucle**.

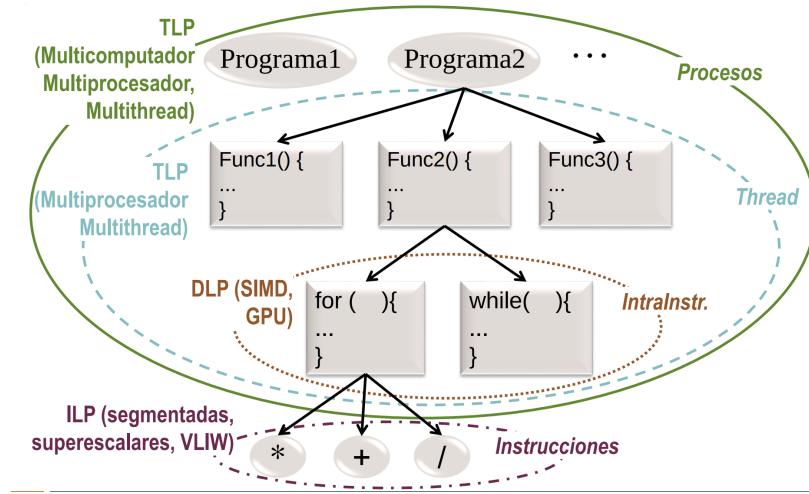
Estructura de funciones lógica de una aplicación. Ej.: decodificador JPEG



Paralelismo de datos. Ej: suma de dos vectores



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



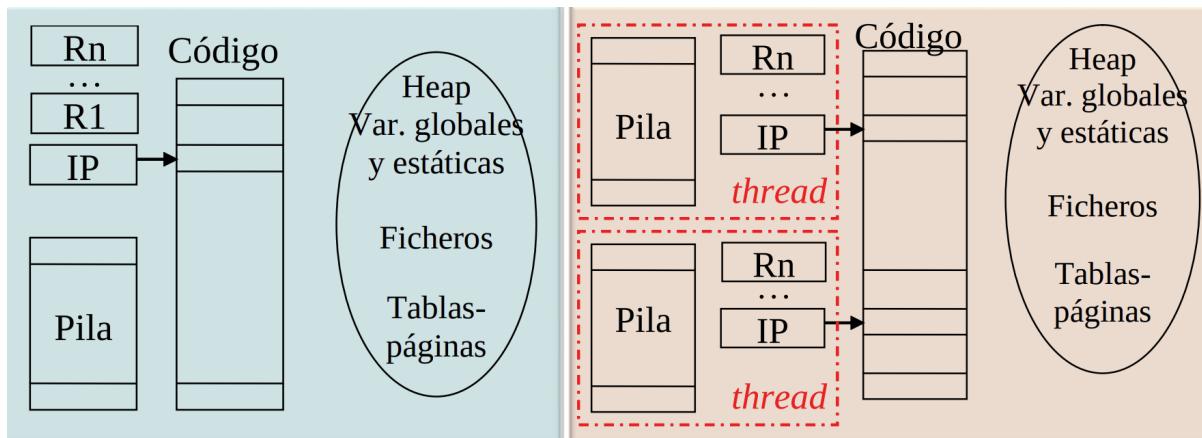
Nivel de paralelismo explícito. Unidades en ejecución en un computador

- **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. Es la menor secuencia de instrucciones que se puede ejecutar en paralelo o concurrentemente.
- **Procesos o process (concepto del SO):** Mayor unidad de ejecución que gestiona el SO. Un proceso consta de uno o varios thread.

Threads versus procesos I

- **Proceso:** Comprende el código de un 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 proceso 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.

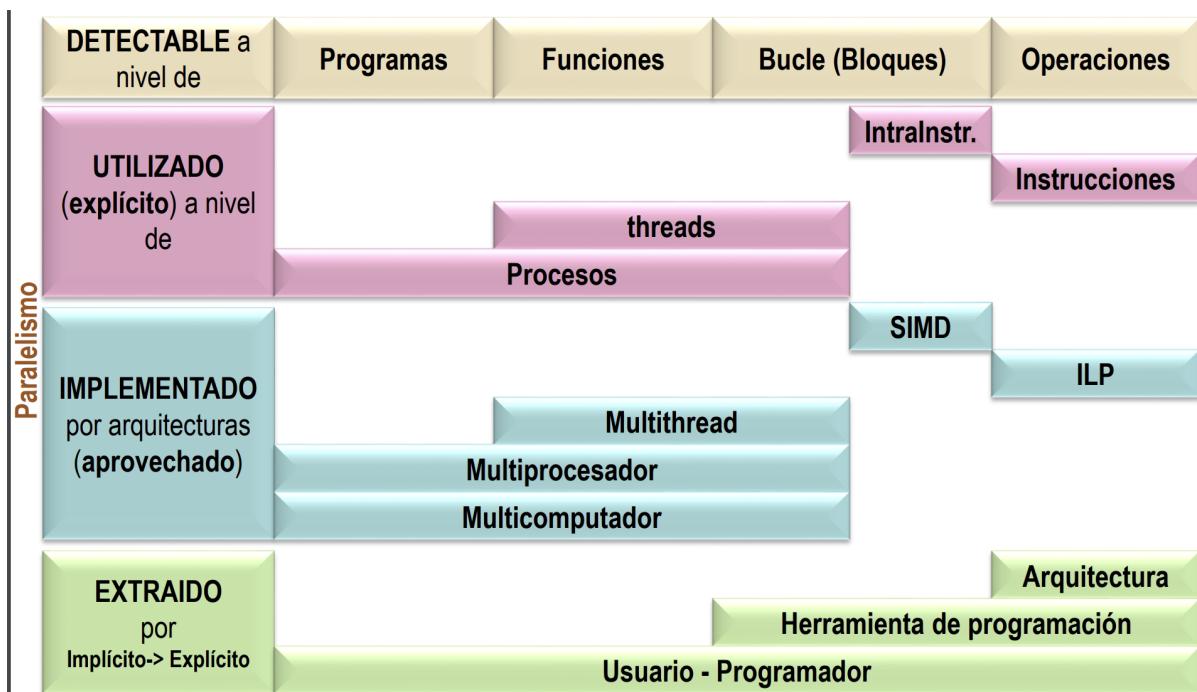
- Para comunicar threads de un proceso se usa la memoria que comparten.



Características ventajosas de las threads:

- Destrucción de threads en menor tiempo.
- Conmutación en menor tiempo.
- Creación de threads en menor tiempo.
- Comunicación en menor tiempo.
- Menor granulidad para threads.

Detección, utilización, implementación y extracción del paralelismo



LECCIÓN 2: Clasificación de arquitecturas paralelas

Computación paralela y computación distribuida

Computación paralela: 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, multicomputadores, clúster).

Computación distribuida: Estudia los aspectos hardware y software relacionados con el desarrollo y ejecución de aplicaciones en un **sistema distribuido**; es decir, **una colección de recursos autónomos** (PC, servidores de datos, software, ..., supercomputadores...) situados en **distintas localizaciones físicas**.

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 **infraestructuras de red local**.

Computación grid: Estudia los aspectos relacionados con el desarrollo y ejecución de aplicaciones de recursos autónomos de **múltiples dominios administrativos geográficamente distribuidos** conectados con **infraestructura de telecomunicaciones**.

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 (**multitenant**)
- Están conectados con métodos **estándar independientes de la plataforma de acceso**.

Clasificaciones de arquitecturas y sistemas paralelos

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 de paralelismo aprovechado (propuesta de clasif.)

Segmento del mercado



“De menor precio y mayor volumen de venta, a mayor precio y menor volumen de venta”

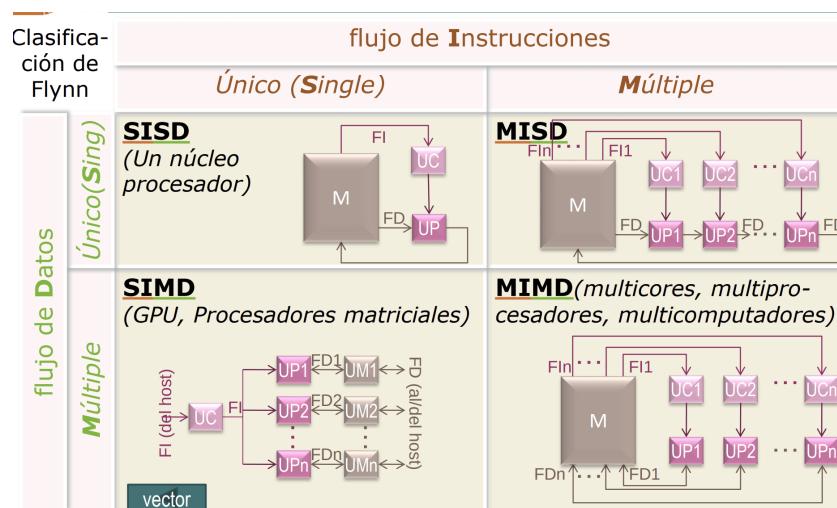
Clasificación de Computadores según segmento

- **Externo (desktop, laptop, server, cluster,...):** 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):

- ❖ Aplicaciones de propósito específico: videojuegos, teléfonos, coches, electrodomésticos, ...
- ❖ Restricciones típicas: consumo de potencia, precio, tamaños reducidos, tiempo real.

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



SISD (un núcleo procesador): Único flujo de instrucciones.

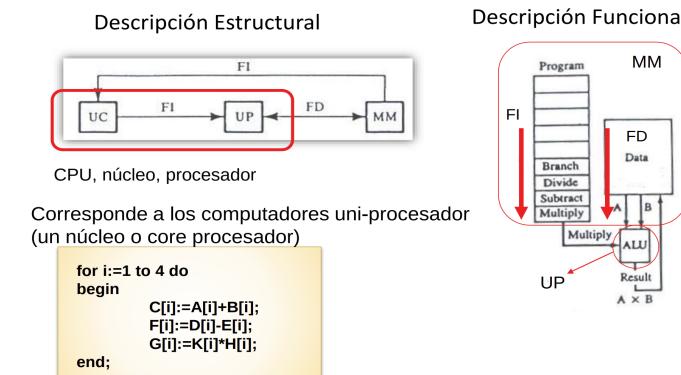
SIMD(GPU, Procesadores matriciales): Único flujo de datos unidad de control.

Sirven para ejecutar instrucciones vectoriales. La GPU, nos va a servir también para aprovechar paralelismo de datos.

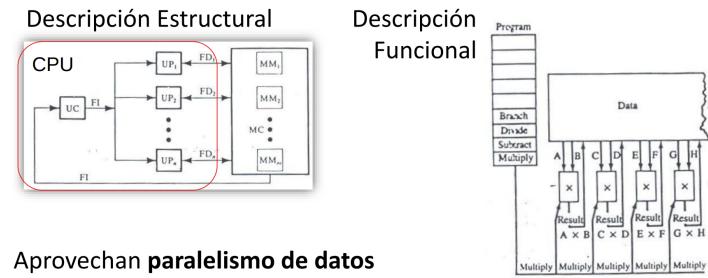
MISD (p.ej: Decodificador JPEG): Múltiples flujos de instrucciones → múltiples unidades de control. La salida de una se toma como entrada de otra unidad de procesamiento.

MIMD: Múltiples flujos de instrucciones → múltiples unidades de control. Como se replican núcleos, han el mismo número de flujo de instrucciones que de datos.

Arquitectura SISD



Arquitectura SIMD



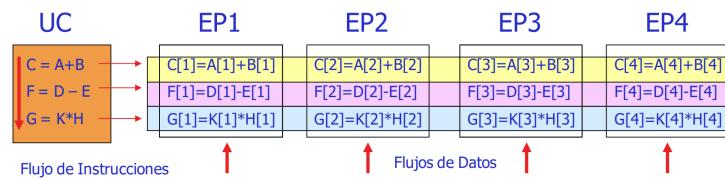
Aprovechan **paralelismo de datos**



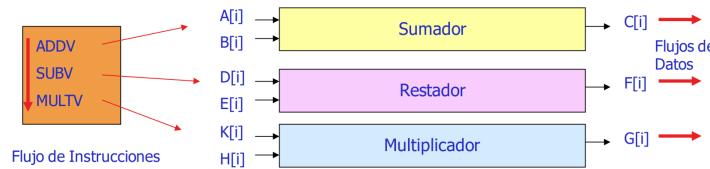
v → Indica que la instrucción es vectorial

EJEMPLO:

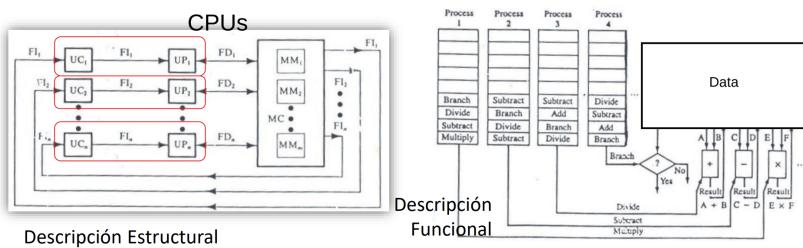
Procesador Matricial



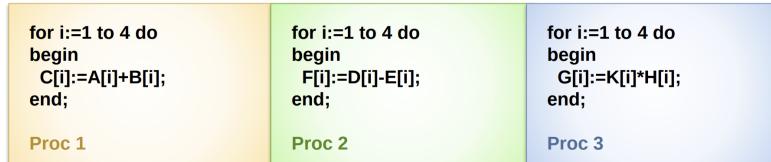
Procesador Vectorial



Arquitecturas MIMD

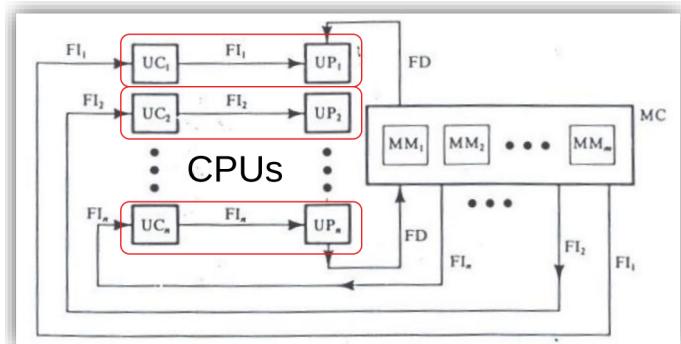


Corresponde con Multinúcleos, Multiprocesadores y Multicomputadores: Puede aprovechar, además, **paralelismo funcional**

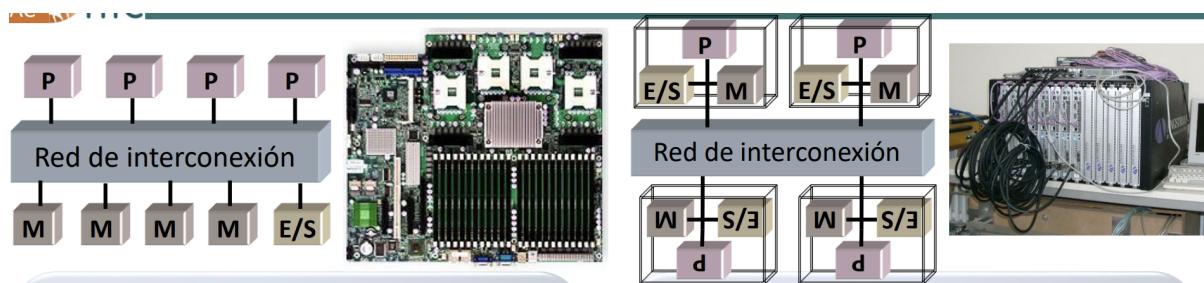


Arquitecturas MISD

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.



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. (cada procesador solo puede acceder a su espacio de direcciones). El programador necesita conocer dónde están almacenados los datos.

Comparativa SMP (Symmetric MultiProcessor) y multicomputadores I

Multiprocesador con memoria centralizada (SMP): Tiene una mayor latencia y es poco escalable. La comunicación implícita se lleva a cabo mediante variables compartidas. Los datos no duplicados están en memoria principal.

Necesita implementar primitivas de sincronización.

La distribución de código y datos entre procesadores no es necesaria.

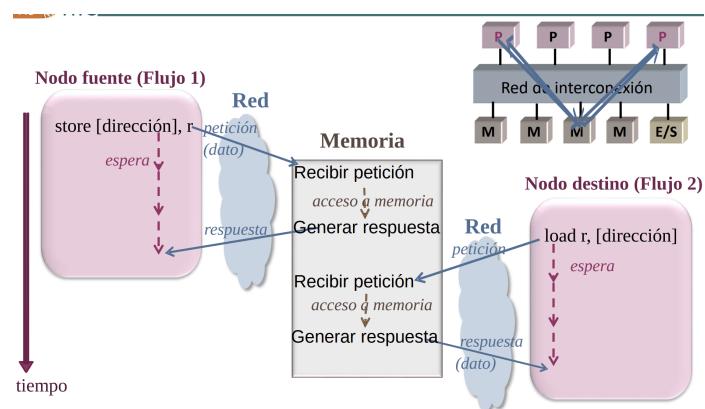
La programación, generalmente es más sencilla.

Multicomputador: Tiene una menor latencia y es más escalable. La comunicación explícita se lleva a cabo mediante software para paso de mensajes (send / receive). Los datos duplicados residen en memoria principal, copia de datos.

La sincronización se lleva a cabo mediante software de comunicación.

La distribución de código y datos entre procesadores es necesaria → herramientas de programa más sofisticadas. La programación generalmente es más difícil.

Comunicación uno-a-uno en un multiprocesador



Comunicación uno-a-uno en un multiprocesador

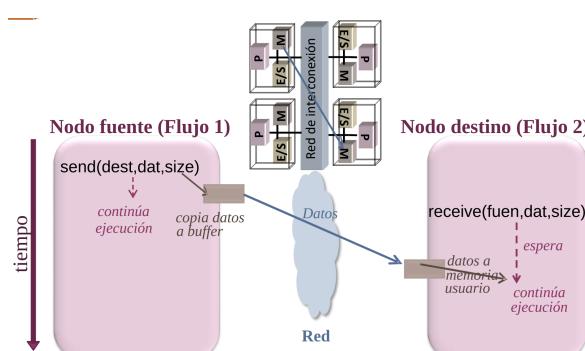
Secuencial	Paralelo		
...	<u>F1</u>	<u>F2</u>	
<u>A=valor;</u>	...	<u>A=valor;</u>	<u>F1</u> es el flujo de instrucciones <u>productor</u> del dato (<u>envía</u> el dato)
<u>copia=A;</u>	...	<u>copia=A;</u>	<u>F2</u> es el flujo de instr. <u>consumidor</u> del dato (<u>recibe</u> el dato)
...		...	

➤ 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)	
<u>F1</u>	<u>F2</u>
...	...
<u>A=valor;</u>	<u>while (K==0) {;</u>
<u>K=1;</u>	<u>copia=A;</u>
...	...

NOTA: La copia es trivial porque la memoria es compartida

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



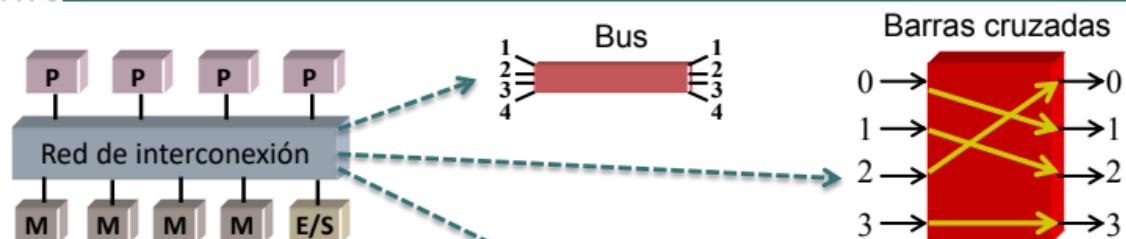
Comunicación uno-a-uno en multicomputador

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

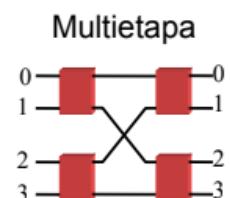
F1 es el flujo de instrucciones productor del dato (envía el dato)
F2 es el flujo de instr. consumidor del dato (recibe el dato)

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

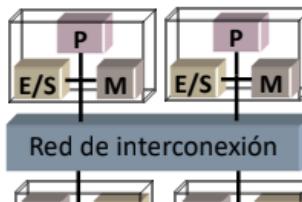
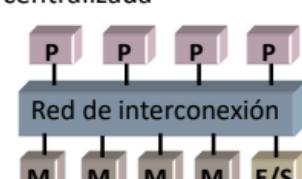
Incremento de escalabilidad en multiprocesadores y red de interconexión



- 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

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

UMA (multiprocesadores con acceso a memoria uniforme):

El tiempo de acceso de los procesadores a una determinada posición de MP es igual sea cuál sea el procesador. El acceso a una posición de memoria en caché es igual para todos los procesadores.

NUMA (multiprocesadores con acceso a memoria no uniforme):

El tiempo de acceso de los procesadores a una determinada posición de MP depende del procesador, ya que un procesador tardará menos tiempo en acceder al bloque de memoria local . Para que un NUMA sea realmente escalable, se debe disminuir la latencia media reduciendo el número de accesos de un procesador a posiciones situadas en bloques de memoria localizados en nodos remotos.

NCC-NUMA (non-cache-coherent non-uniform memory access):

o arquitecturas con acceso a memoria no uniforme sin coherencia de caché entre nodos, o simplemente NUMA. No incorporan hardware para evitar problemas por incoherencia entre caché de distintos nodos. Los datos modificables compartidos no se pueden trasladar a caché de nodos remotos, hay que acceder a ellos individualmente a través de la red. Se puede hacer más tolerable la latencia utilizando precaptación (prefetching) de memoria y procesamiento multihebra.

CC-NUMA(cache-coherent non-uniform memory access):

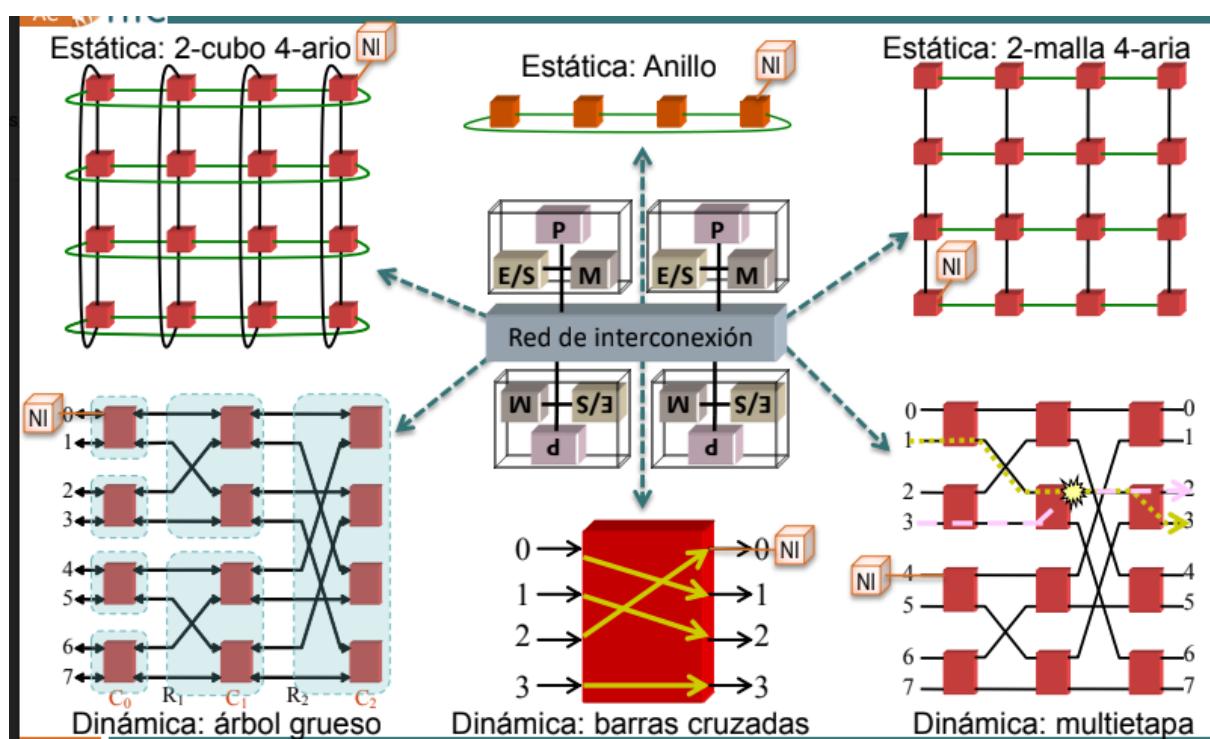
o arquitecturas con acceso a memoria no uniforme con coherencia de caché. Tienen hardware para mantener coherencia entre caché de distintos nodos, que se encarga de las transferencias de datos compartidos entre nodos. Supone un coste

añadido e introduce un retardo que hace que estos sistemas escalen en menor grado que un NUMA.

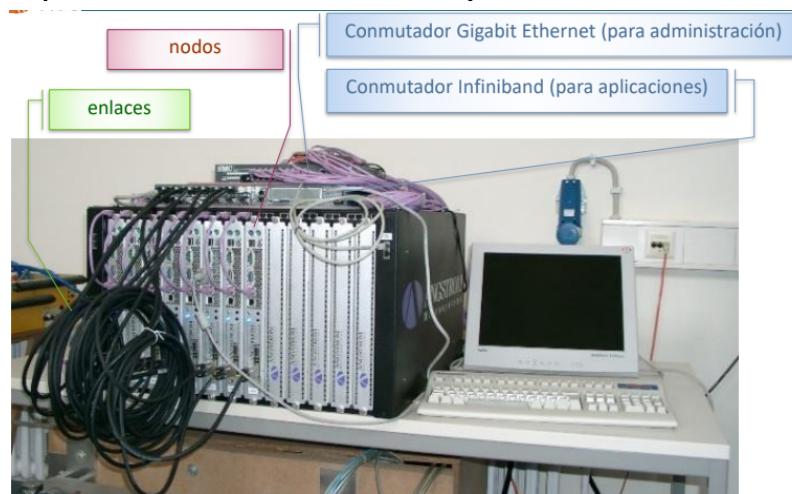
COMA (caché only memory access):

La memoria local de los procesadores se gestiona como caché. El sistema de mantenimiento de coherencia se encarga de llevar dinámicamente (en tiempo de ejecución) el código y los datos a los nodos donde se necesiten. Permite replicación y migración de bloques en memoria en función de su frecuencia de uso de los nodos. El coste de estos sistemas y el retardo que añade el hardware es mayor que en CC-NUMA. No existe actualmente ningún sistema comercial COMA.

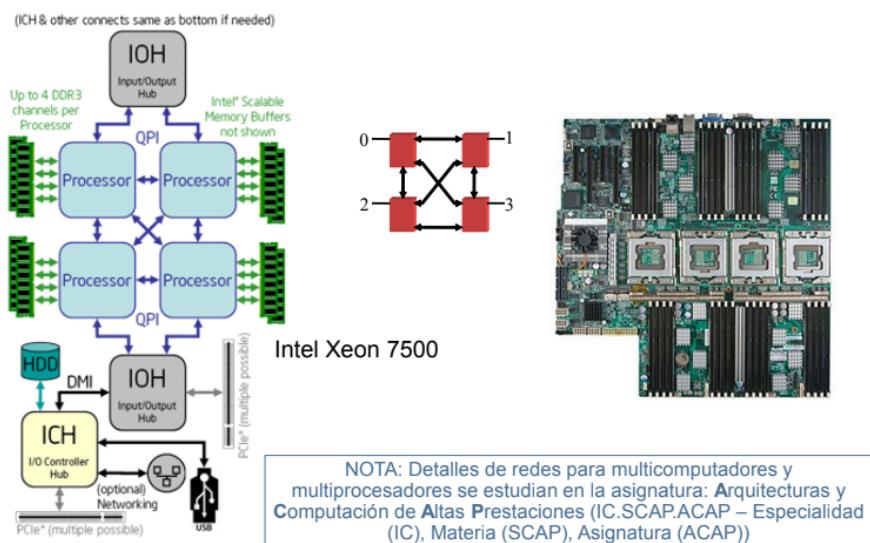
Red en sistemas con memoria físicamente distribuida (NI: Network Interface)



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



Ejemplo: Placa CC-NUMA con red estática



Propuesta clasificación computadores con múltiples flujos de instrucciones o threads

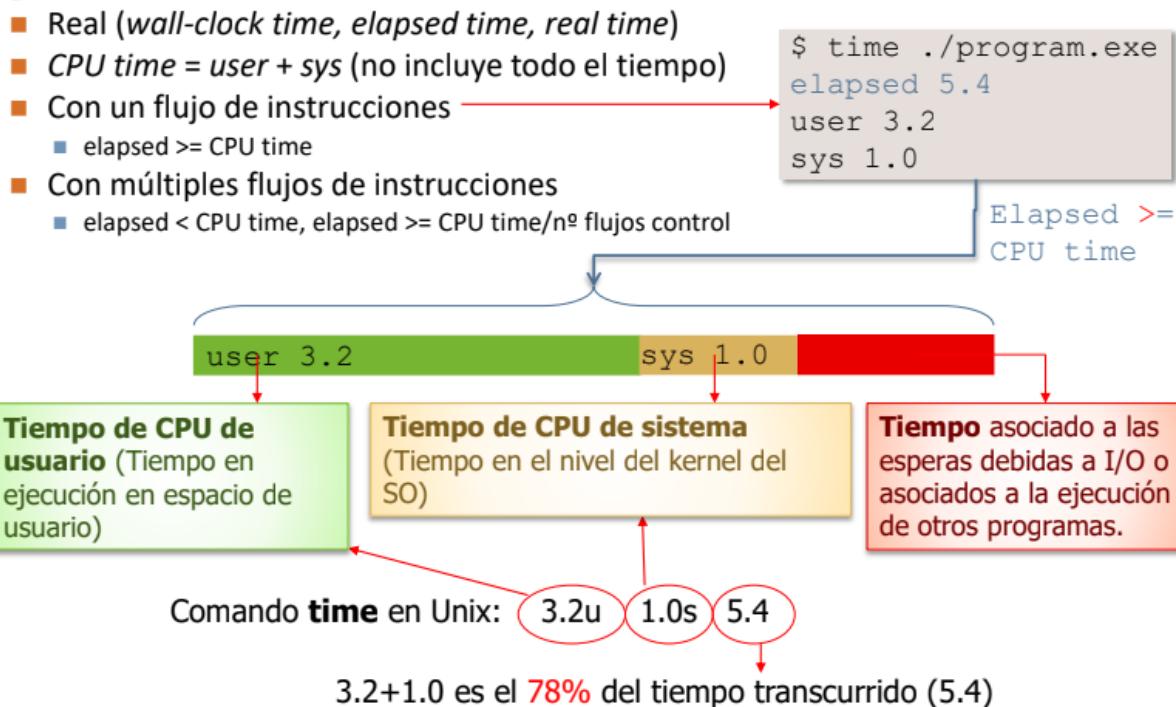


Arquitecturas con DLP, ILP y TLP (thread=flujo de control o de instrucciones)

Arq. con DLP (Data Level Parallelism)	Arq. con ILP (Instruction Level Parallelism)	Arq. con TLP (Thread Level Parallelism) explícito y una instancia de SO	Arq. con TLP explícito y múltiples instancias SO
Ejecutan las operaciones de una instrucción concurr. o en paralelo	Ejecutan múltiples instrucciones concurr. o en paralelo	Ejecutan múltiples flujos de instrucciones concurr. o en paralelo	Ejec. múltiples flujos de instr. en paralelo
Unidades funcionales vectoriales o SIMD	Cores escalares segmentados, superescalares o VLIW/EPIC	Cores que modifican la arquit. escalar segmentada, superescalares o VLIW/EPIC para ejecutar threads concurr. o en paralelo	Multi-procesadores: ejecutan threads en paralelo en un computador con múltiples cores (incluye multicores)

LECCIÓN 3: Evaluación de prestaciones de una arquitectura

Tiempo de respuesta de un programa en una arquitectura

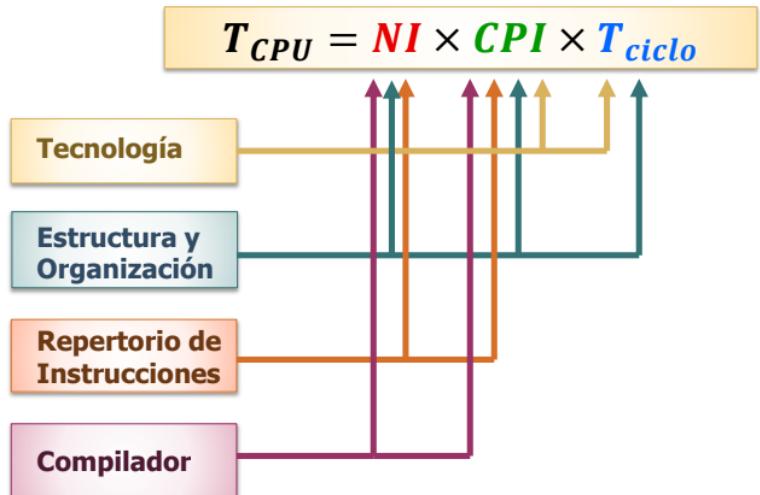
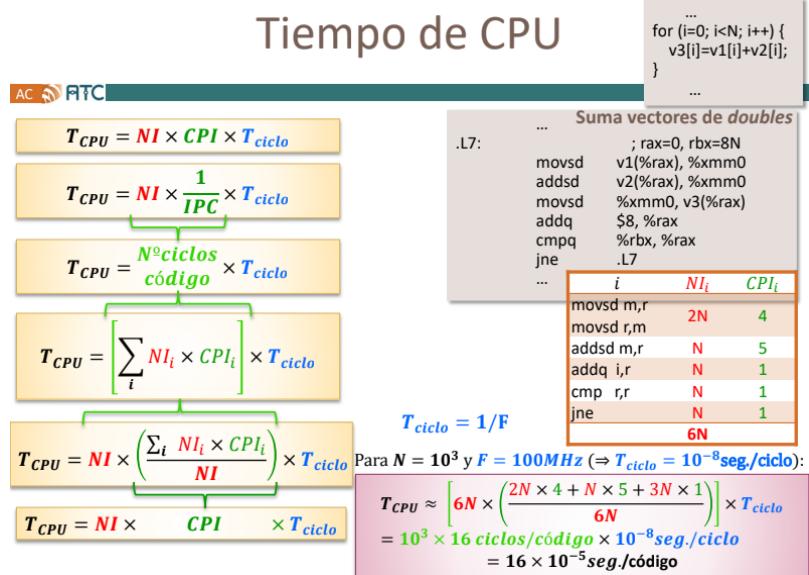


Algunas alternativas para obtener tiempos

Función	Fuente	Tipo	Resolución aprox. (microsegundos)
time	SO (/usr/bin/time)	elapsed, user, system	10000
clock()/_CLOCKS_PER_SEC	SO (time.h)	CPU	10000
gettimeofday()	SO (sys/time.h)	elapsed	1
clock_gettime()/_clock_getres()	SO (time.h)	elapsed	0.001
omp_get_wtime()/_omp_get_wtick()	OpenMP (omp.h)	elapsed	0.001
SYSTEM_CLOCK()	Fortran	elapsed	1

La resolución depende de la plataforma

Tiempo de CPU



MIPS:

➤ 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 con el programa (no sirve para caracterizar la máquina)
- Puede variar inversamente con las prestaciones (mayor valor de MIPS corresponde a peores prestaciones)

MFLOPS:

- Millones de operaciones punto flotante por segundo (MFLOPS):

$$MFLOPS = \frac{n^o FP}{T_{CPU} \times 10^6}$$

- No es una medida adecuada para todos los programas (solo viene en cuenta las operaciones en coma flotante)
- El conjunto de operaciones en coma flotante no es constante en máquinas diferentes y la potencia de las operaciones en coma flotante no es igual para todas las operaciones

MIPS y FLOPS

AC RTC
-O2

```
;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=2²⁶)=0.182 seg.

$$GIPS = \frac{NI}{T_{CPU} \times 10^9} = \frac{N \times 7}{0.182 \times 10^9}$$

$$= \frac{2^{26} \times 7}{0.182 \times 10^9} \approx 2.58 GIPS$$

$$GFLOPS = \frac{n^o FP}{T_{CPU} \times 10^9} = \frac{N \times 2}{0.182 \times 10^9}$$

$$= \frac{2^{26} \times 2}{0.182 \times 10^9} \approx 0.737 GFLOPS$$

-O3

```
;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=2²⁶)=0.178 seg.

$$GIPS = \frac{NI}{T_{CPU} \times 10^9} = \frac{(N/2) \times 9}{0.178 \times 10^9}$$

$$= \frac{2^{25} \times 9}{0.178 \times 10^9} \approx 1.7 GIPS$$

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

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

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

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

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

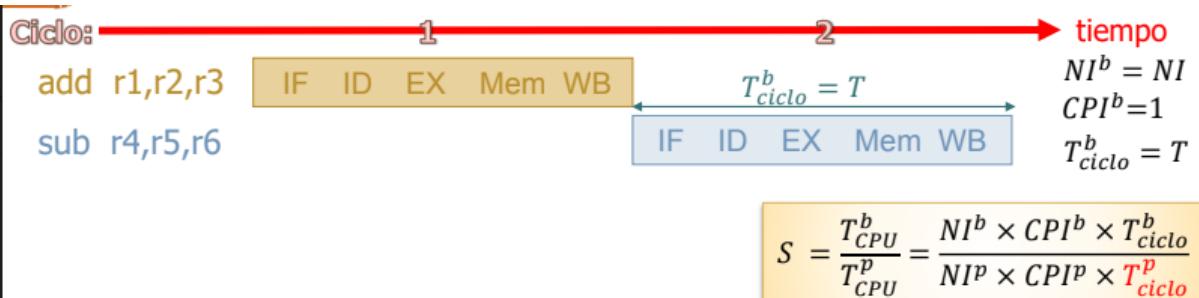
V_b Velocidad de la máquina base

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

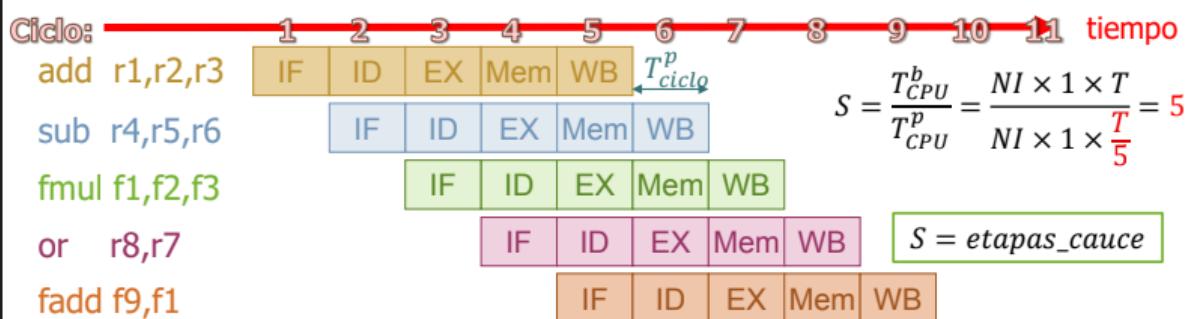
T_b Tiempo de ejecución en la máquina base

T_p Tiempo de ejecución en la máquina mejorada

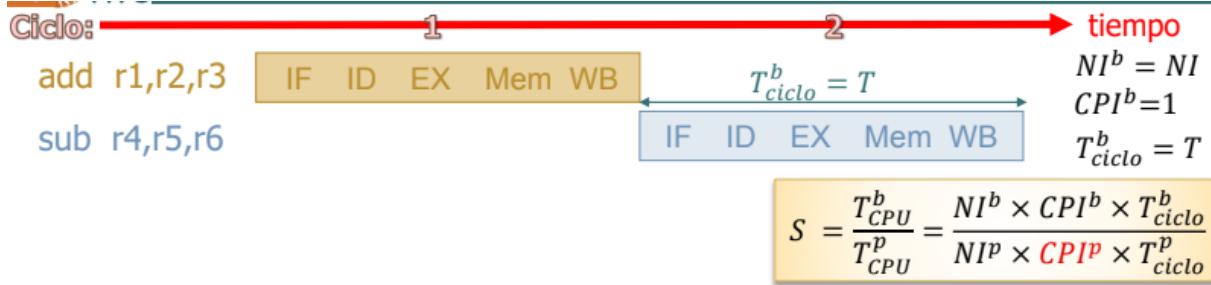
Mejora en un núcleo de procesamiento: segmentación



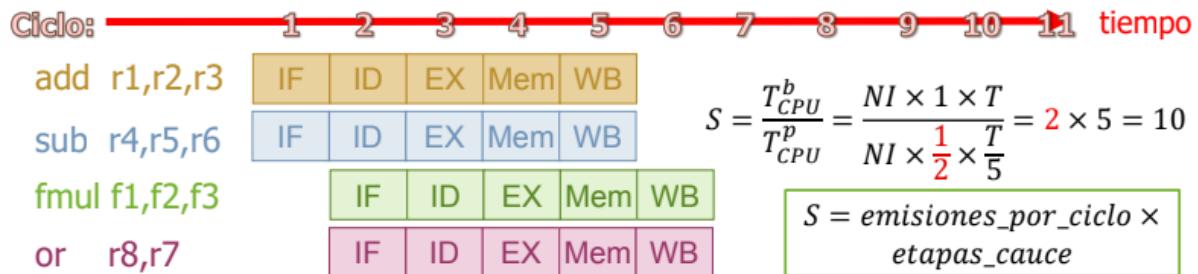
Núcleo segmentado en 5 etapas:



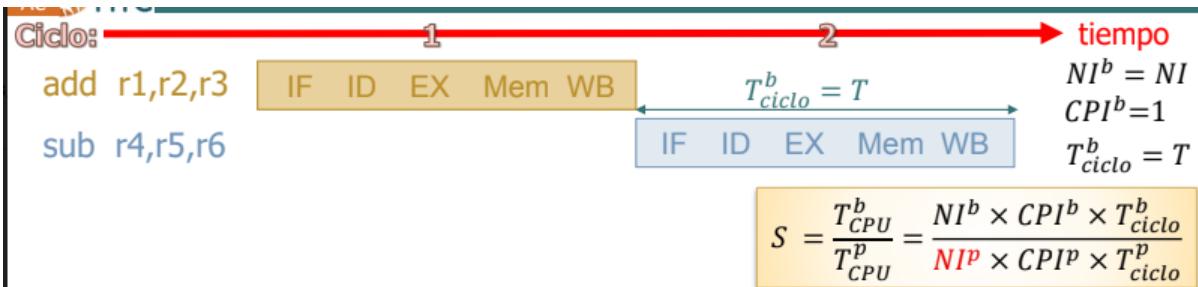
Mejora en un núcleo de procesamiento: operación superscalar



Núcleo **superescalar** con 2 emisiones por ciclo y 5 etapas:

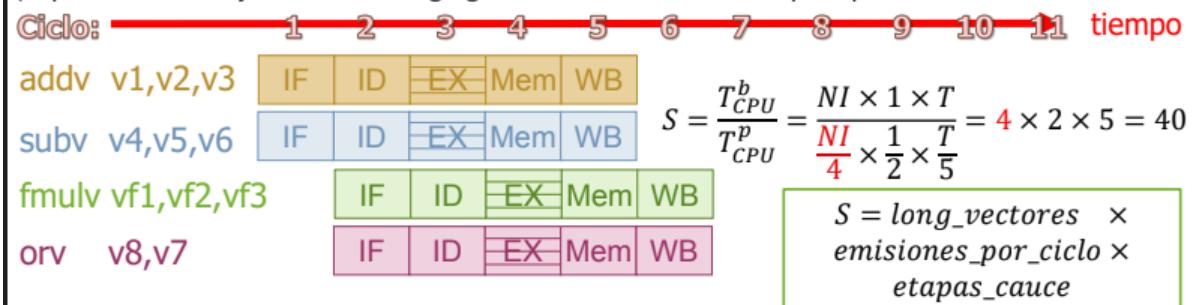


Mejora en un núcleo de procesamiento: unidades funcionales SIMD

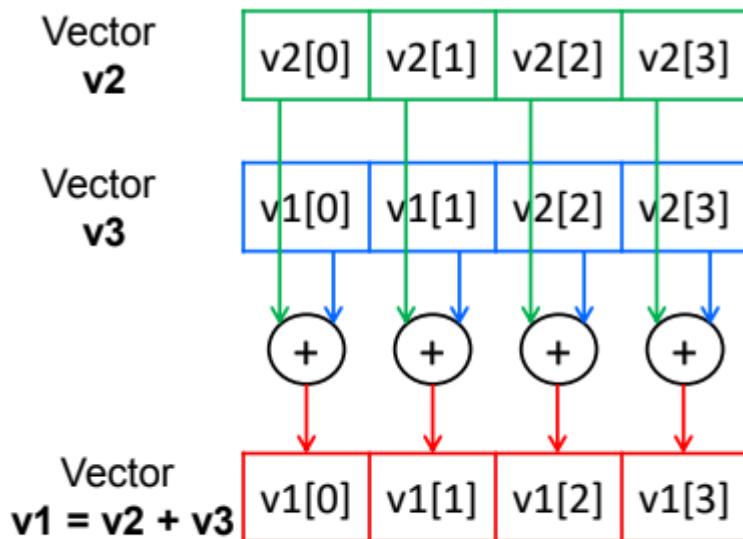


Núcleo **superescalar** con 2 emisiones por ciclo y 5 etapas, y **unidades funcionales SIMD** (vectoriales) que procesan **vectores de 4 componentes**

(suponemos el mejor caso: el código genera sólo instrucciones que operan con vectores de 4 componentes)



Paralelismo de datos. Ej: suma de dos vectores



addv v1,v2,v3

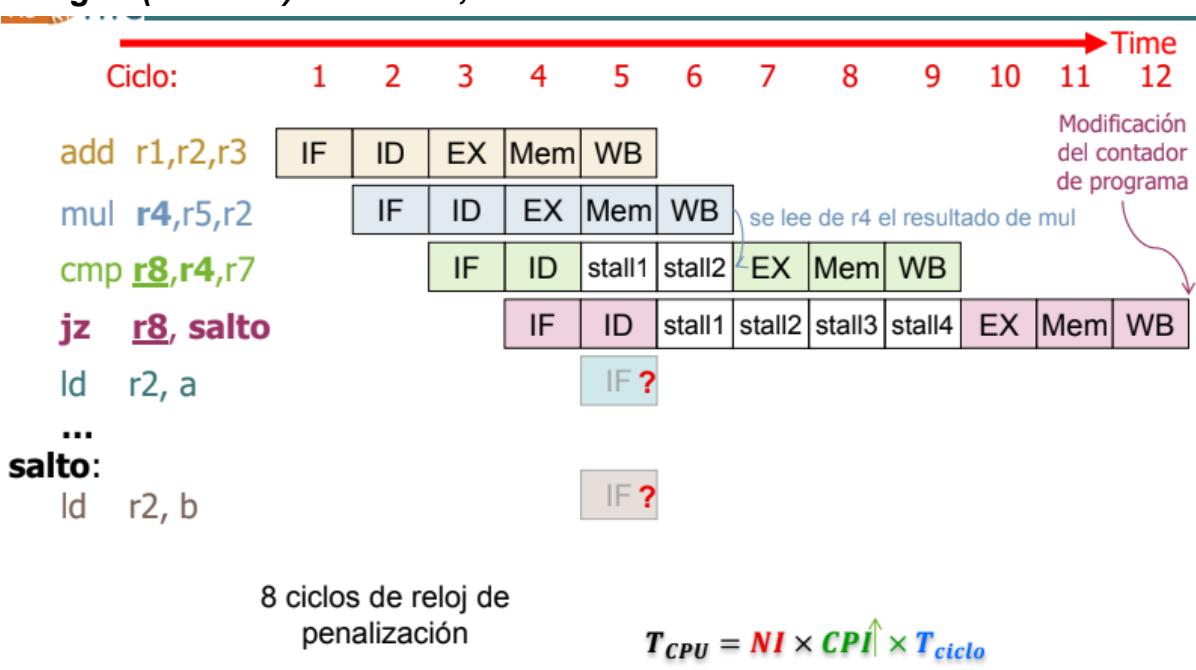
¿Qué impide que se pueda obtener la ganancia en velocidad pico?

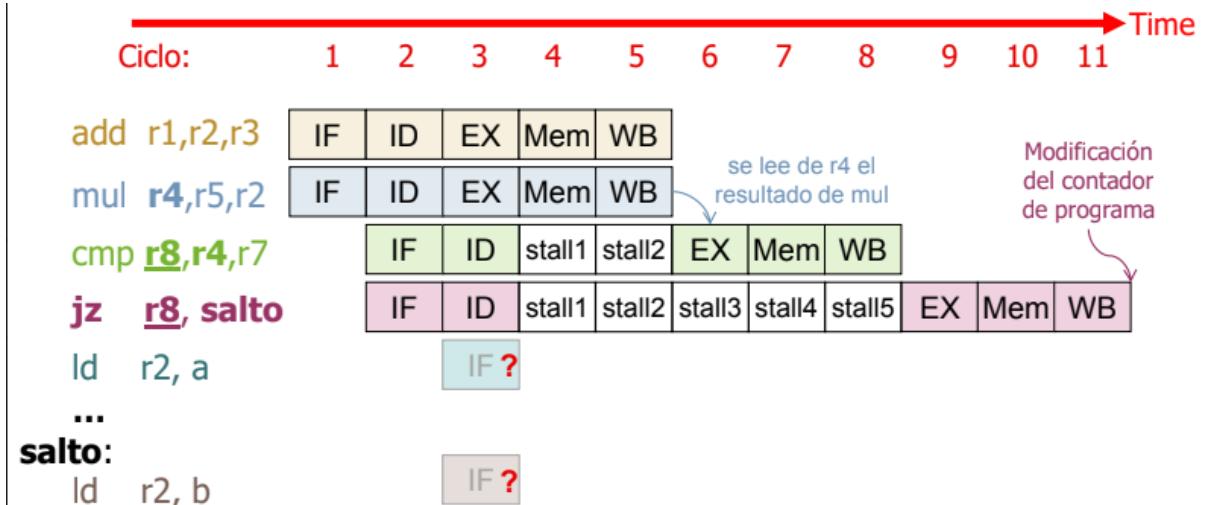
Riesgos :

- datos
- control
- estructurales

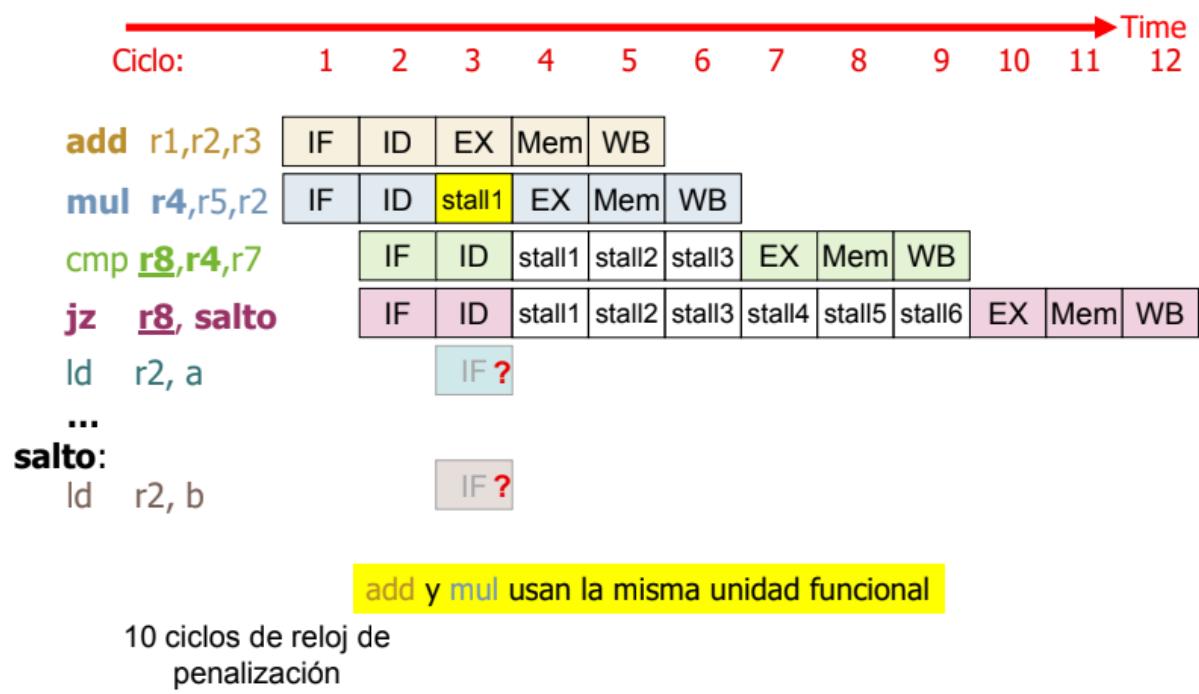
Acceso a memoria (debido a la jerarquía)

Riesgos (hazards): de datos, de control





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



Ley de Amdahl

$$S = \frac{\text{Tiempo de ejecución original del sistema}}{T_p}$$

$$\Rightarrow S = \frac{T_p}{f T_b + (1-f) T_b}$$

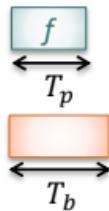
$$\Rightarrow \lim_{p \rightarrow \infty} \frac{1}{f + (1-f)} = \frac{1}{f}, \text{ Si } f \rightarrow 0 \Rightarrow \lim = p$$

Si no una sola mejoría

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

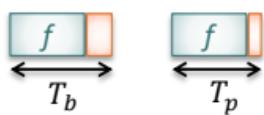
$$S = \frac{V_p}{V_b} = \frac{T_b}{T_p} \leq \frac{1}{f + \frac{1-f}{p}} = \frac{p}{1+f(p-1)}$$

→ $\frac{1}{f}$
→ 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)

Benchmarks

Propiedades exigidas a medidas de prestaciones:

- **Fiabilidad** => Representativas (de la carga de trabajo que vamos a utilizar en la máquina), 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

- ❖ **De bajo nivel o microbenchmark** (sirven para evaluar partes de la arquitectura):
 - test ping-pong, evaluación de las operaciones con enteros o con flotantes
- ❖ **Kernels** (trozos de código que se emplean en aplicaciones reales) :
 - resolución de sistemas de ecuaciones, multiplicación de matrices, FFT, descomposición LU

❖ **Sintéticos** (son los menos fiables. No obtienen un resultado útil):

- Dhystone, Whetstone

❖ **Programas reales**

- Ej. SPEC CPU2017, enteros (gcc, perlbench)

❖ **Aplicaciones diseñadas**

- Predicción de tiempo, dinámica de fluidos, animación etc. (p. ej. SPEC2017).