

2º curso / 2º cuatr.

Grado en
Ing. Informática

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

ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



ATC

Departamento de Arquitectura
y Tecnología de Computadores
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
 - Ganancia en prestaciones al realizar una mejora
 - Conjunto de programas de prueba (*Benchmark*)

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. Ränder. *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)
time	SO (/usr/bin/time)	<i>elapsed, user, system</i>	10000
clock()/CLOCKS_PER_SEC	SO (time.h)	<i>CPU</i>	10000
gettimeofday()	SO (sys/time.h)	<i>elapsed</i>	1
clock_gettime()/clock_getres()	SO (time.h)	<i>elapsed</i>	0.001
omp_get_wtime()/ omp_get_wtick()	OpenMP (omp.h)	<i>elapsed</i>	0.001
SYSTEM_CLOCK()	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];
}
...
```

Suma vectores de *doubles*

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

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

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

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

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

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

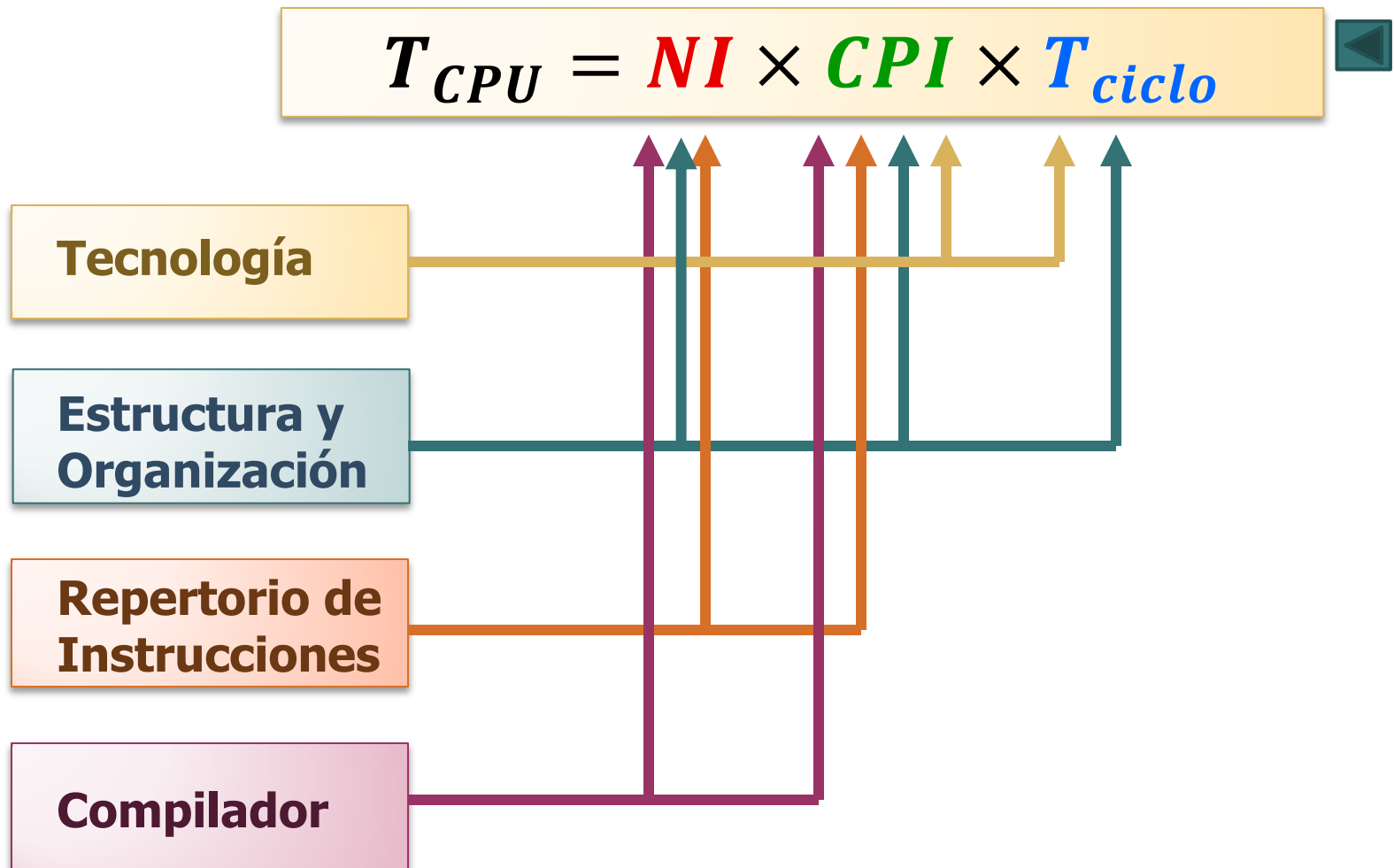
$$T_{CPU} = \frac{\text{Nº ciclos código}}{\text{código}} \times T_{ciclo}$$

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

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

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

Tiempo de CPU



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):

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

MIPS y FLOPS

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

AC ATC

-02

```
;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
cpl %eax, %ebp
jg .L6
```

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

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

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

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

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

-03

```
;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)
cpl %ebp, %eax
jb .L7
```

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

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

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

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

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*

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

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

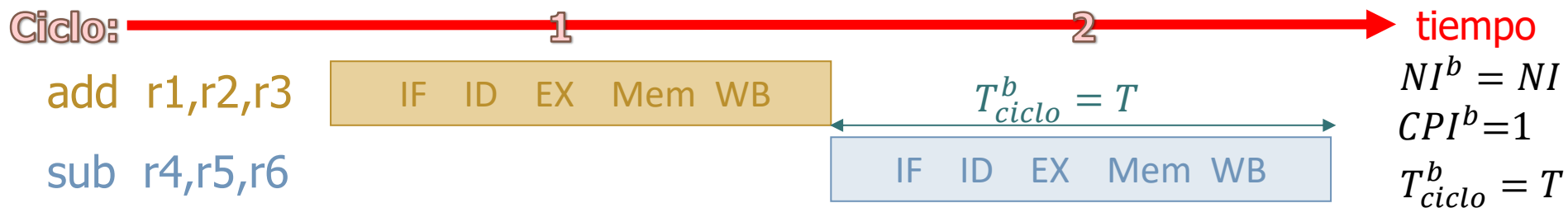
V_b Velocidad de la máquina base

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

T_b Tiempo de ejecución en la máquina base

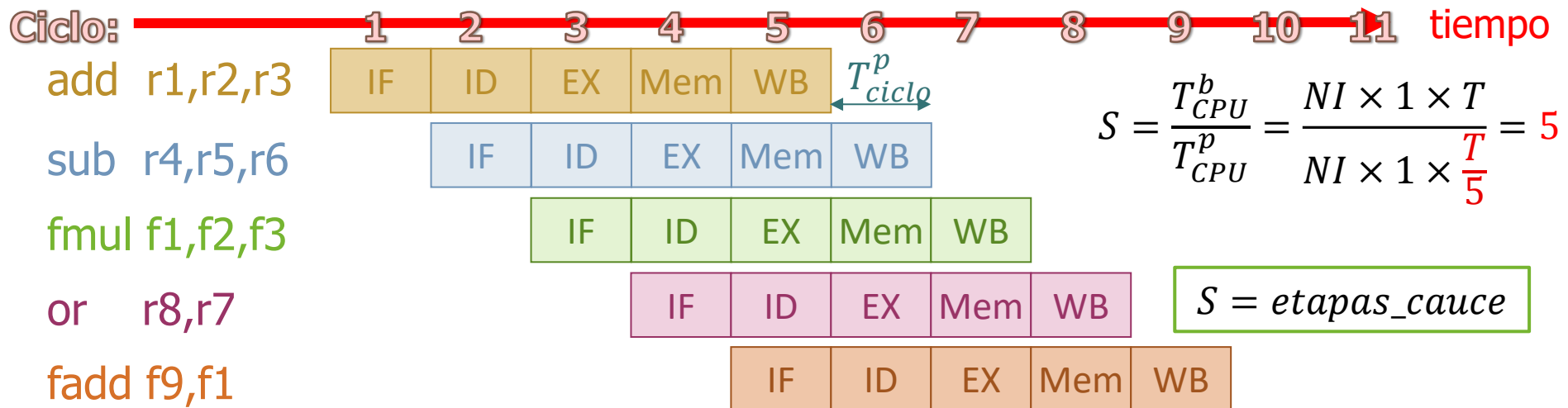
T_p Tiempo de ejecución en la máquina mejorada

Mejora en un núcleo de procesamiento: segmentación

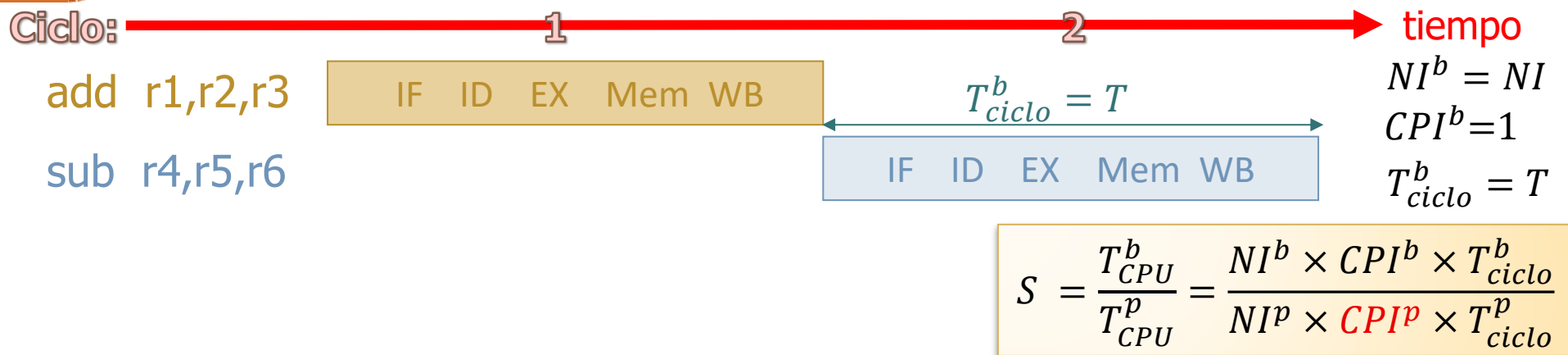


$$S = \frac{T^b_{CPU}}{T^p_{CPU}} = \frac{NI^b \times CPI^b \times T^b_{ciclo}}{NI^p \times CPI^p \times T^p_{ciclo}}$$

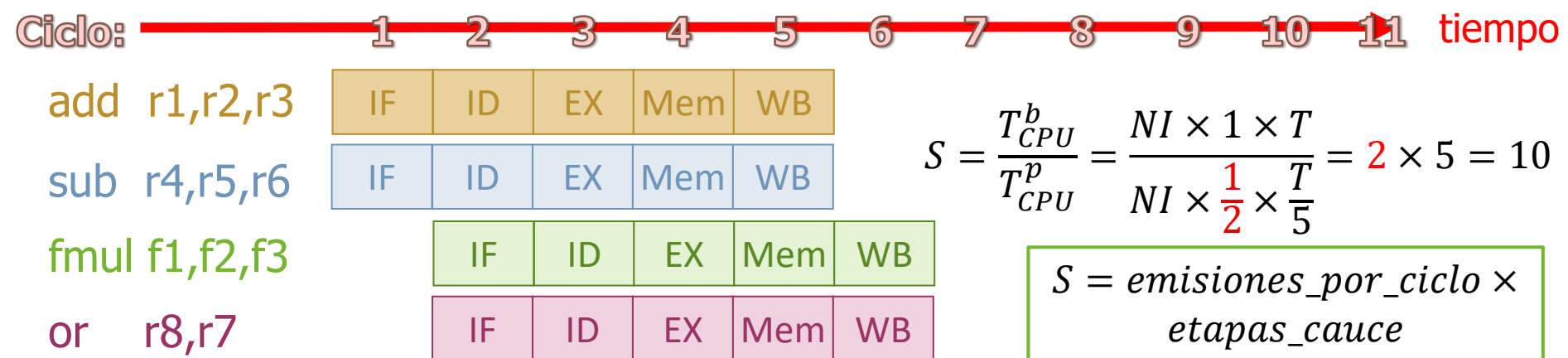
Núcleo segmentado en 5 etapas:



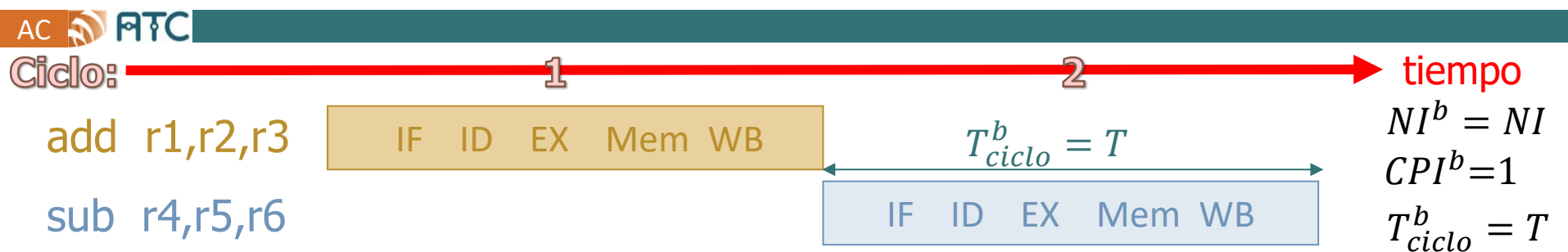
Mejora en un núcleo de procesamiento: operación superscalar



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

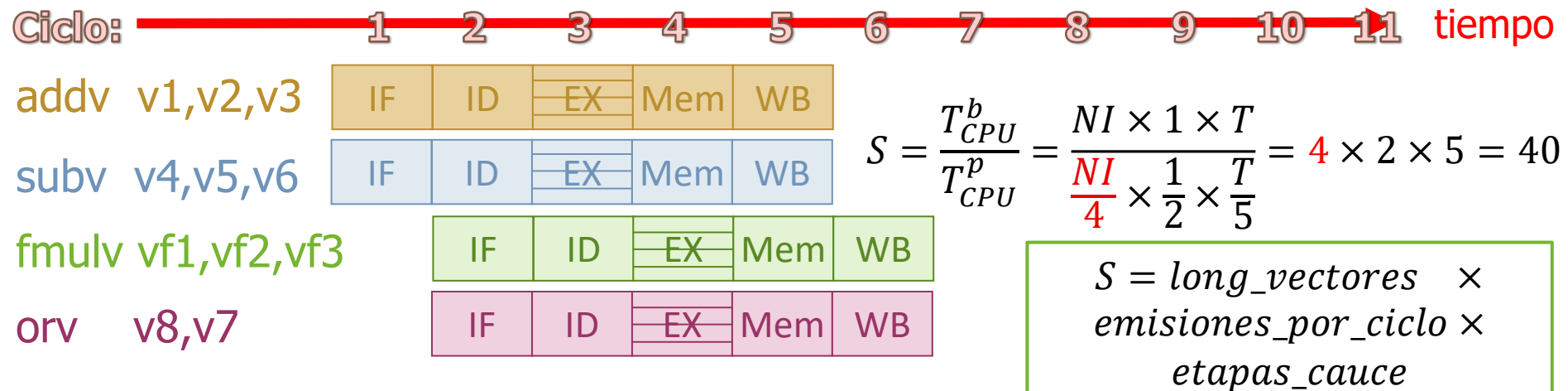


Mejora en un núcleo de procesamiento: unidades funcionales SIMD

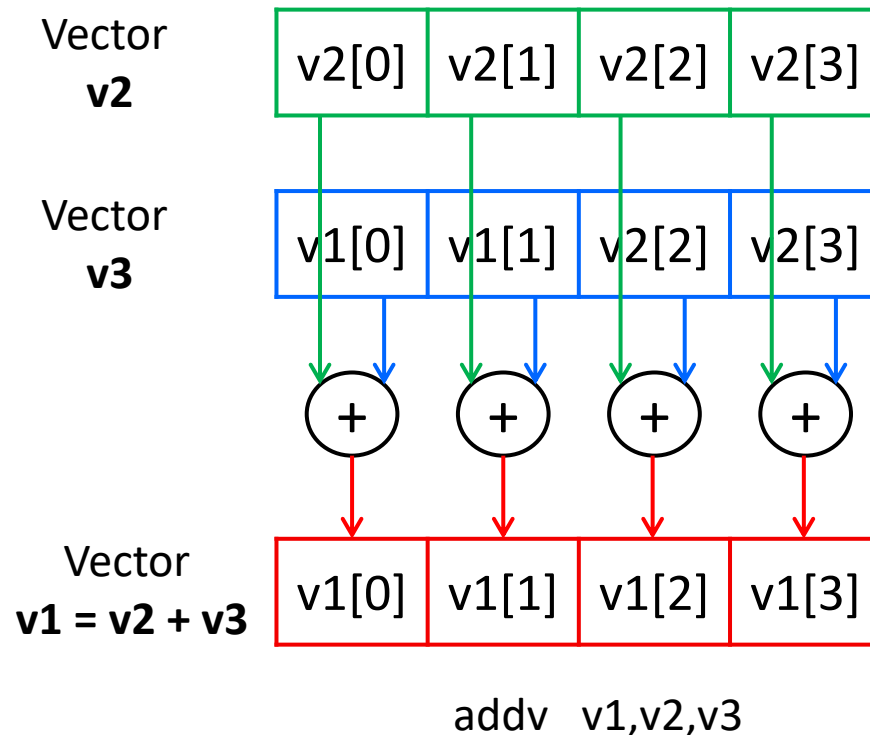


$$S = \frac{T_{CPU}^b}{T_{CPU}^p} = \frac{NI^b \times CPI^b \times T_{ciclo}^b}{\textcolor{red}{NI}^p \times CPI^p \times T_{ciclo}^p}$$

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?

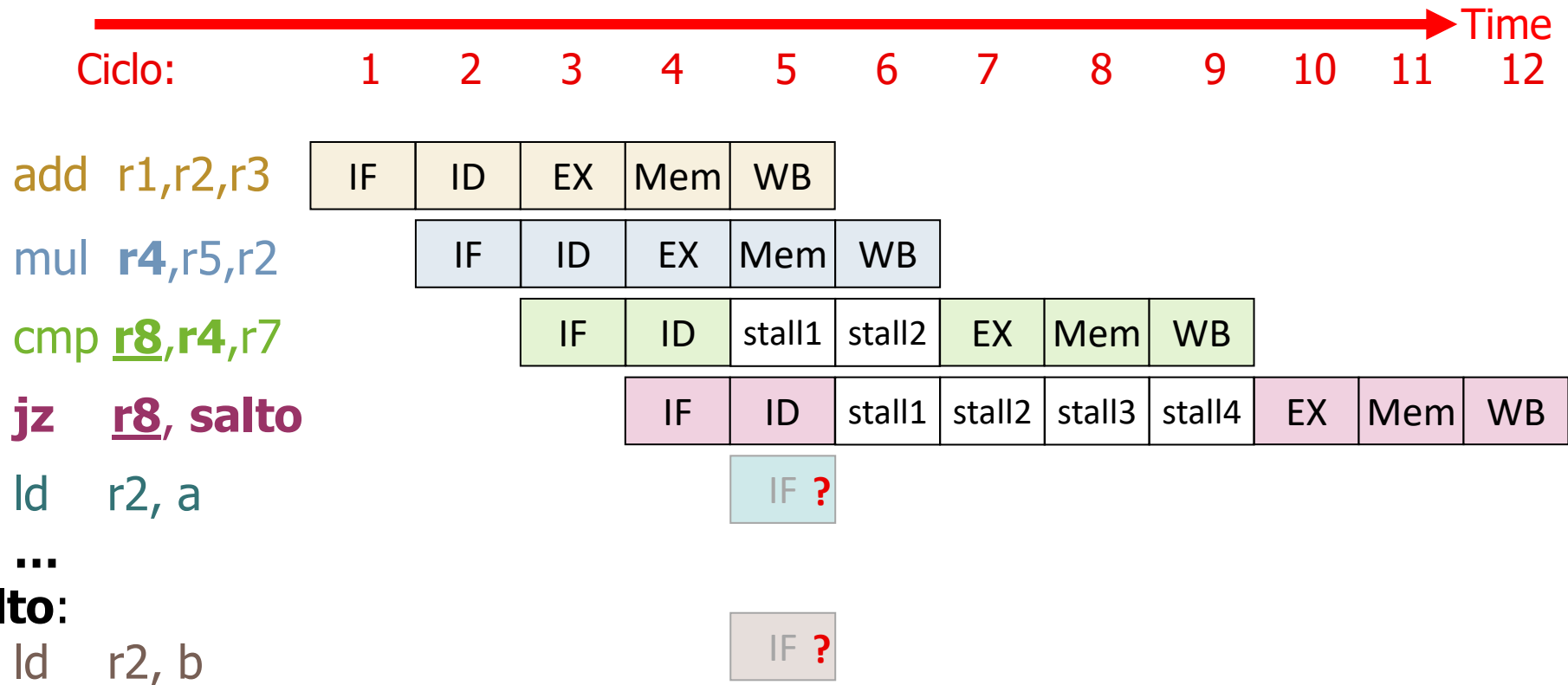
➤ Riesgos:

- Datos
- Control
- Estructurales

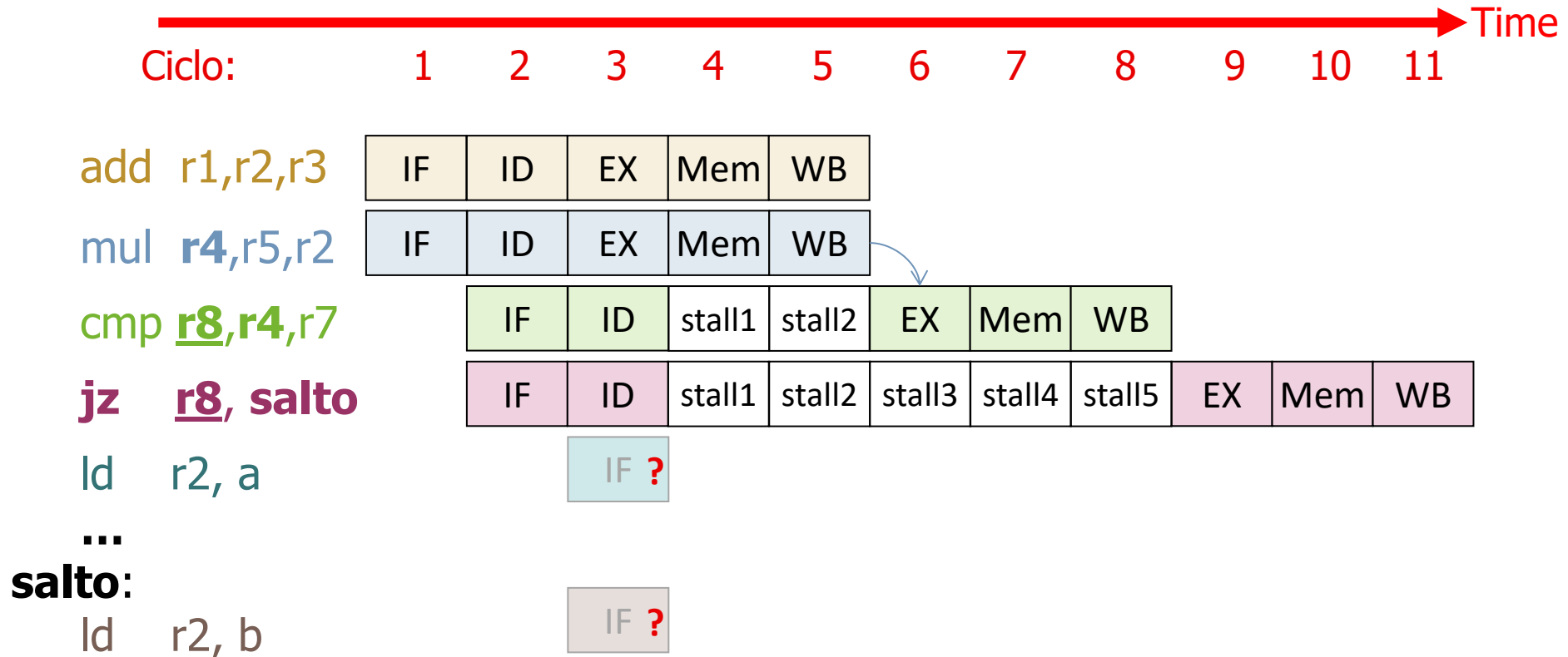
➤ Accesos a memoria (debido a la jerarquía)



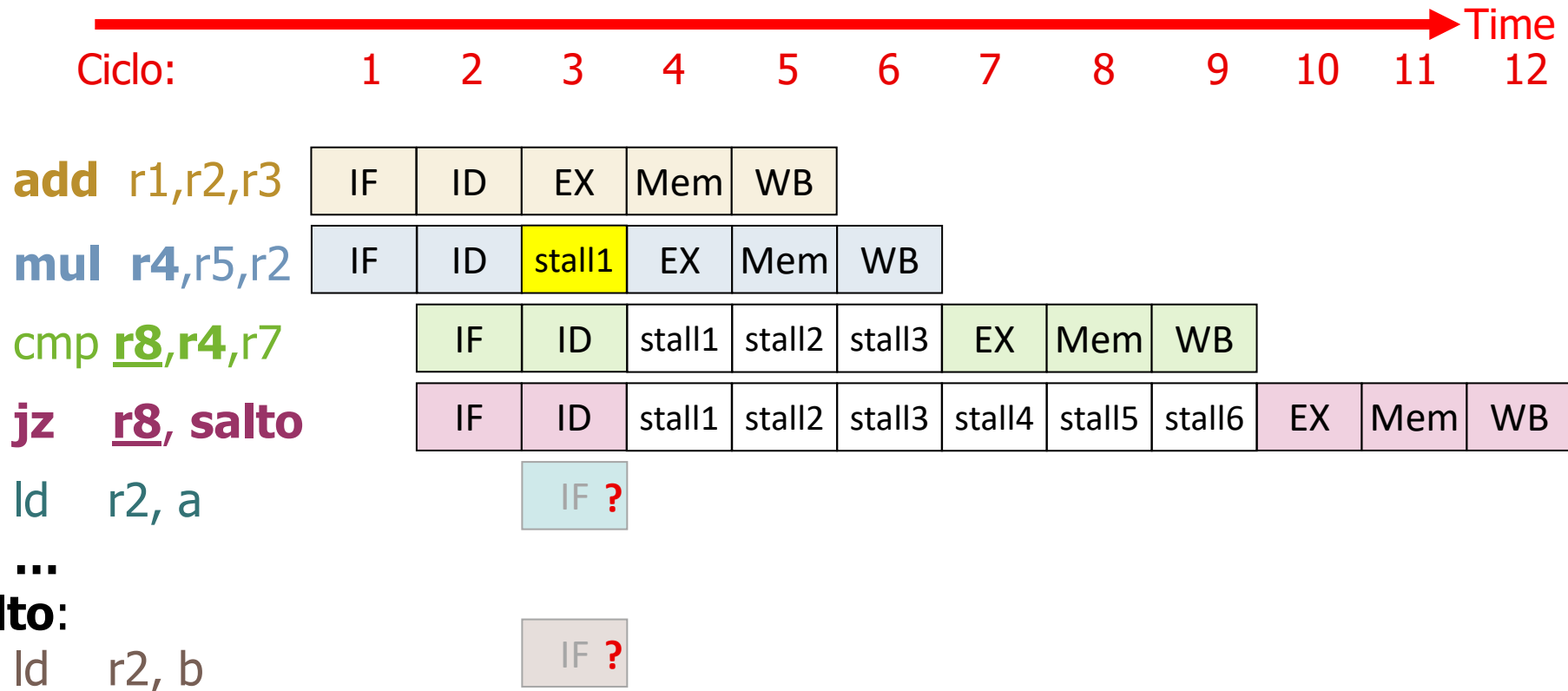
Riesgos (*hazards*): de datos, de control



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



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



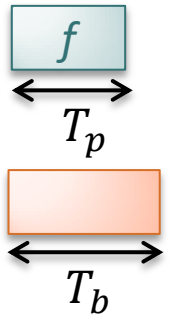
add y **mul** usan la misma unidad funcional

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:

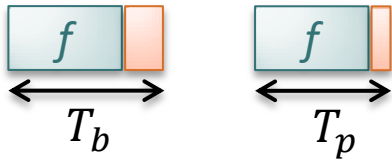
$$S = \frac{V_p}{V_b} = \frac{T_b}{T_p} \leq \frac{1}{f + \frac{1-f}{p}} = \frac{p}{1 + f(p-1)}$$

$\xrightarrow[p \rightarrow \infty]{\text{red}} \frac{1}{f}$
 $\xrightarrow[f \rightarrow 0]{\text{green}} p$



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, perlbnk),

➤ Aplicaciones diseñadas

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