

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

ETSIIT

Escuela Técnica Superior  
de Ingenierías Informática  
y de Telecomunicación



ATC

Departamento de Arquitectura  
y Tecnología de Computadores  
UNIVERSIDAD DE GRANADA



# 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

# 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
    - Tiempo de respuesta
    - Productividad: MIPS, MFLOPS
  - Conjunto de programas de prueba (*Benchmark*)
  - Ganancia en prestaciones

# Bibliografía

## ➤ Fundamental

- 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. R nder. *Parallel Programming: for Multicore and Cluster Systems*. Springer 2010. Disponible en l nea (biblioteca UGR): <http://dx.doi.org/10.1007/978-3-642-04818-0>

# Evaluación de prestaciones de una arquitectura

- Medidas usuales para evaluar prestaciones

- Tiempo de respuesta

- Productividad: MIPS, MFLOPS

- Conjunto de programas de prueba (*Benchmark*)

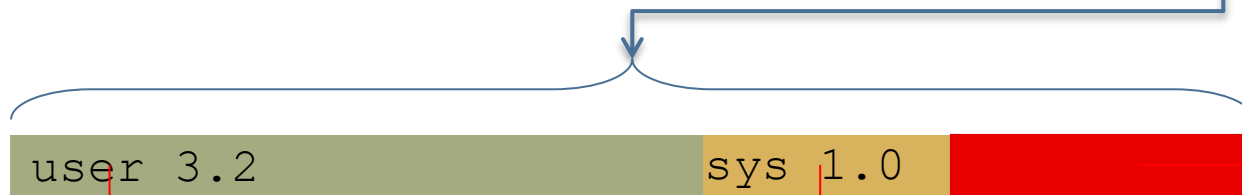
- Ganancia en prestaciones

# 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 control
  - elapsed  $\geq$  CPU time
- Con múltiples flujos de control
  - elapsed < CPU time, elapsed  $\geq$  CPU time / nº flujos control

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

Elapsed  $\leq$   
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 78%

3.2+1.0 es el del tiempo transcurrido (5.4)

# Algunas alternativas para obtener tiempos

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

La resolución depende de la plataforma

# Tiempo de CPU I

$$\text{Tiempo de CPU (T}_{\text{CPU}}) = \text{Ciclos\_del\_Programa} \times T_{\text{CICLO}} = \frac{\text{Ciclos\_del\_Programa}}{\text{Frecuencia\_de\_Reloj}}$$

$$\text{Ciclos por Instrucción (CPI)} = \frac{\text{Ciclos\_del\_Programa}}{\text{Numero\_de\_Instrucciones(NI)}}$$

$$T_{\text{CPU}} = \text{NI} \times \text{CPI} \times T_{\text{CICLO}}$$

$$\text{Ciclos\_del\_Programa} = \sum_{i=1}^n \text{CPI}_i \times \text{I}_i$$

$$\text{CPI} = \frac{\sum_{i=1}^n \text{CPI}_i \times \text{I}_i}{\text{NI}}$$

En el programa hay  $I_i$  instrucciones del tipo  $i$  ( $i=1, \dots, n$ )

Cada instrucción del tipo  $i$  consume  $\text{CPI}_i$  ciclos

Hay  $n$  tipos de instrucciones distintos.



# Tiempo de CPU II

$$T_{\text{CPU}} = NI \times \underbrace{(CPE / IPE)}_{\text{CPI}} \times T_{\text{ciclo}}$$

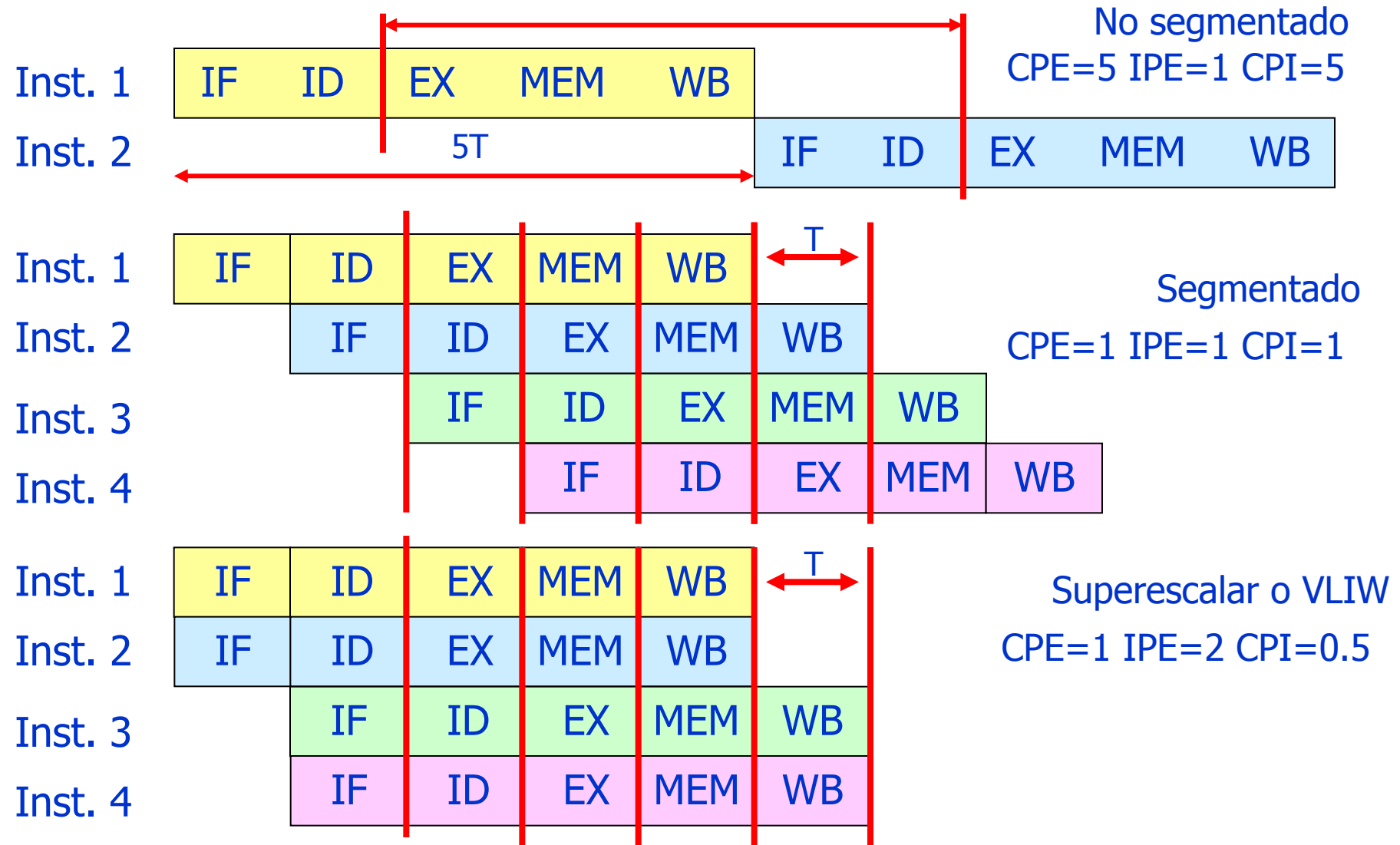
**CPI**

Hay procesadores que pueden lanzar para que empiecen a ejecutarse (emitir) varias instrucciones al mismo tiempo.

**CPE:** Número mínimo de ciclos transcurridos entre los instantes en que el procesador puede emitir instrucciones

**IPE:** Instrucciones que pueden emitirse (para empezar su ejecución) cada vez que se produce dicha emisión.

# Tiempo de CPU III



# Tiempo de CPU IV

$$T_{\text{CPU}} = \underbrace{(\text{Noper}/\text{Op\_instr})}_{\text{NI}} \times \text{CPI} \times T_{\text{ciclo}}$$

**NI**

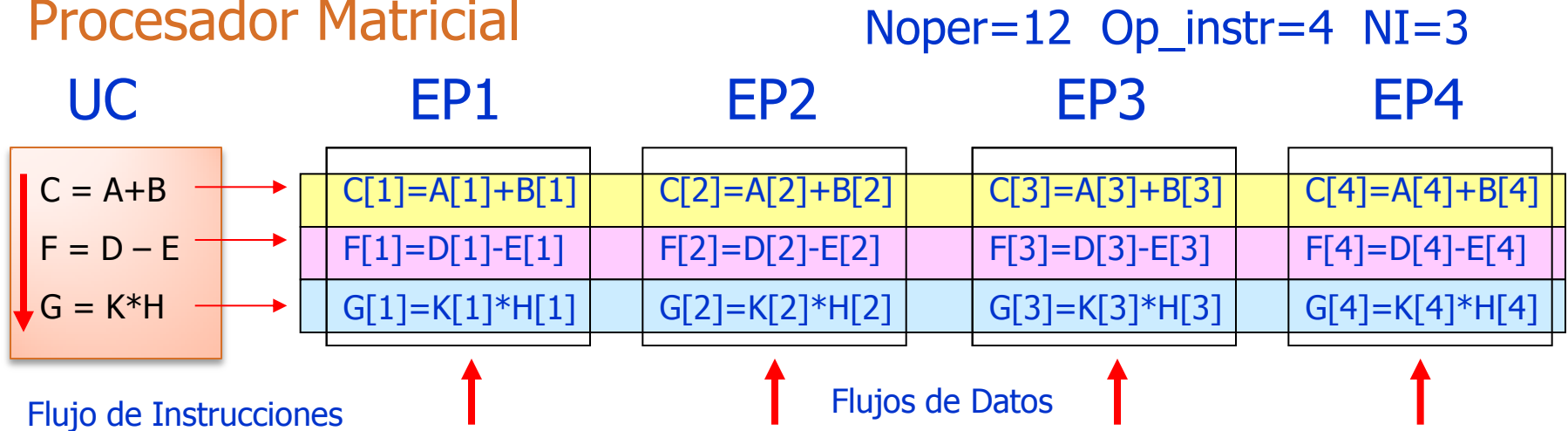
Hay procesadores que pueden codificar varias operaciones en una instrucción.

**Noper:** Número de operaciones que realiza el programa

**Op\_instr:** Número de operaciones que puede codificar una instrucción.

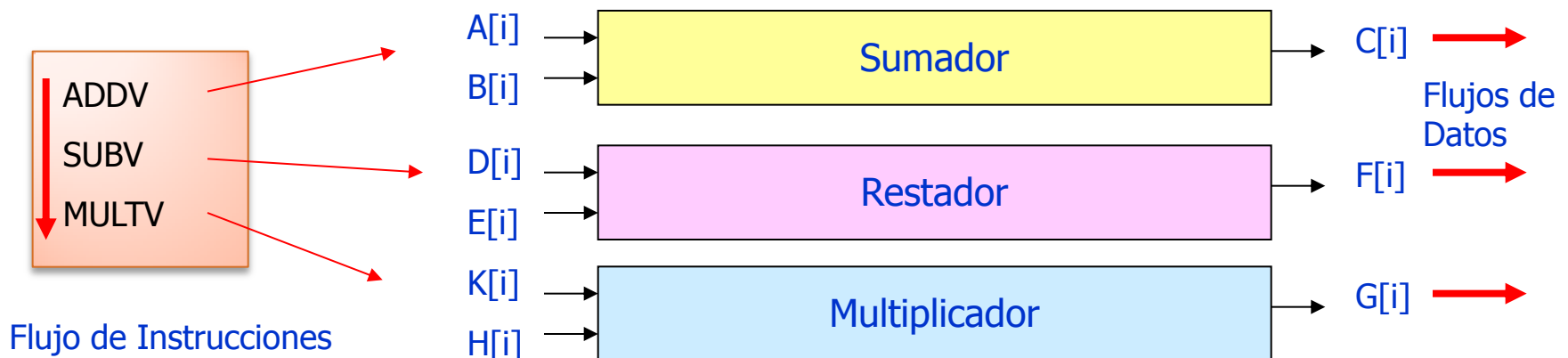
# Tiempo de CPU V

## Procesador Matricial

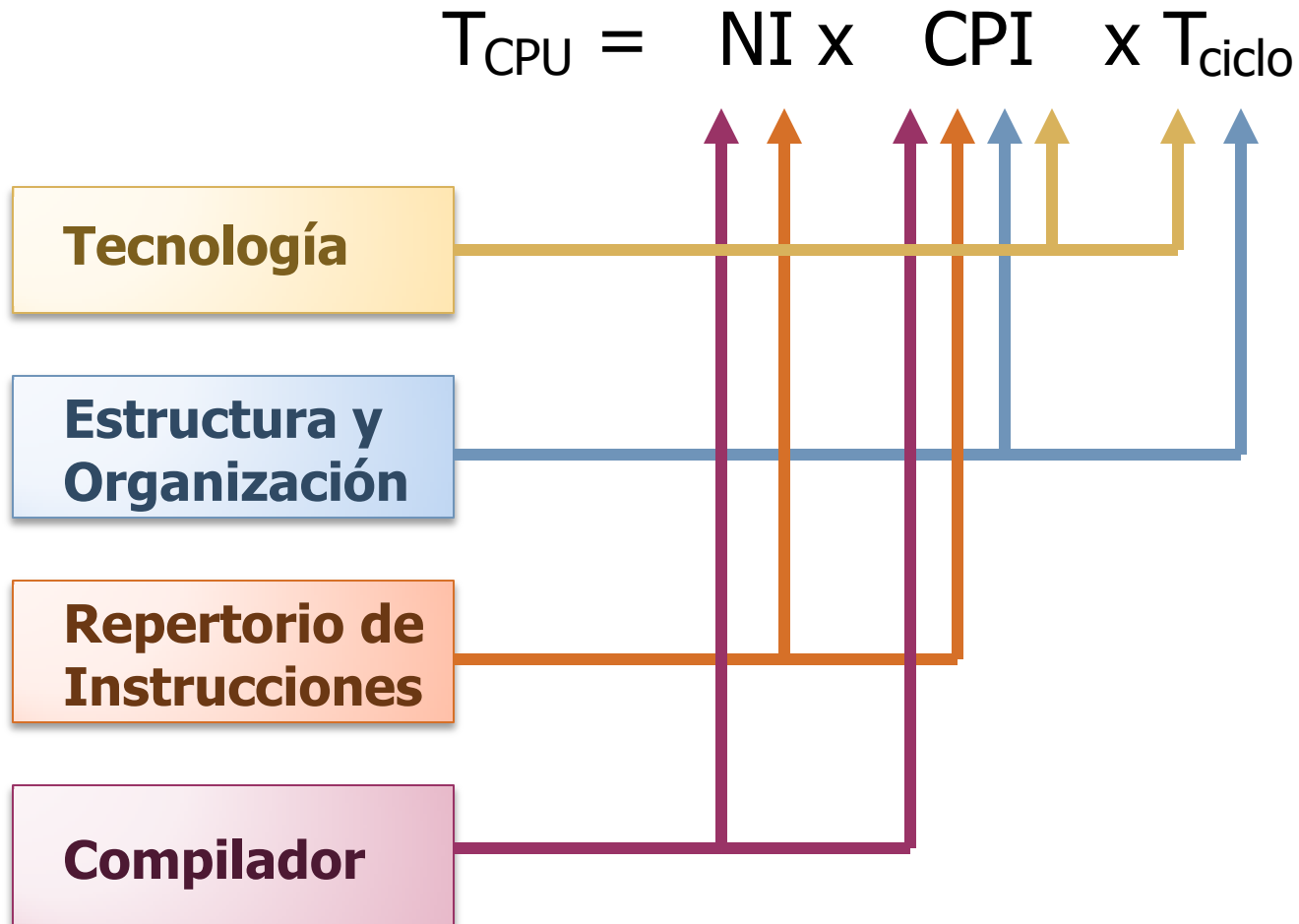


## Procesador Vectorial

Noper=12 Op\_instr=4 NI=3



# Tiempo de CPU VI



# Evaluación de prestaciones de una arquitectura

- Medidas usuales para evaluar prestaciones
  - Tiempo de respuesta
  - Productividad: MIPS, MFLOPS
- Conjunto de programas de prueba (*Benchmark*)
- Ganancia en prestaciones

# MIPS

## MIPS: Millones de Instrucciones por segundo

$$\text{MIPS} = \frac{\text{NI}}{T_{\text{CPU}} \times 10^6} = \frac{F(\text{frecuencia})}{\text{CPI} \times 10^6}$$

- **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)

*MIPS (Meaningless Indication of Processor Speed)*

# MFLOPS

**MFLOPS:** Millones de operaciones en coma flotante por segundo

$$\text{MFLOPS} = \frac{\text{Operaciones\_en\_Coma\_Flotante}}{T_{\text{CPU}} \times 10^6}$$

- **No es una medida adecuada para todos los programas** (sólo tiene en cuenta las operaciones en coma flotante del programa)
- **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** (por ejemplo, con diferente precisión, no es igual una suma que una multiplicación..):

**Es necesaria una normalización de las instrucciones en coma flotante**



# Evaluación de prestaciones de una arquitectura

- Medidas usuales para evaluar prestaciones
  - Tiempo de respuesta
  - Productividad: MIPS, MFLOPS
- Conjunto de programas de prueba (*Benchmark*)
- Ganancia en prestaciones

# 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 CPU2006: enteros (gcc, gzip, perlbnk)
  - Aplicaciones diseñadas
    - Predicción de tiempo, simulación de terremotos.

# Benchmark suites I

- Benchmark: SPEC CPU2006
  - **Dirección:** <http://www.spec.org/cpu2006/>
  - **Aplicación:** evaluación de operaciones con enteros (CINT2006) y con punto flotante (CFP2006) en un core
  - **Tipo:** aplicaciones reales
    - CINT2006: compilador gcc, compresor bzip2, planificación de vehículos de transporte, inteligencia artificial, análisis de secuencia de proteínas, compresión de vídeo, ...
    - CFP2006: dinámica de fluidos, dinámica molecular, Image Ray-tracing, programación lineal, reconocimiento de voz, modelado y predicción del tiempo atmosférico, ...
  - **Herramientas:** C, C++ y Fortran

# Benchmark suites II

- Benchmark paralelo: SPEC OMP 2001 (SPEC OpenMP)
  - **Dirección:** [www.spec.org/hpg/omp2001](http://www.spec.org/hpg/omp2001)
  - **Aplicación:** Científico
  - **Estilo:** variables compartidas.
  - **Tipo:** Aplicaciones diseñadas. Basado en SPEC CPU2000. Evalúa procesador, memoria, SO y herr. de programación
  - **Herramientas:** OpenMP

# Benchmark suites III

- Benchmark paralelo: SPEC HPC2002
  - **Dirección:** [www.specbench.org/hpc2002/](http://www.specbench.org/hpc2002/)
  - **Aplicación:** Científico.
  - **Estilo:** Variables compartidas, paso de mensajes, y combinación de ambos.
  - **Tipo:** Basado en aplicaciones HPC diseñadas reales. Evalúa procesador, comunicación, E/S, compilador y bibliotecas
  - **Herramientas:** Serie, OpenMP, MPI, combinación MPI-OpenMP.

# Benchmark suites VI

- Benchmark: TPC (Transaction Processing Performance Council)
  - **Dirección:** [www.tpc.org](http://www.tpc.org)
  - **Aplicación:** Procesamiento de transacciones o OLTP (TPC-C); sistemas de soporte de decisiones o DSS (TPC-R, TPC-H); comercio electrónico o e-commerce (TPC-W) o servidores web y de aplicaciones (TPC-App).
  - **Tipo:** entradas software comercial (bases de datos, servidores de información de Internet) y carga de trabajo diseñada

# Benchmark suites VII

- Benchmark paralelo : NPB2, NPB3 (NAS Parallel Benchmark)
  - **Dirección:**  
<http://www.nas.nasa.gov/Resources/Software/npb.html>
  - **Aplicación:** Científico.
  - **Estilo:** paso de mensajes, variables compartidas.
  - **Tipo:** núcleos y aplicaciones diseñadas.
  - **Herramientas:** NPB2 (MPI). NPB3 (OpenMP, java, HPF)



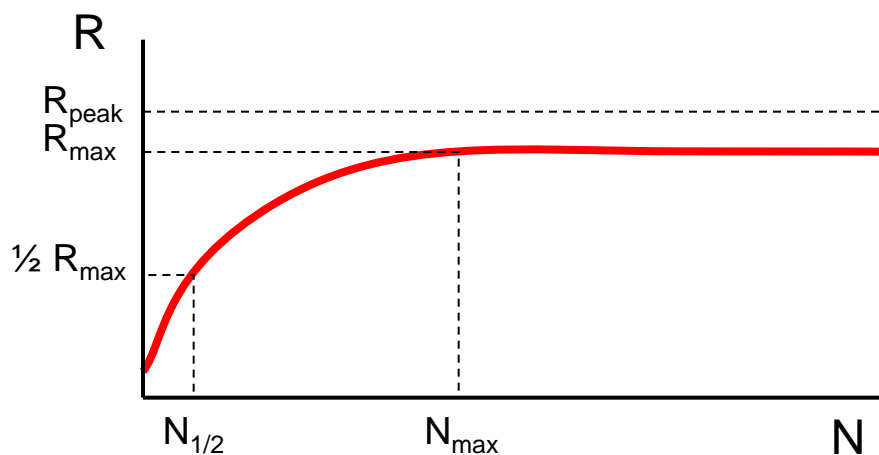
# Benchmark suites VIII

- Implementaciones de la biblioteca BLAS (*Basic Linear Algebra Subprograms*)
  - **Tipo:** núcleos con operaciones del álgebra lineal:
    - operaciones con vectores, como producto escalar o AXPY (*Alpha X Plus Y*) ,
    - vector-matriz, como producto matriz por vector
    - matriz-matriz, producto de matrices (*GEMM-GEneral Matrix Multiply*)
  - **Herramientas:** hay implementaciones con diferentes herramientas de programación (Fortran, C, C++, OpenCL, CUDA, ...) y optimizadas para diferentes arquitecturas (Intel x86, AMD, GPU, ...)

# LINPACK

El núcleo de este programa es una rutina denominada DAXPY (**D**ouble **p**recision- **r**eal **A**lpha **X** **P**lus **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];
```



## TOP500

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

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

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

# Evaluación de prestaciones de una arquitectura

- Medidas usuales para evaluar prestaciones
  - Tiempo de respuesta
  - Productividad: MIPS, MFLOPS
- Conjunto de programas de prueba (*Benchmark*)
- Ganancia en prestaciones

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

Si en un computador se incrementan las prestaciones de un recurso haciendo que su velocidad sea **p veces mayor** (ejemplos: se utilizan **p procesadores** en lugar de uno, la ALU realiza **las operaciones en un tiempo p veces menor**,...):

El incremento de velocidad que se consigue en la nueva situación con respecto a la previa (**máquina base**) se expresa mediante la ganancia de velocidad o *speed-up*,  $S_p$

$$S_p = \frac{V_p}{V_1} = \frac{T_1}{T_p}$$

$V_1$  Velocidad de la máquina base

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

$T_1$  Tiempo de ejecución en la máquina base

$T_p$  Tiempo de ejecución en la máquina mejorada

# 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:

$$S \leq \frac{p}{1 + f(p - 1)}$$

donde **f** es la fracción del tiempo de ejecución en la máquina sin la mejora durante el que no se puede aplicar esa 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  $S \leq 2/(1+0.75)=1.14$**

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

Ley enunciada por Amdahl en relación con la eficacia de los computadores paralelos: dado que en un programa hay código secuencial que no puede paralelizarse, los procesadores no se podrían utilizar eficazmente. (Tema 2)

# 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 Millenium”. IEEE Computer. Julio, 2000.
- O’Neal, D.: “On Microprocessors, Memory Hierarchies, and Amdahl’s Law”.