ARQUITECTURA DE COMPUTADORES II

AUTORES:
David Expósito Singh
Florin Isaila
Daniel Higuero Alonso-Mardones
Javier García Blas
Borja Bergua Guerra

Área de Arquitectura y Tecnología de Computadores
Departamento de Informática
Universidad Carlos III de Madrid

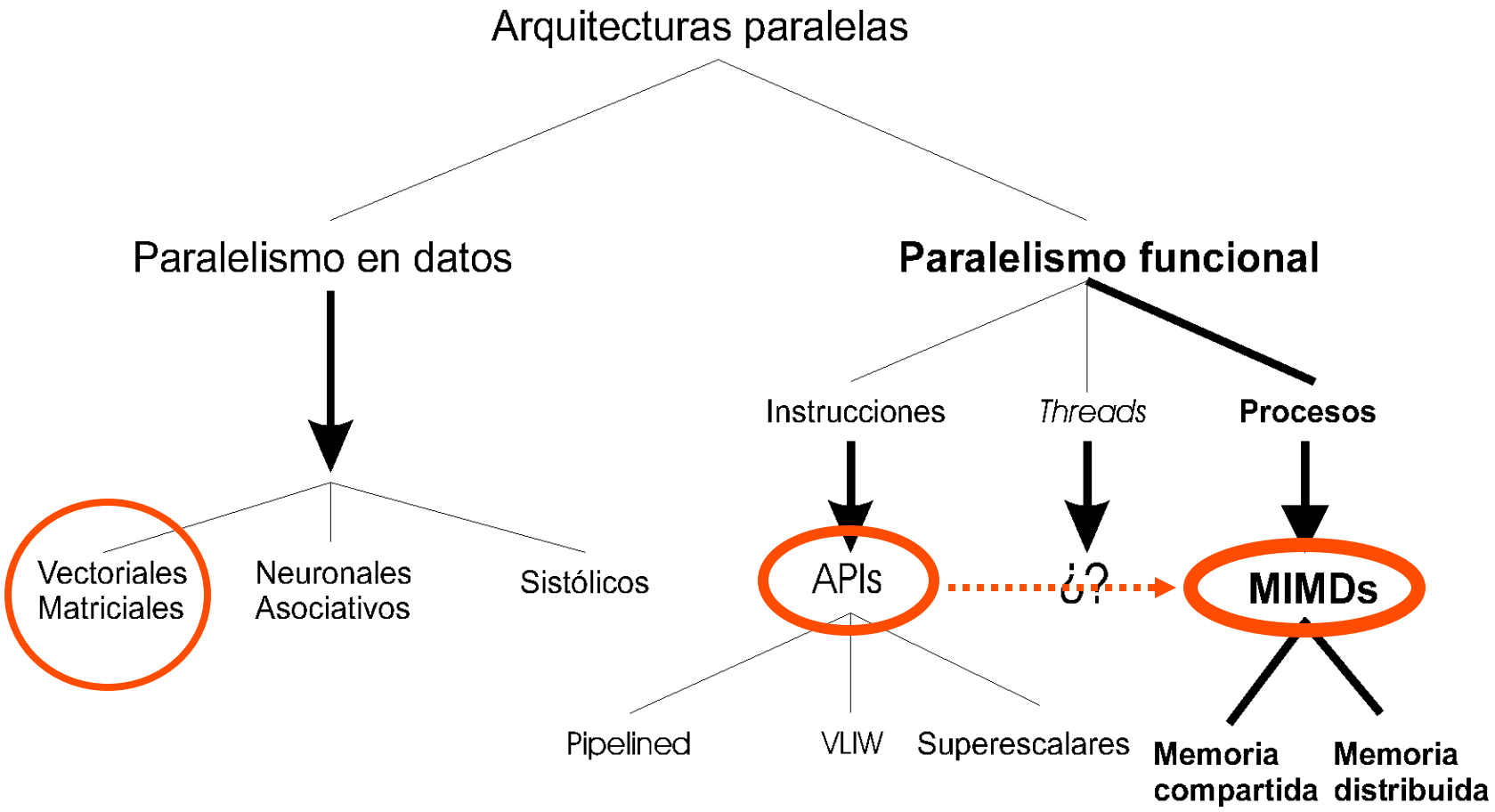
Julio de 2012

TEMA 1:
EJECUCIÓN PARALELA: FUNDAMENTOS(II)

Índice

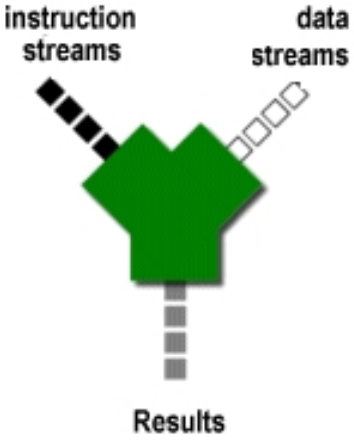
1. ¿Por qué se demanda ejecución paralela?
   - Demanda de cómputo
   - Logros tecnológicos
   - Logros en la arquitectura
   - Aspectos económicos
2. **Arquitecturas paralelas: clasificación**
   - **Procesadores vectoriales**
   - **Multiprocesadores**
   - ***Clusters* y Cluster Computing**
   - **Computación GRID**
5. Beneficios del paralelismo
6. Paralelización: principios
   - Ley de Amdahl
   - *Overhead* del paralelismo
   - Proximidad y equilibrado
   - Tamaño del 'grano'
   - Depuración de pp. paralelos
7. Casos reales:
   - Google (Hw)
   - Top500

Arquitecturas paralelas: clasificación

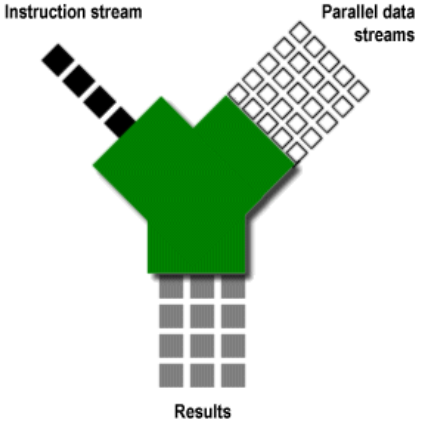


Arquitecturas

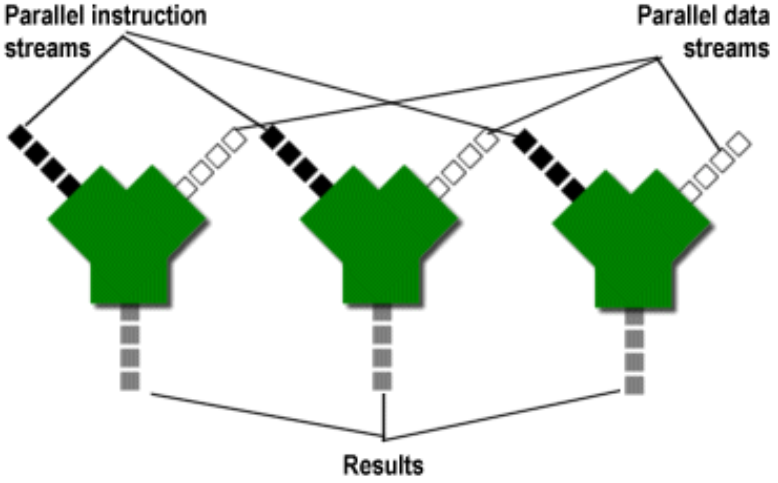
Escalar



Vectorial



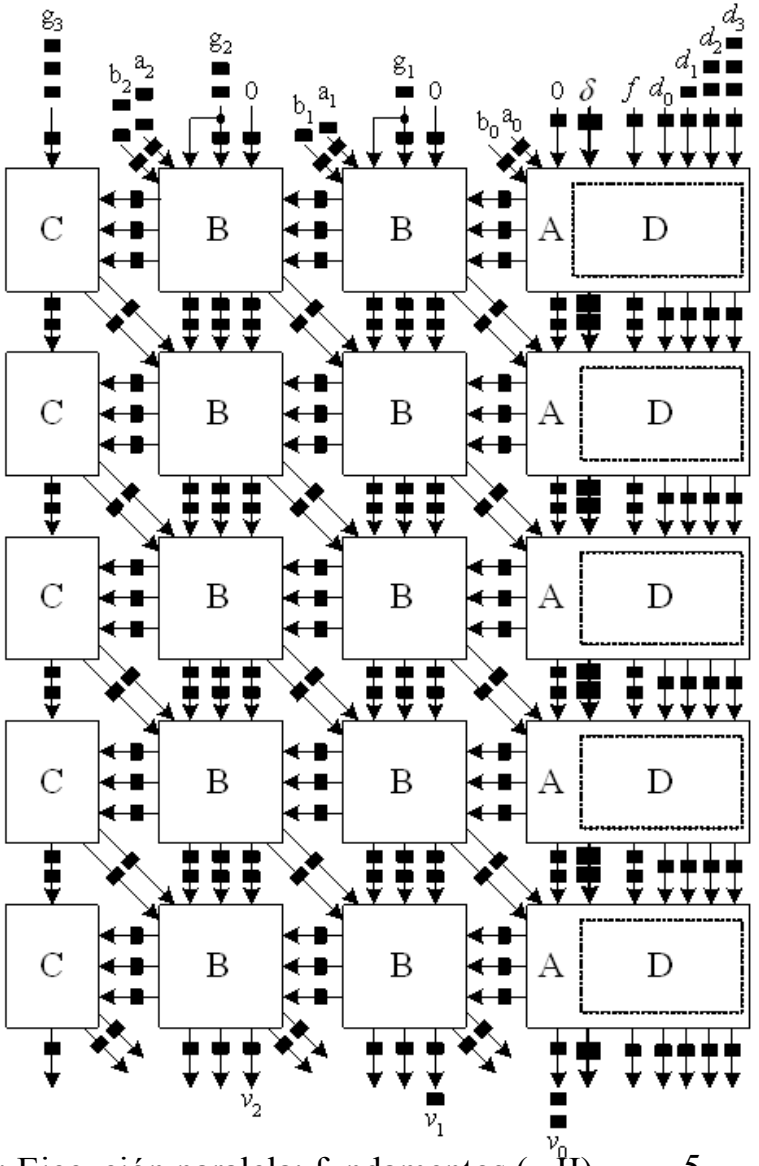
Paralela



- Clasificación de Flynn (1972):
  - Single Instruction, single data (SISD)
  - Single Instruction, multiple data (SIMD)
  - Multiple instruction, multiple data (MIMD)

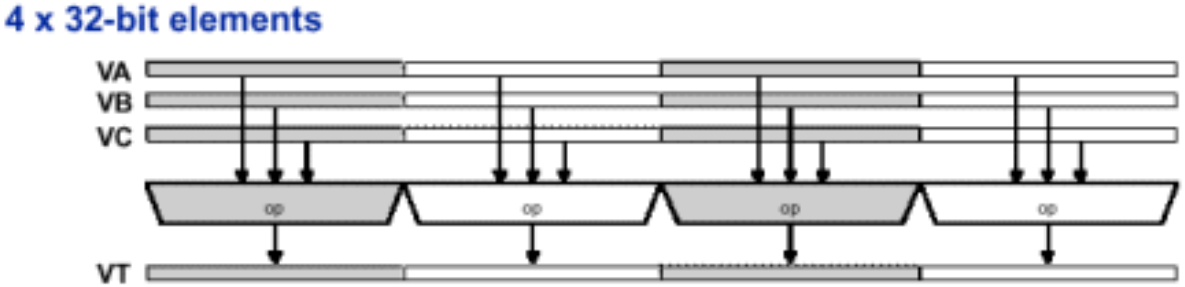
Arquitecturas sistólicas.

- Originados por límite en el nivel de integración.
- Unidades de procesamiento muy simple (hasta 1 bit).
- Problemas algebraicos muy regulares.
- Alta capacidad de procesamiento.



Procesadores vectoriales

- Idea:
  - Los operandos de las instrucciones son vectores.
  - Unidades Funcionales operan (***en paralelo***) con los componentes del vector.



- Uso:
  - Fueron los grandes "supercomputadores" en el pasado.
  - Sólo son útiles para determinados tipos de aplicaciones. En general, aplicaciones "numéricas".

Procesadores vectoriales

Ejemplo de código vectorial

Versión escalar:

```
    …
loadf f2, (r2)          ; load C(i,j)
loop:
    loadf f3, (r3)      ; load A(i, k)
    loadf f4, (r4)      ; load B(k,j)
    mypf f3,f3,f4       ; A(i,k)*B(k,j)

    addf f2,f2,f3       ; update C(i,j)
    store f2 (r2)       ; store C(i,j)
    addi r1,r1,#4       ; incr. ptr. A(i,k+1)
    addi r3,r3,#4       ; incr. ptr. B(k+1,j)

    bnz r1,loop:        ; branch to loop if (r1/4).eq. n)
    …
```

Multiplicación de matrices:

```
DO  i = 1,n
   DO  k=1,n
      DO  j= 1,n
         C(i,j)= C(i,1:n)+A(i,k)*B(k,j)
      ENDDO
   ENDDO
```

Versión vectorial:

```
 …

LV V2, (r2)             ; load A(i,k)
LV V3,(r3); load B(k, j)
MULTV v3,v2,v3          ; A(i,k)*B(k,j)



…
```

Procesadores vectoriales

- Algunos vectoriales clásicos:

Computer	clock cycle (ns)	max CPUs	peak single CPU (Mflop/s)	peak total (Mflop/s)
Cray 2	4.1	4	488	1952
Cray 3	2.0	16	1000	16000
Cray Y-MP	6.0	8	333	2667
Cray C90	4.0	16	1000	16000
Fujitsu VP2600	4.0	1	4000	
Hitachi S-820/80	4.0	1	2000	
NEC SX-3	2.9	4	5500	22000

Procesadores vectoriales

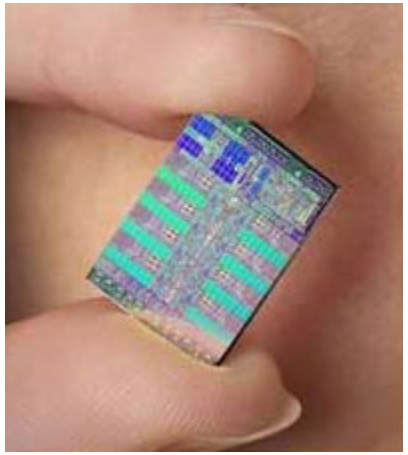
- **Ejemplo de arquitectura actual que emplea procesadores vectoriales:**

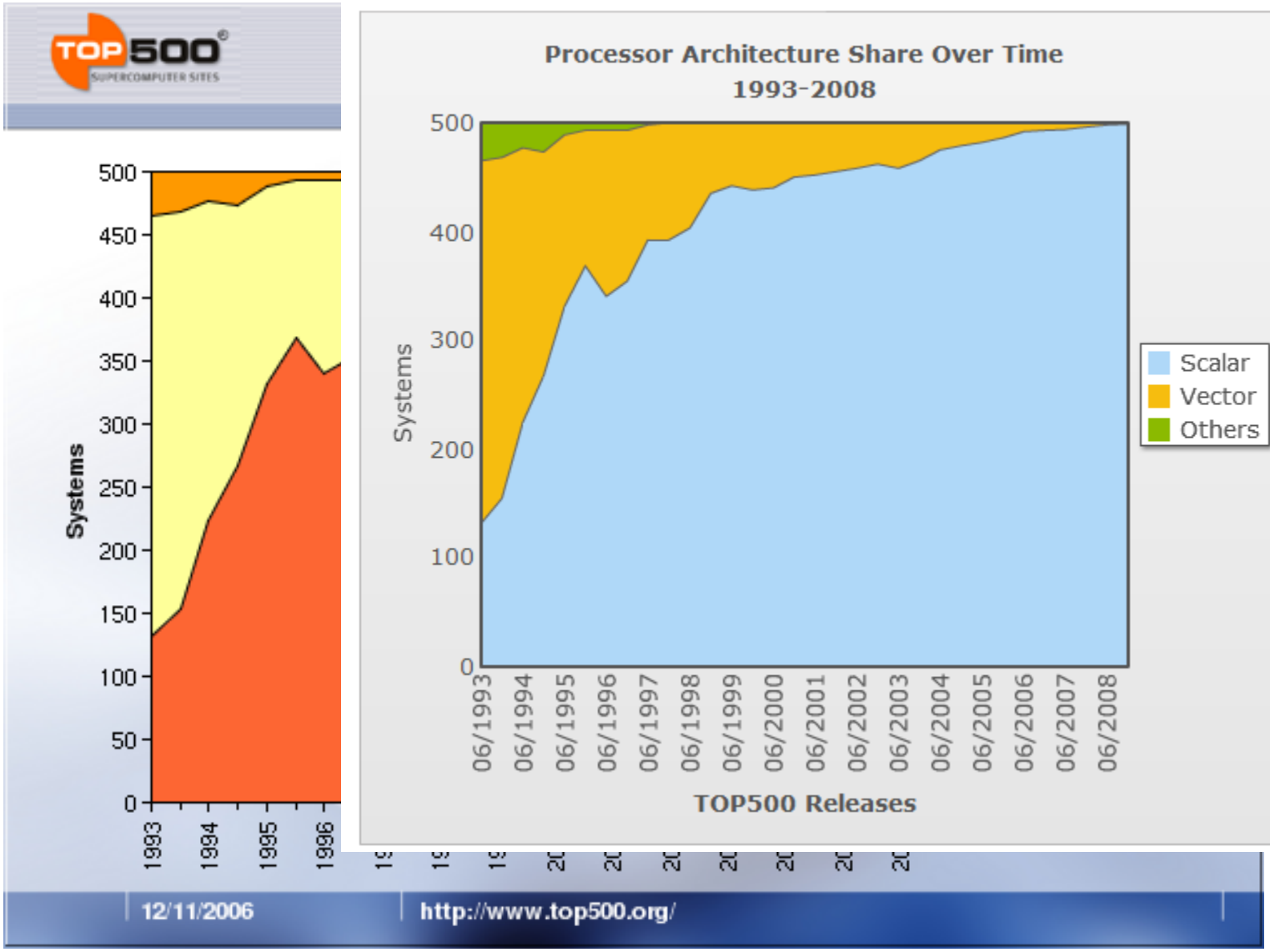
- MIPS III Emotion Engine CPU.
- 2 SIMD unidades vectoriales en punto flotante.

Procesadores vectoriales

- **Ejemplo de arquitectura actual que emplea procesadores vectoriales:**

- Procesador Cell con un núcleo basado en IBM POWER y 7 núcleos SPE.



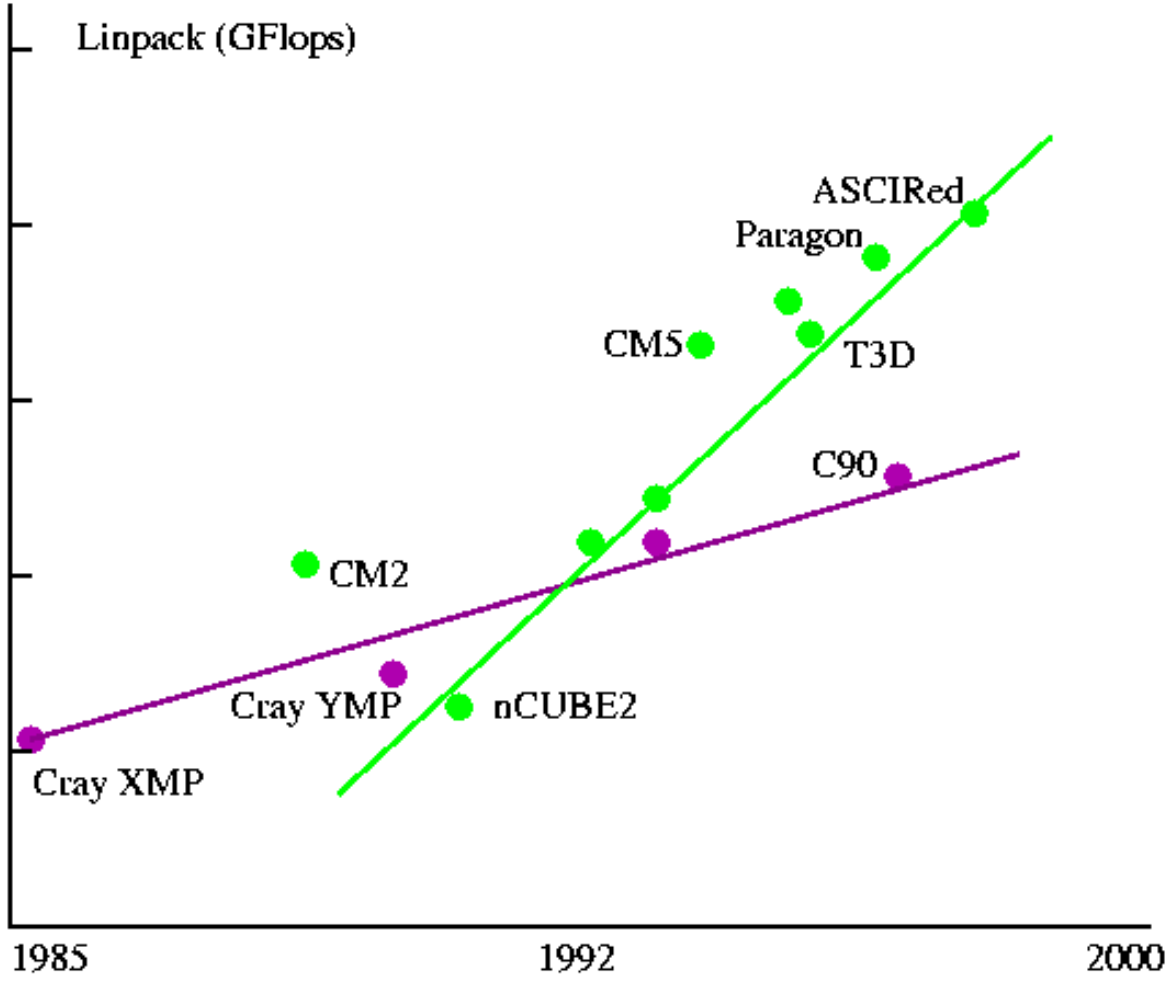

TOP500
SUPERCOMPUTER SITES
Processor Architecture Share Over Time
1993-2008
Systems
500
400
300
200
100
0
06/1993
06/1994
06/1995
06/1996
06/1997
06/1998
06/1999
06/2000
06/2001
06/2002
06/2003
06/2004
06/2005
06/2006
06/2007
06/2008
TOP500 Releases
Scalar
Vector
Others
Systems
500
450
400
350
300
250
200
150
100
50
0
1993
1994
1995
1996
12/11/2006
http://www.top500.org/

Multiprocesadores

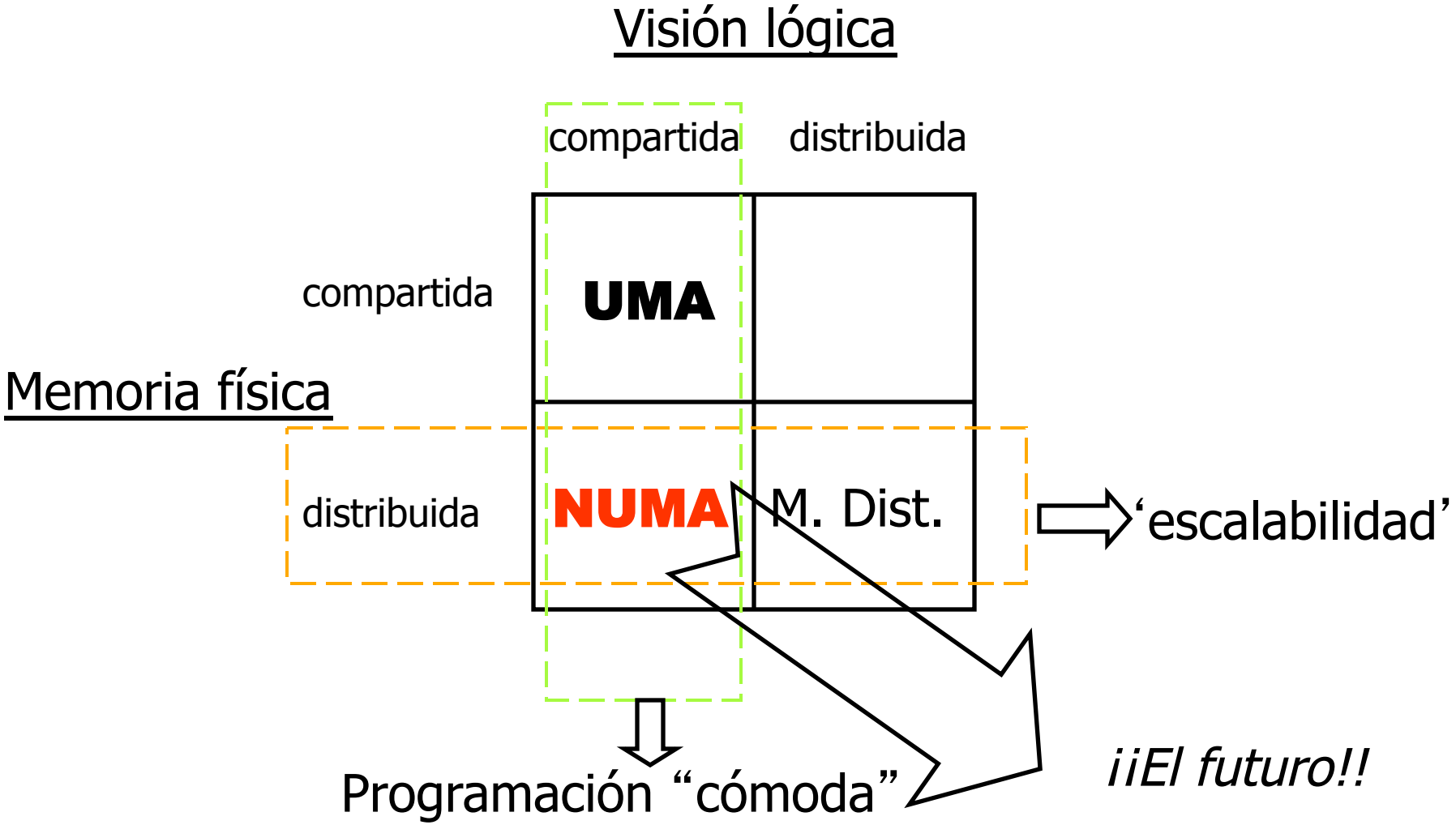
- Qué es un **multiprocesador** (**MP**):
  - Son sistemas **MIMD.**
  - Dos o más **procesadores**, en general, ***idénticos.***
  - "**Acoplamiento fuerte**": baja latencia en accesos.
  - Los procesadores son **microprocesadores comerciales.**
  - Los procesadores **comparten** información.
  - **Único S.O.**
  - Extensión **natural** (y **económica**) de los monoprocesadores.
  - La mayoría de los constructores ofrece versiones MP de sus computadores.

> *"Si compra un procesador para su Sun Enterprise, le regalamos otro igual" --Sun Microsystems, 1999*

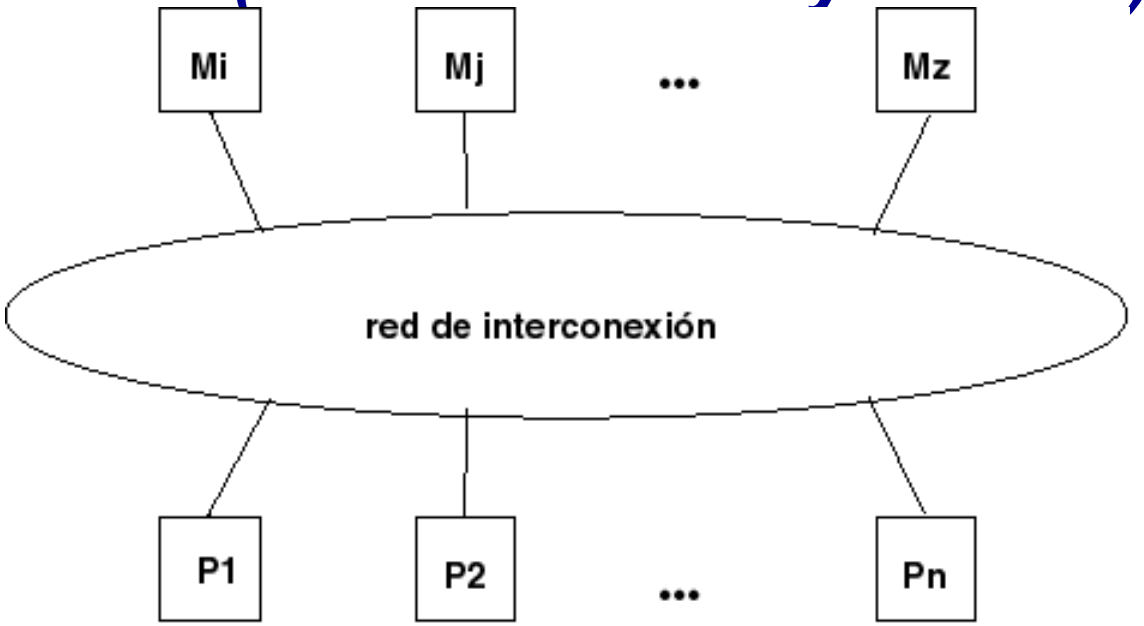
Vectoriales .vs. multiprocesadores



Clasificación de los MP



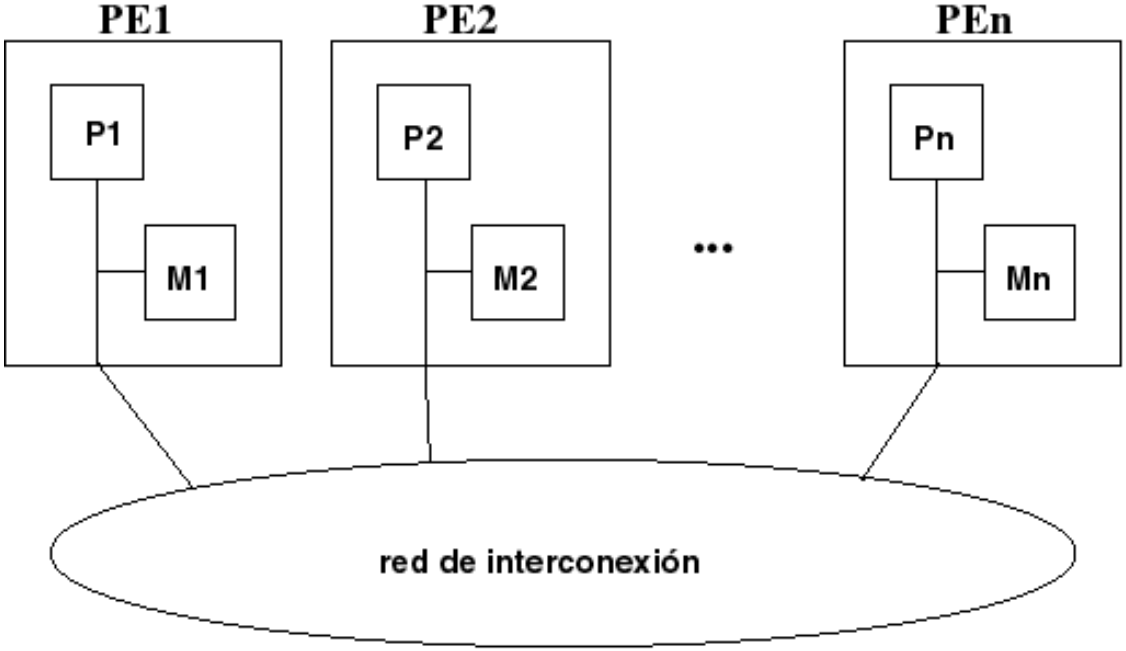
UMA *(Uniform Memory Access)*



UMA

- Sun Fire 15000
  Hasta 106 UltraSparc III

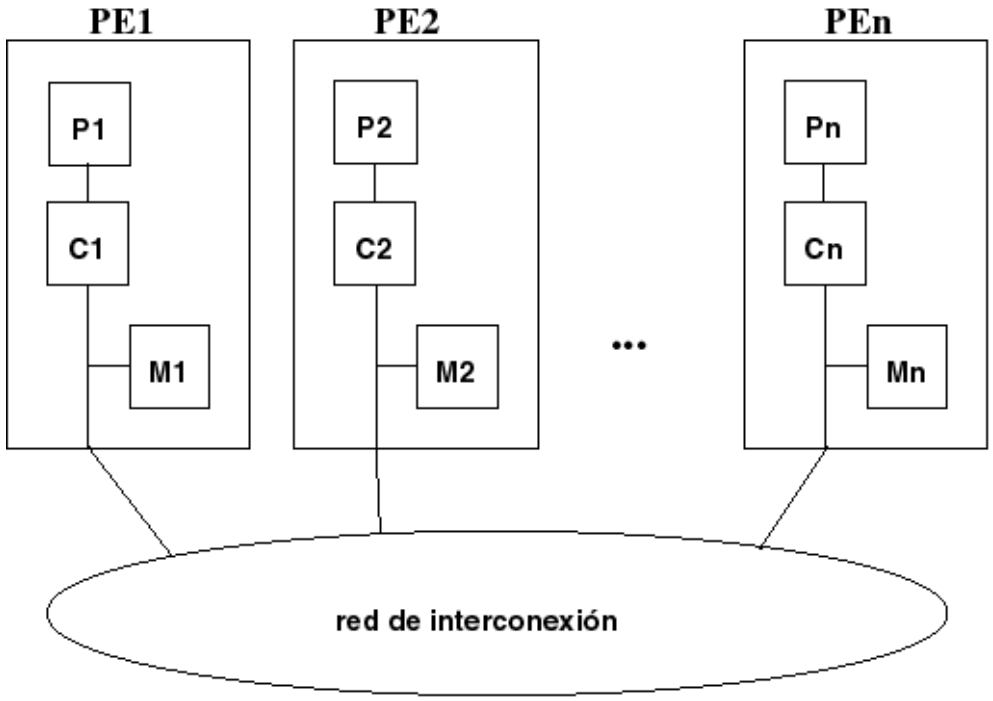
NUMA *(Non Uniform Memory Access)*



NUMA

- Explota principio de localidad.
- Favorece la escalabilidad.
- Ejemplo: Cray T3E

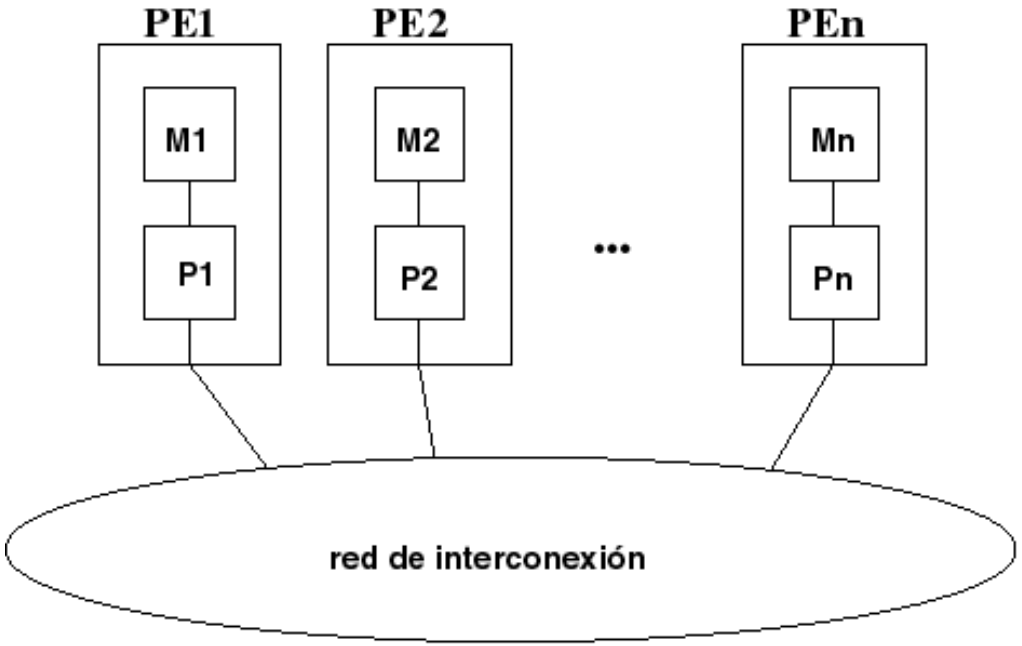
CC-NUMA *(Cache Coherent NUMA)*



CCNUMA

- Memoria principal compartida.
- Coherencia en la gestión de la memoria caché.
- Ejemplo: SGI Origin 2000.

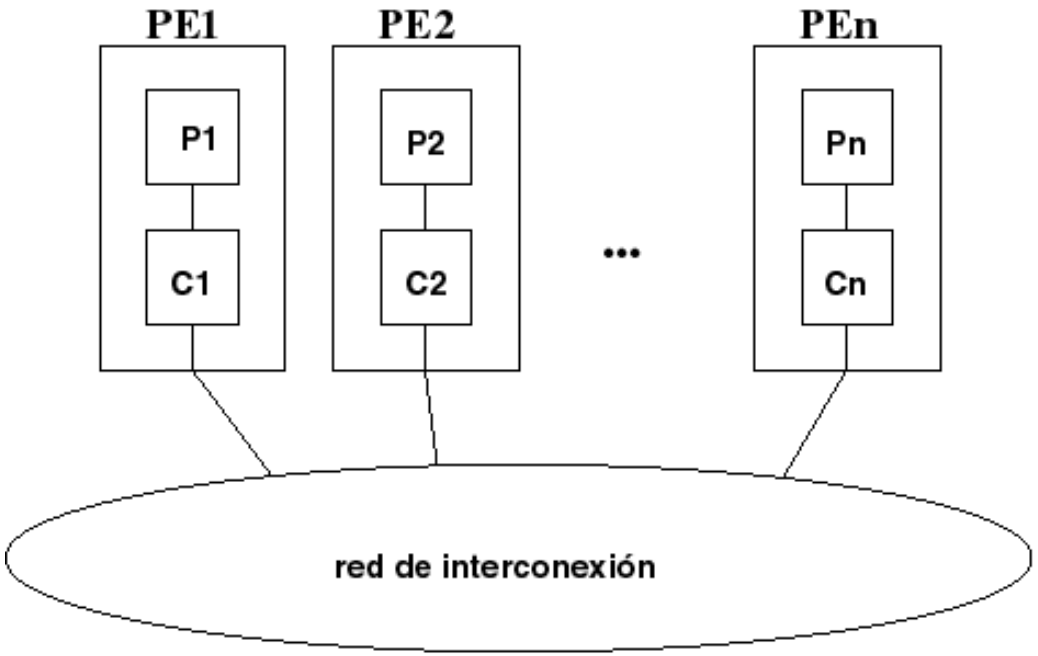
M. Distribuida (Multicomputadores)



M. Distribuida

- Mínima gestión de memoria.
- Alta escalabilidad.
- Ejemplo: Cray T3D, IBM SP2.

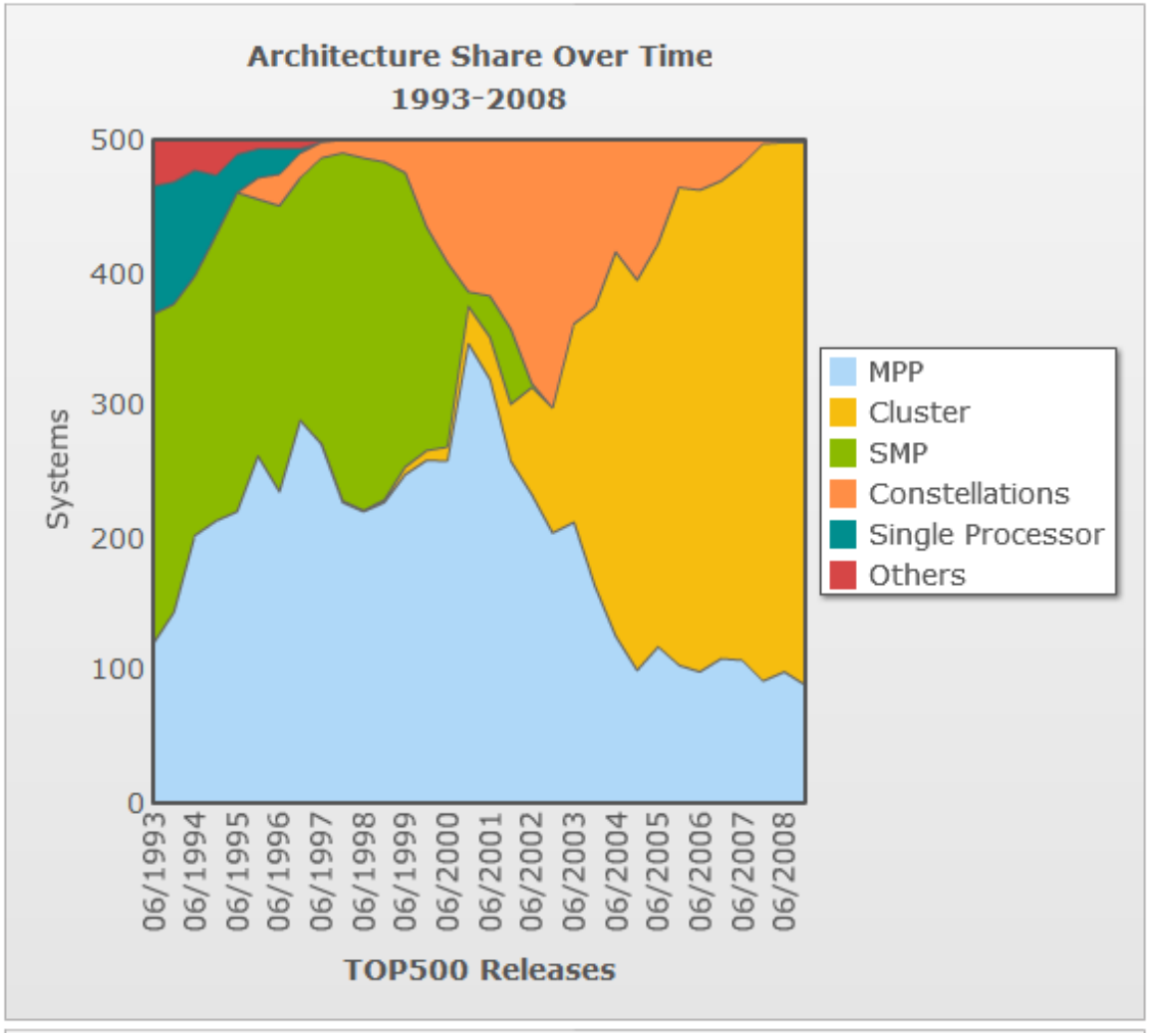
COMA *(Cache Only Machine)*



COMA

- Toda la memoria es una caché.
- Coherencia hardware.
- Ejemplo: Kendall Square Research 1.

Evolución del Top500



La convergencia

Uso de recursos distribuidos:

- **Redes de estaciones de trabajo**: empleo de recursos pertenecientes a una misma organización.
- **Grids**: uso de recursos distribuidos de distintas organizacones.
- **Internet computing**: uso de ordenadores personales a escala global (SETI@home).

Clusters y Cluster Computing

- Qué es un cluster:

> A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected **stand-alone/complete computers** cooperatively working together as a **single,** integrated computing resource. [Buyya98]

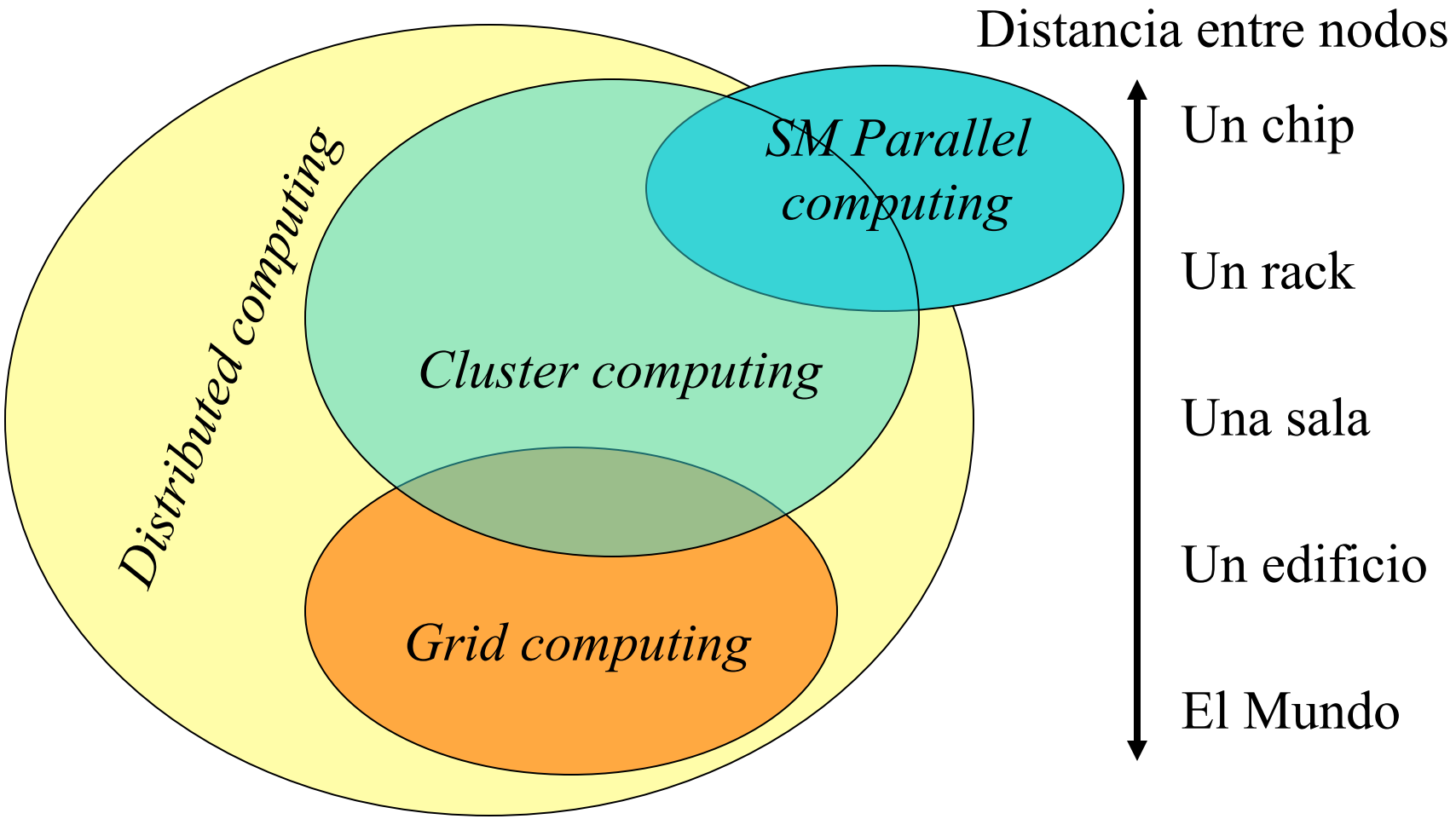
- Infraestructura de comunicación:
  - Redes más rápidas que las LAN habituales.
  - Protocolos de comunicación con latencias bajas.
  - Conexión más "débil" que en MP.

Clusters y Cluster Computing

- Ejemplos de redes de clusters:
  - Ethernet (10Mbps) (*), Fast Ethernet (100Mbps), Gigabit Ethernet (1Gbps), ATM, Myrinet (1.2Gbps), Fiber Channel, FDDI, etc.

- Algunos proyectos sobre clusters:
  - **Beowulf** (CalTech and NASA) - USA
  - Condor - Wisconsin State University, USA
  - DQS (Distributed Queuing System) - Florida State University, USA.
  - **HPVM** -(High Performance Virtual Machine),UIUC&now UCSB,USA
  - *far* - University of Liverpool, UK
  - **Gardens** - Queensland University of Technology, Australia
  - **MOSIX** - Hebrew University of Jerusalem, Israel
  - **NOW** (Network of Workstations) - Berkeley, USA

Cluster computing vs. otros



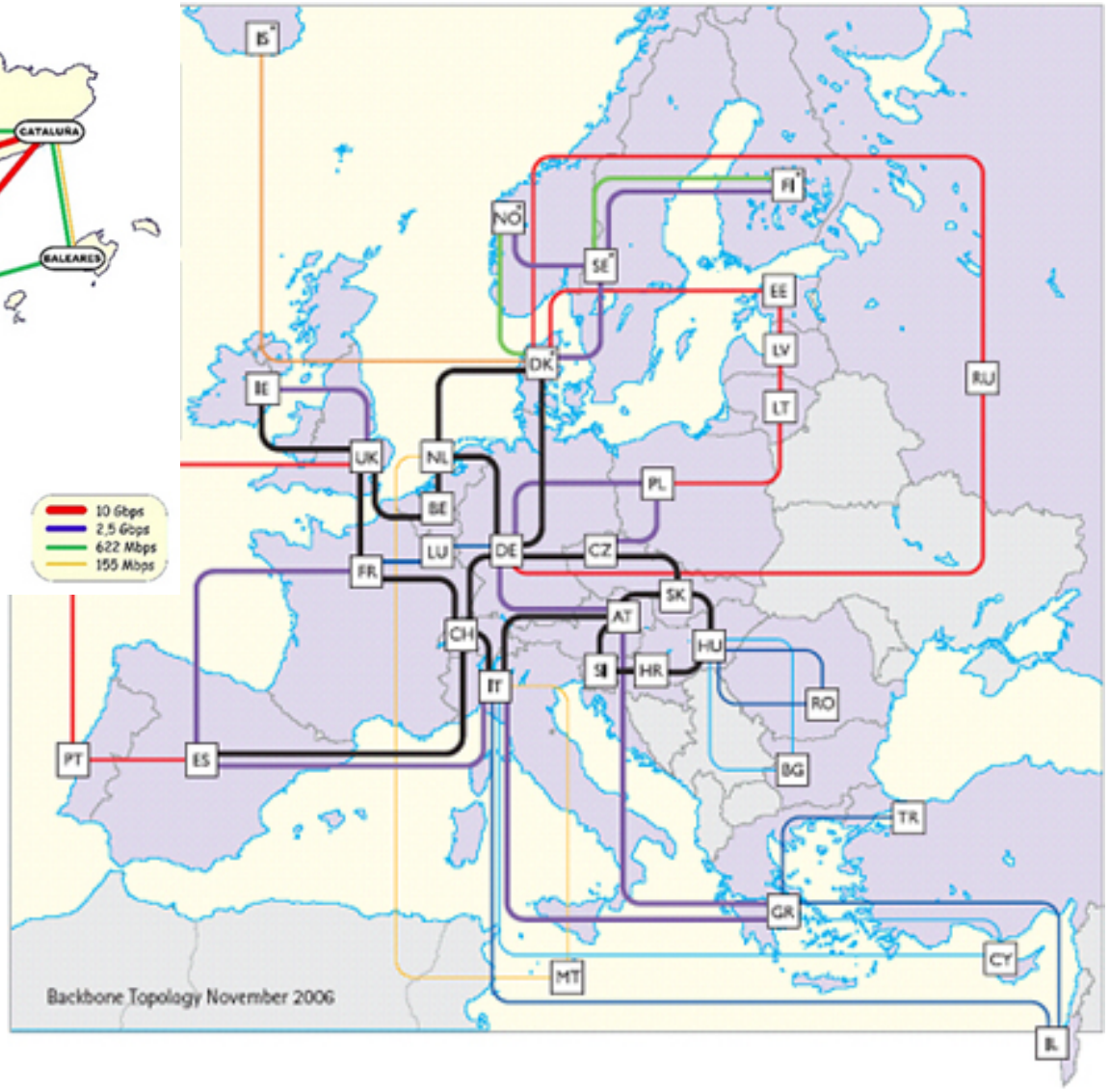
Computación GRID

- ¿Qué es?:

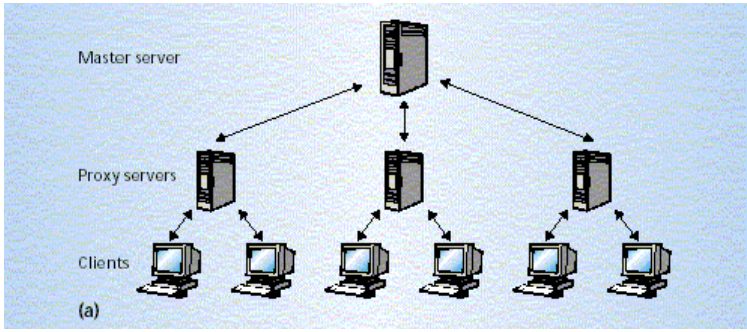
  Artículo en IEEE Computer en enero de 2000 (antes de que se acuñara el término GRID Computing): *Distributed Net Applications Create Virtual Supercomputers*

- Cluster vs. GRID:
  - **Cluster**: Red de máquinas dedicadas 100 % a ejecutar una tarea específica (normalmente en una misma entidad).
  - **Computación GRID**: red de computadores con software adecuado para compartir todos los recursos computacionales existentes en diferentes lugares del planeta.


ASTURIAS
CANTABRIA
GALICIA
PAIS VASCO
NAVARRA
RIOJA
CASTILLA Y LEÓN
ARAGÓN
CATALUÑA
MADRID
NACIONAL
BALEARES
CASTILLA LA MANCHA
VALENCIA
EXTREMADURA
MURCIA
ANDALUCIA
CANARIAS (tenerife)
CANARIAS (las palmas)


10 Gbps
2,5 Gbps
622 Mbps
155 Mbps
IS
NO
FI
SE
EE
LV
LT
RU
DK
IE
UK
NL
PL
BE
LU
DE
CZ
FR
SK
AT
CH
HU
SI
HR
IT
RO
PT
ES
BG
TR
GR
CY
MT
Backbone Topology November 2006

Requisitos para un proyecto GRID



- Tasa elevada de calculo/datos.
  No congestionar la red por constante comunicación.
- Tareas con paralelismo independiente:
  Para aplicaciones con muchas sincronizaciones se utiliza en hardware los MP o software los Clusters.
- Tareas que toleran errores.
  Si un paquete de información no ha sido tratado correctamente se descarta, o se compara con otros resultados de otros PC's (SETI)
- Las redes mejorarán rápidamente en estos años.

Índice

1. ¿Por qué se demanda ejecución paralela?
   - Demanda de cómputo
   - Logros tecnológicos
   - Logros en la arquitectura
   - Aspectos económicos
2. Arquitecturas paralelas: clasificación
   - Procesadores vectoriales
   - Multiprocesadores
   - *Clusters* y Cluster Computing
   - Computación GRID
5. **Beneficios del paralelismo**
6. Paralelización: principios
   - Ley de Amdahl
   - *Overhead* del paralelismo
   - Proximidad y equilibrado
   - Tamaño del 'grano'
   - Depuración de pp. paralelos
7. Casos reales:
   - Google (Hw)
   - Top500

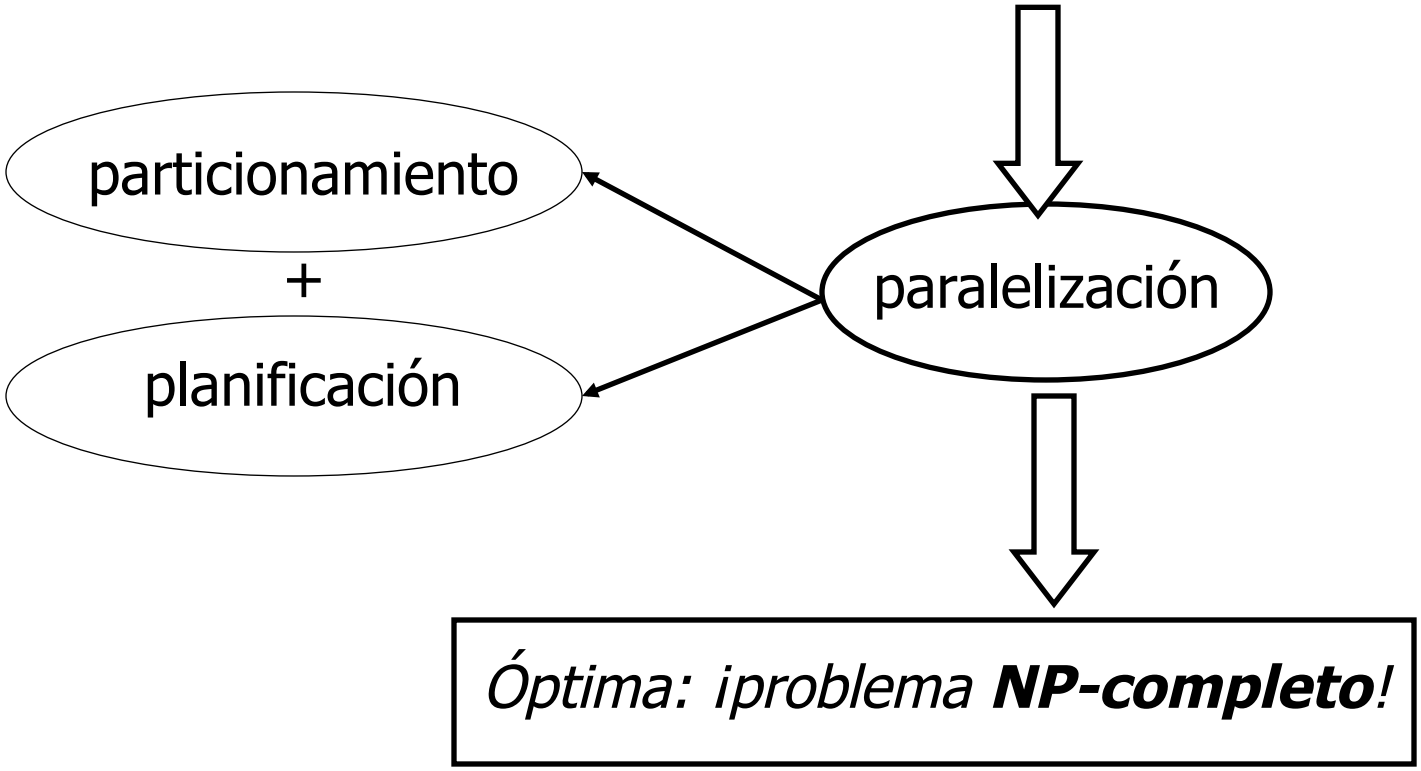
Beneficios del paralelismo

- Concurrencia: "*Coincidir en el espacio más de dos personas o cosas*".
  - En informática: existencia simultánea (no implica ejecución simultánea) de varios procesos en ejecución.
  - Favorece la comprensión del código.

- Paralelismo: caso particular de concurrencia en el que existe *ejecución simultánea* de instrucciones.
  - Persigue la reducción del tiempo de ejecución.
  - Aumento del *troughput*.

Beneficios del paralelismo

- ¿Merece la pena usar paralelismo?
  - **Fácil: multiproceso:** ⇧ ***throughput***
  - **(Muy) difícil:** *¡el gran reto!*. ⇩ ***t ejecución***



Índice

1. ¿Por qué se demanda ejecución paralela?
   - Demanda de cómputo
   - Logros tecnológicos
   - Logros en la arquitectura
   - Aspectos económicos
2. Arquitecturas paralelas: clasificación
   - Procesadores vectoriales
   - Multiprocesadores
   - *Clusters* y Cluster Computing
   - Computación GRID
5. Beneficios del paralelismo
6. **Paralelización: principios**
   - **Ley de Amdahl**
   - ***Overhead* del paralelismo**
   - **Proximidad y equilibrado**
   - **Tamaño del 'grano'**
   - **Depuración de pp. paralelos**
7. Casos reales:
   - Google (Hw)
   - Top500

Paralelización: principios

- **¿Quién paraleliza?**

Automáticamente:

Compiladores-paralelizadores (Polaris.)

Sistema de ejecución (Aurora)

S.O.

Manualmente:

Programador: paralelismo explícito en el programa: Occam, Ada, HPF, pC++, PVM, MPI, OpenMP, etc.

1. Encontrar **suficiente paralelismo**: ley de Amdahl.
2. **¡No paralelizar por que sí!**: considerar el overhead asociado.
3. **Proximidad** (localidad) y **equilibrado** (balanceo) de carga.
4. **Depuración** de programas paralelos: lógica y rendimiento.

Ley de Amdahl

- 1. Encontrar **suficiente paralelismo**: Ley de Amdahl

  Sea un MP con p procesadores:

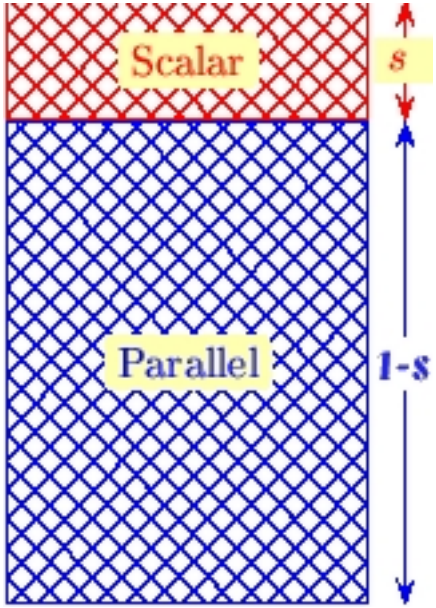
  speedup (p) = tiempo_de_ejecución (1) / tiempo_de_ejecución (p)

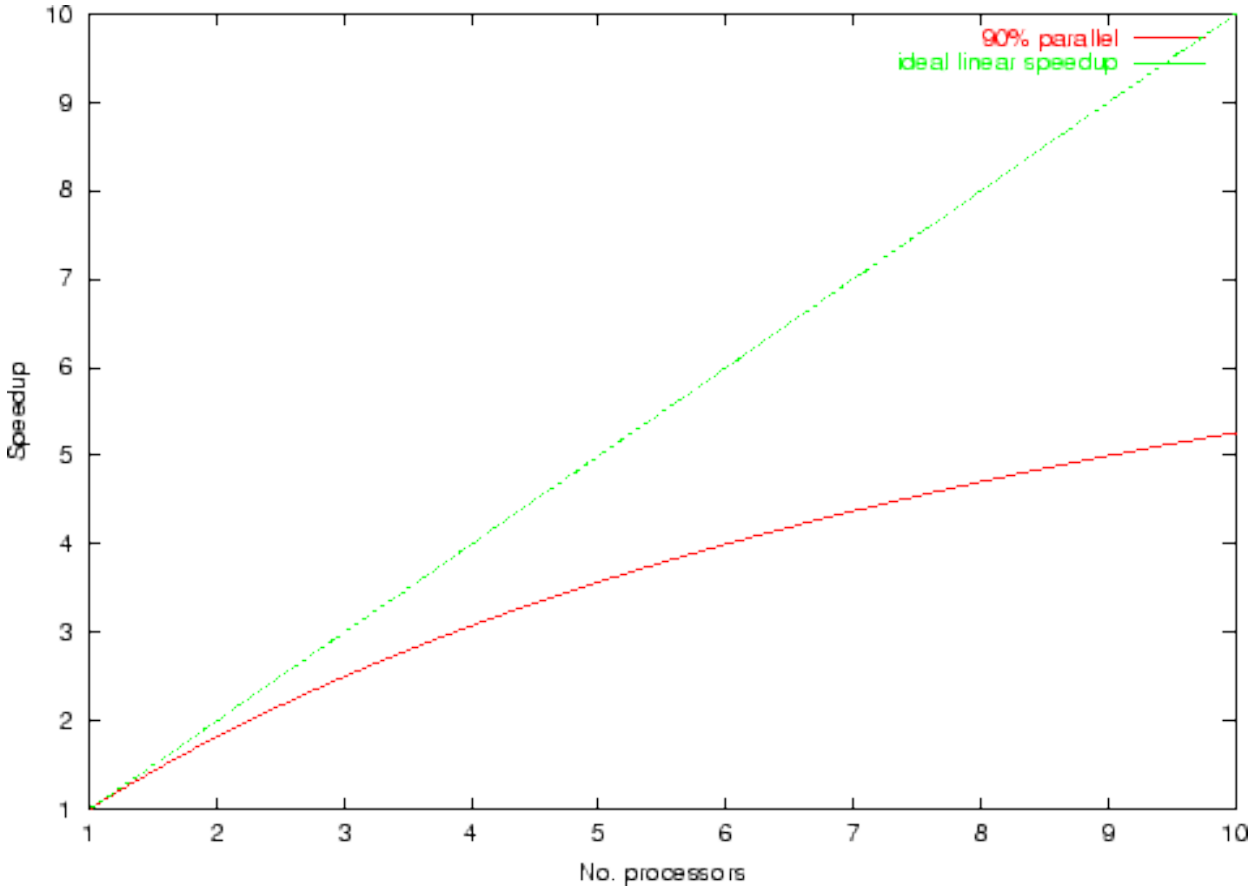
  Ideal: speedup (p) = p

  Ley de Amdahl: "el speedup teórico está limitado por la fracción secuencial s del programa no "paralelizable":

  speedup (p) <= 1 / (s + (1 - s) / p)
  <= 1 / s

Ley de Amdahl





Overhead del paralelismo

- **2. ¡No paralelizar por que sí!**

  Una vez encontrado suficiente trabajo para ejecutar en paralelo hay que averiguar si merece la pena que se ejecute en paralelo: hay que considerar el coste u overhead asociado a la ejecución en paralelo:

> Regla: sólo ejecutar en paralelo si el ahorro en tiempo es mayor que el overhead.

Overhead del paralelismo

- Ejemplos de fuentes de *overhead*:
  - Coste de arrancar un proceso o un *thread*.
  - Coste de comunicar información compartida.
  - Coste de sincronización.
  - Coste de la ejecución extra (y redundante)
  - ...
- El coste suele ser del orden de milisegundos.
- Cuestión muy difícil de analizar pues ¿depende completamente de en qué máquina se ejecute?.

NP-completo y (¡encima!) solución "no portable"

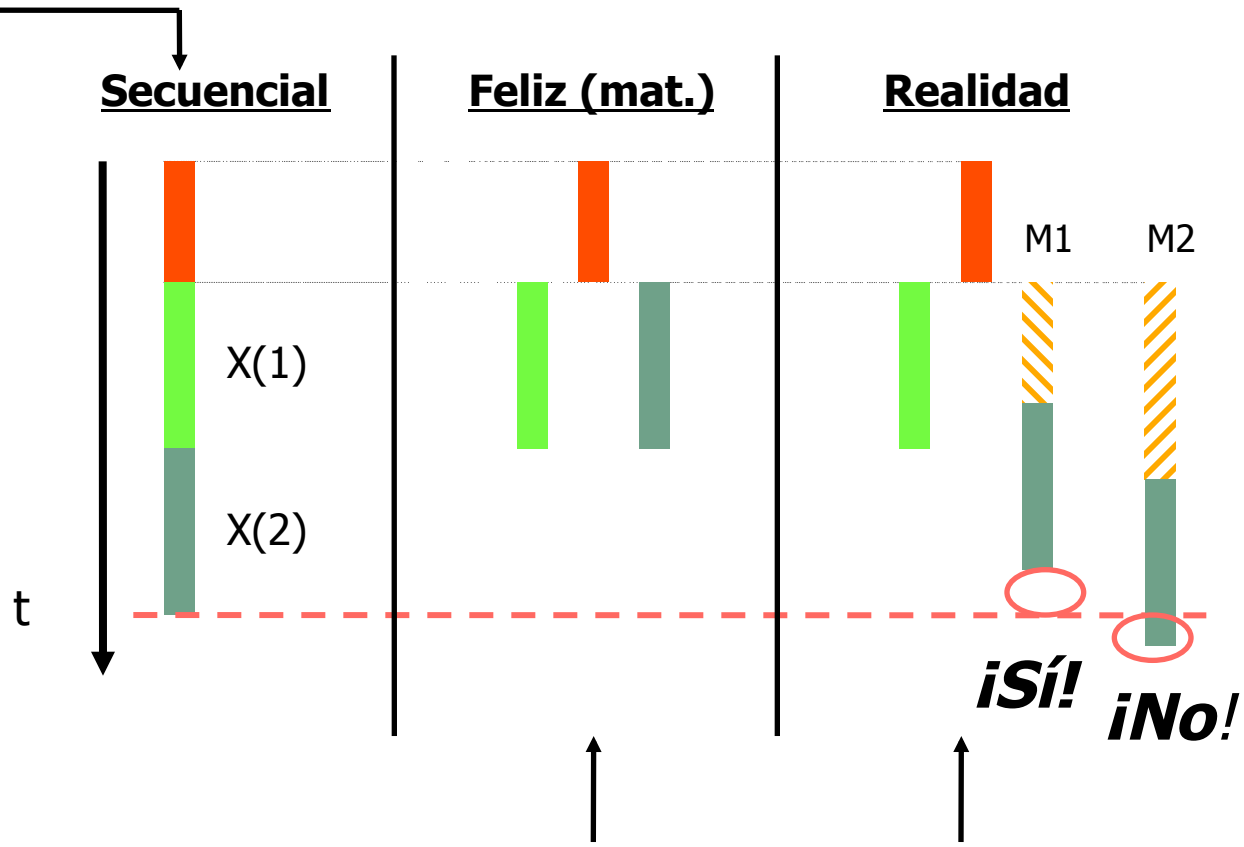
Overhead del paralelismo

- Ejemplo: ¿que sí, que no?

```
for i := 1, 2 loop
        x(i) := x(i) + 1
end loop
```

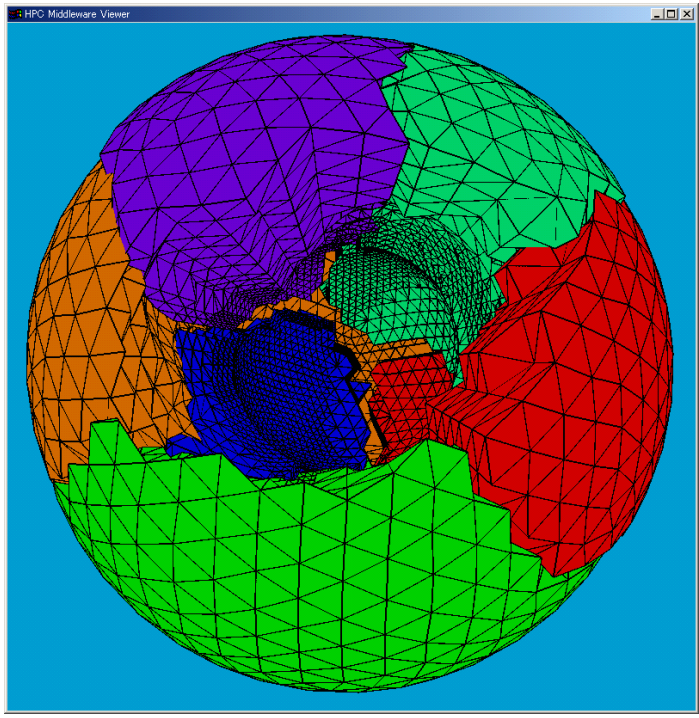
```
do all i = 1, 2
        x(i) := x(i) + 1
end do
```


Overhead



Proximidad y equilibrado

3. Proximidad (localidad) y equilibrado (balanceo) de carga

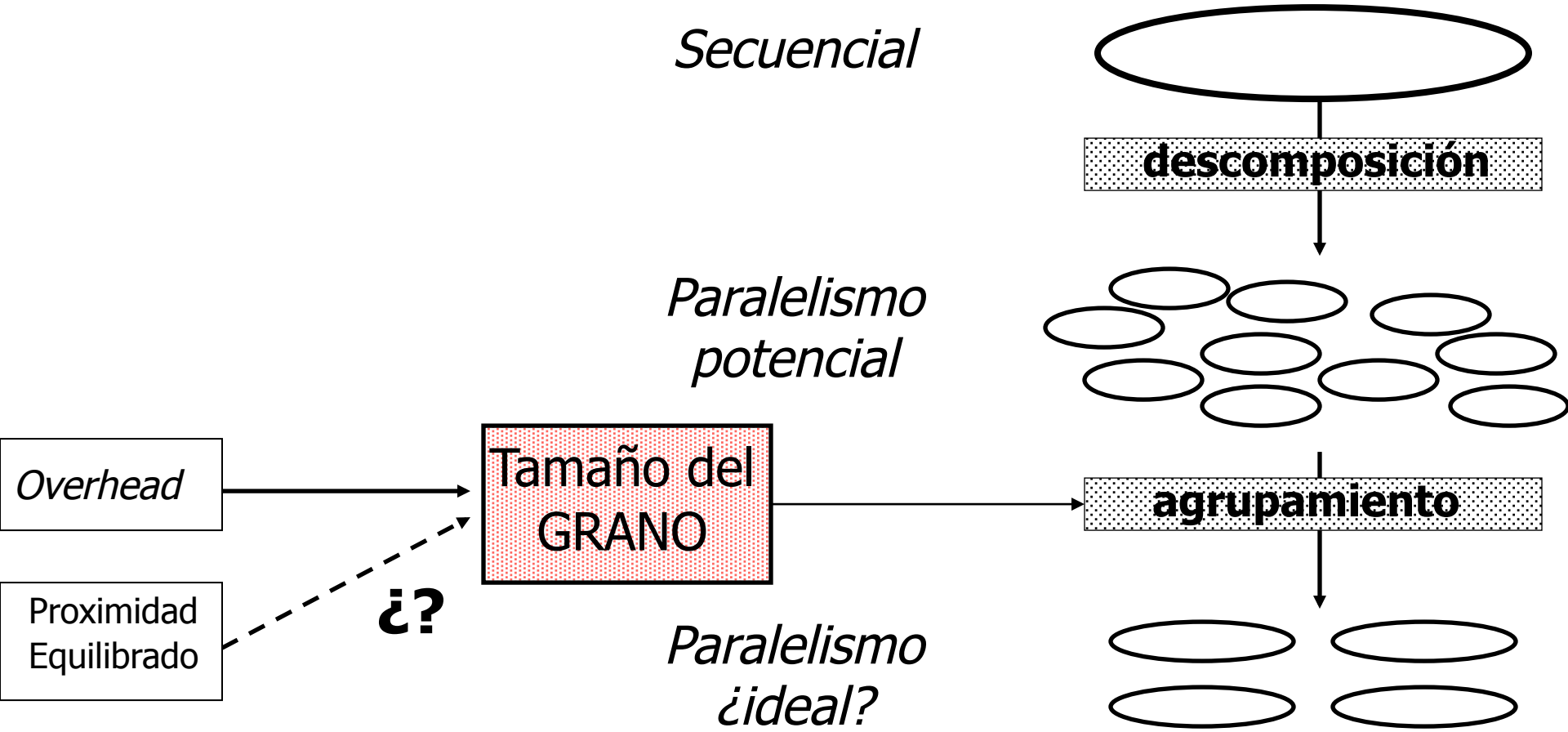


Proximidad: diferencia de tiempo significativa entre accesos remotos y locales: particularmente importante si la memoria está físicamente distribuida y en aplicaciones en las que se usa paralelismo orientado a datos.

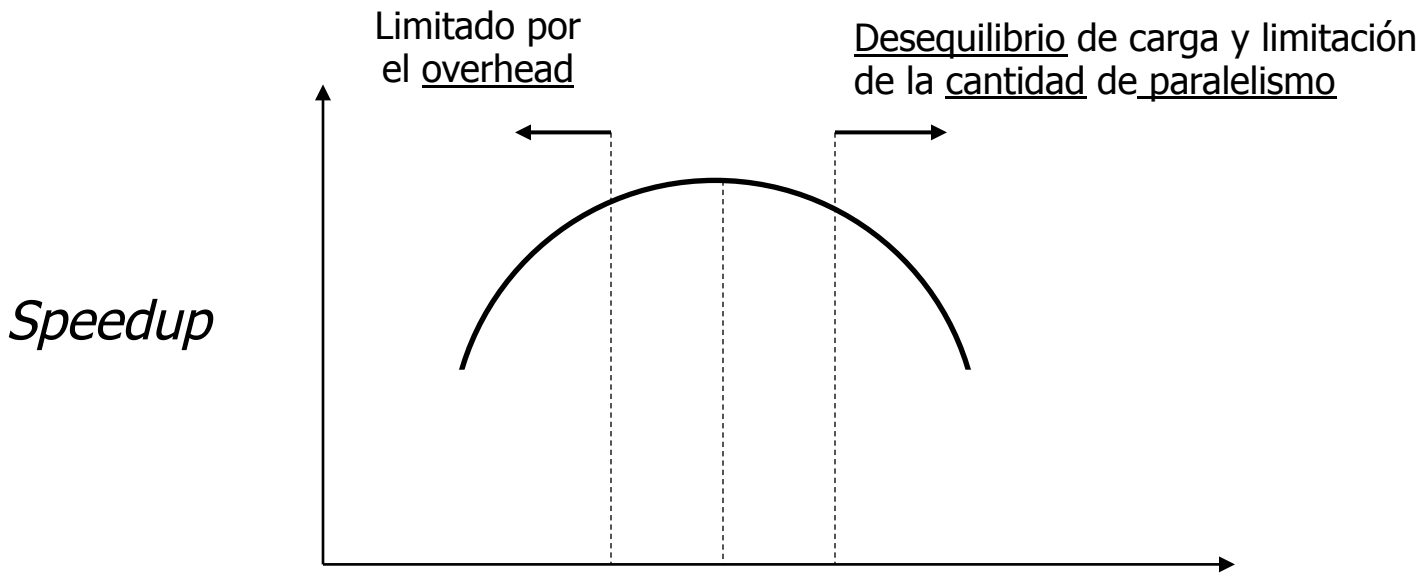
Balanceo: evitar que los procesadores estén ociosos pendiente.

- insuficiente cantidad de paralelismo.
- tareas de tamaño diferente.

Tamaño del 'grano'



Tamaño del 'grano'



- Cuestiones:
  - **¿convexidad de**
  - **¿hasta qué punto es importante el desequilibrio de la carga??**

Depuración de pp. paralelos

- 4. Depuración de programas paralelos: lógica y rendimiento

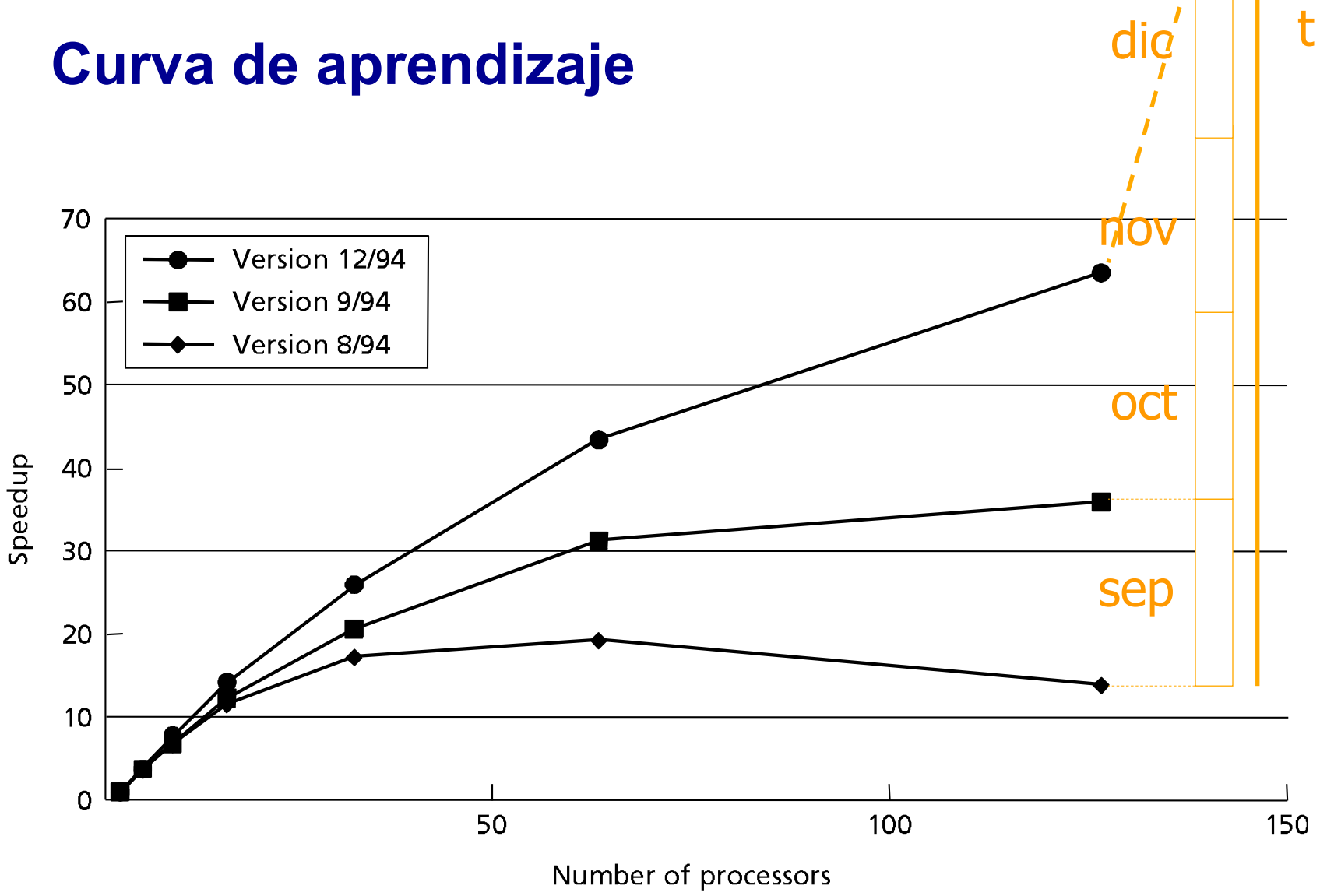
  Los programas paralelos son más difíciles de depurar que los secuenciales: en principio, "irrepetibilidad" de las ejecuciones.

  Doble depuración:

  - Lógica: el programa haga lo que queremos.
  - De **rendimiento**: haciendo lo que queremos, lo haga de la manera más eficiente.
  - Depuración lógica + depuración de rendimiento (logical and performance debugging): tema muy abierto todavía.

→ En la vida real, la depuración de rendimiento se hace fundamentalmente "**probando a ver**": versión/ejecución: "**curva de aprendizaje**": no modelos analíticos: si se usan, mucho, herramientas de visualización de ejecuciones

Curva de aprendizaje



Bibliografía

- ***Parallel Computer Architectures: a Hardware/Software Approach.*** *D.E. Culler, J.P. Singh, with A. Gupta*
  - Capítulo 1

- **TOP500 Supercomputer Sites**
  - http://www.top500.org/