Introducción a la Programación Paralela

Miguel Hermanns

Universidad Politécnica de Madrid, España

19 de abril de 2007

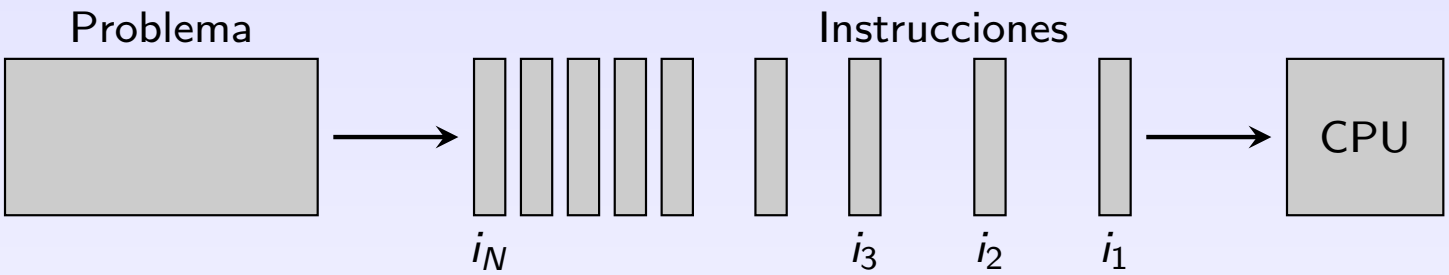
Estructura de la clase

1. Motivación de la programación paralela
2. Clasificación lógica del paralelismo
3. Clasificación física de ordenadores paralelos
4. Paradigmas de programación paralela
5. Conceptos generales y terminología habitual
6. Diseño de programas paralelos
7. Ejemplos de paralelización

¿Qué es el cálculo en paralelo?

Tradicionalmente los programas se han desarrollado para el **cálculo en serie**:

- Funcionan en un ordenador con una única CPU
- Un problema es dividido en un conjunto de instrucciones
- Las instrucciones son ejecutas secuencialmente
- Únicamente una instrucción es ejecutada cada vez



¿Qué es el cálculo en paralelo?

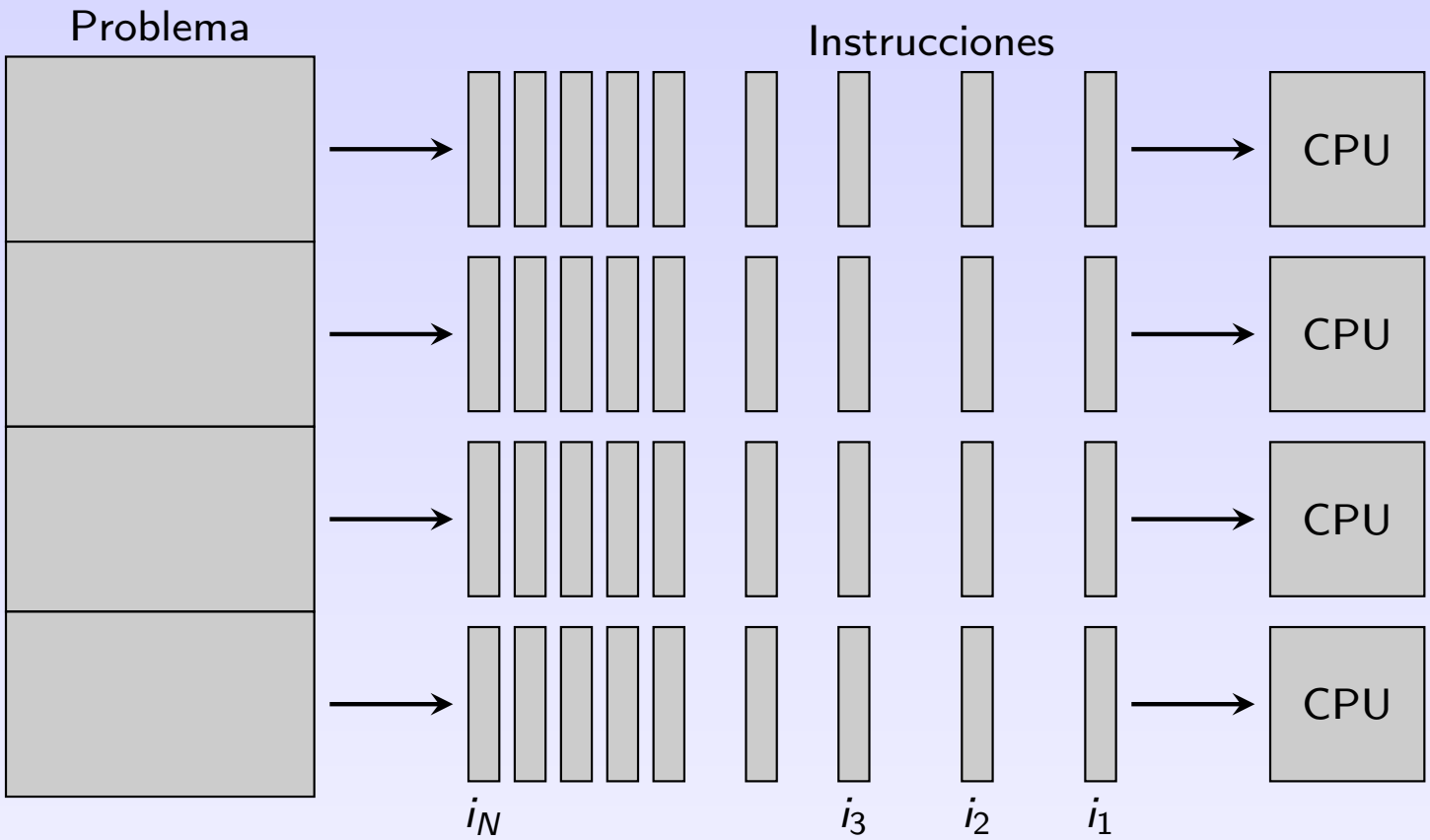
El **cálculo en paralelo** consiste en usar múltiples recursos simultáneamente para resolver un problema dado:

- Hace uso de un ordenador con varias CPUs
- El problema es dividido en partes *independientes*
- Cada parte es dividida en un conjunto de instrucciones
- Las instrucciones son ejecutas secuencialmente
- Las partes son resueltas simultáneamente

El cálculo en paralelo es la evolución natural del cálculo en serie

¿Qué es el cálculo en paralelo?

El **cálculo en paralelo** consiste en usar múltiples recursos simultáneamente para resolver un problema dado:



¿Por qué hacer cálculo en paralelo?

Los **motivos clásicos** más importantes son:

- Resultados en **menos tiempo** (*wall clock time*)
- Resolución de problemas **más grandes** y complejos
- Realización de barridos paramétricos
- Estudio de variaciones de un mismo problema

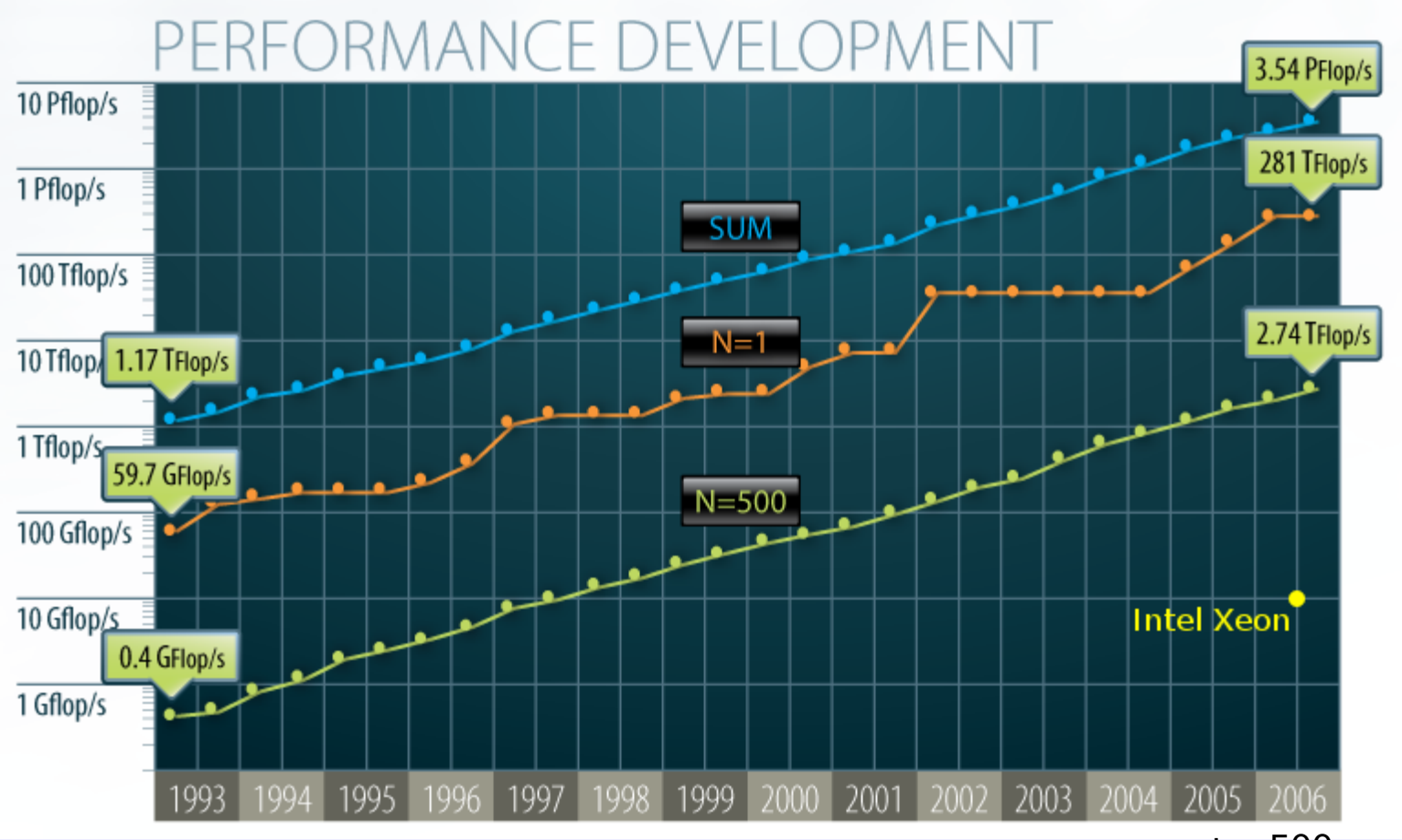
El **motivo actual** más importante es:

- Los procesadores actuales son paralelos: **n-core**

El paralelismo es el futuro de la computación

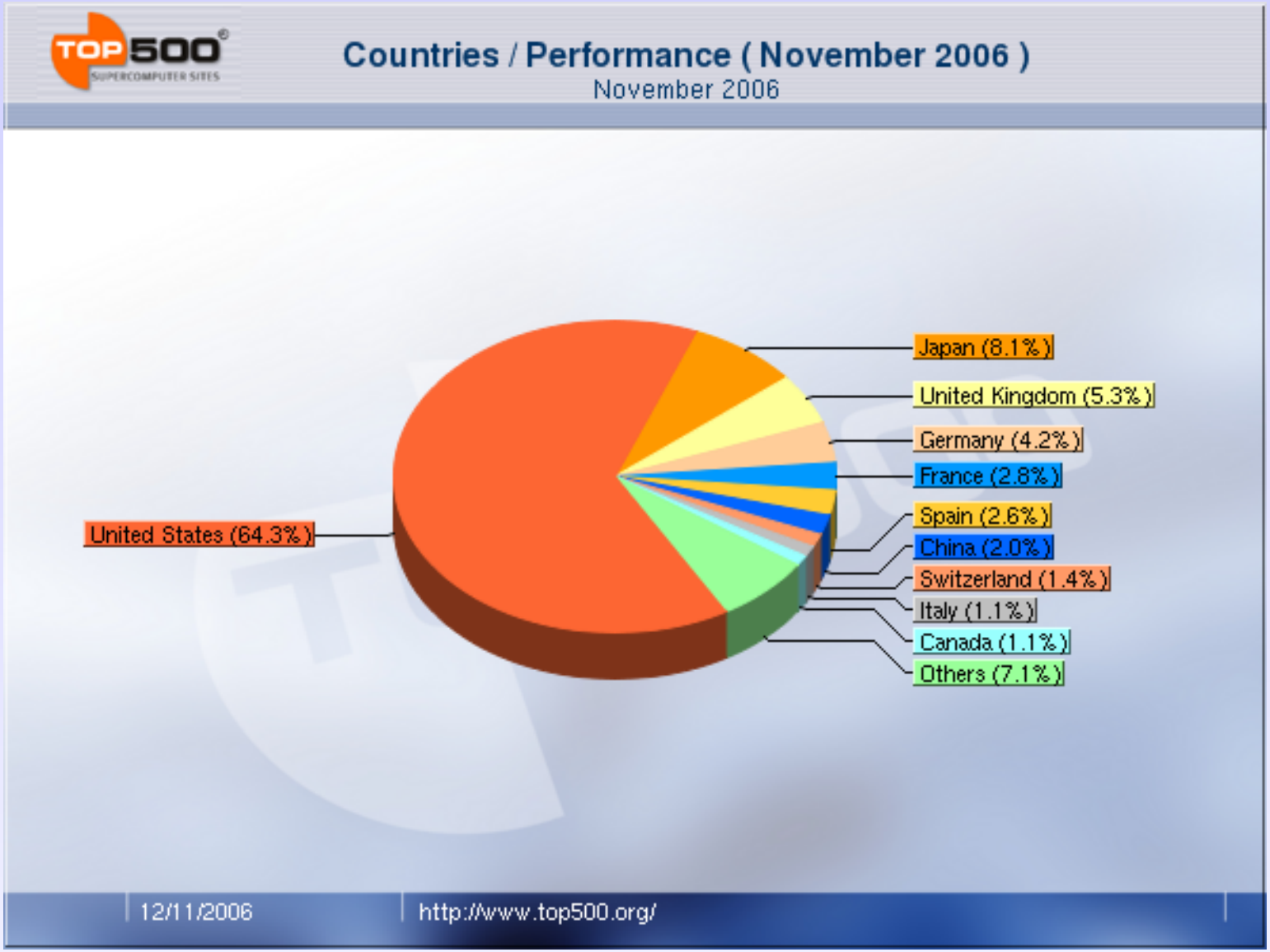
¿Por qué hacer cálculo en paralelo?

Evolución de los ordenadores más rápidos del Mundo:

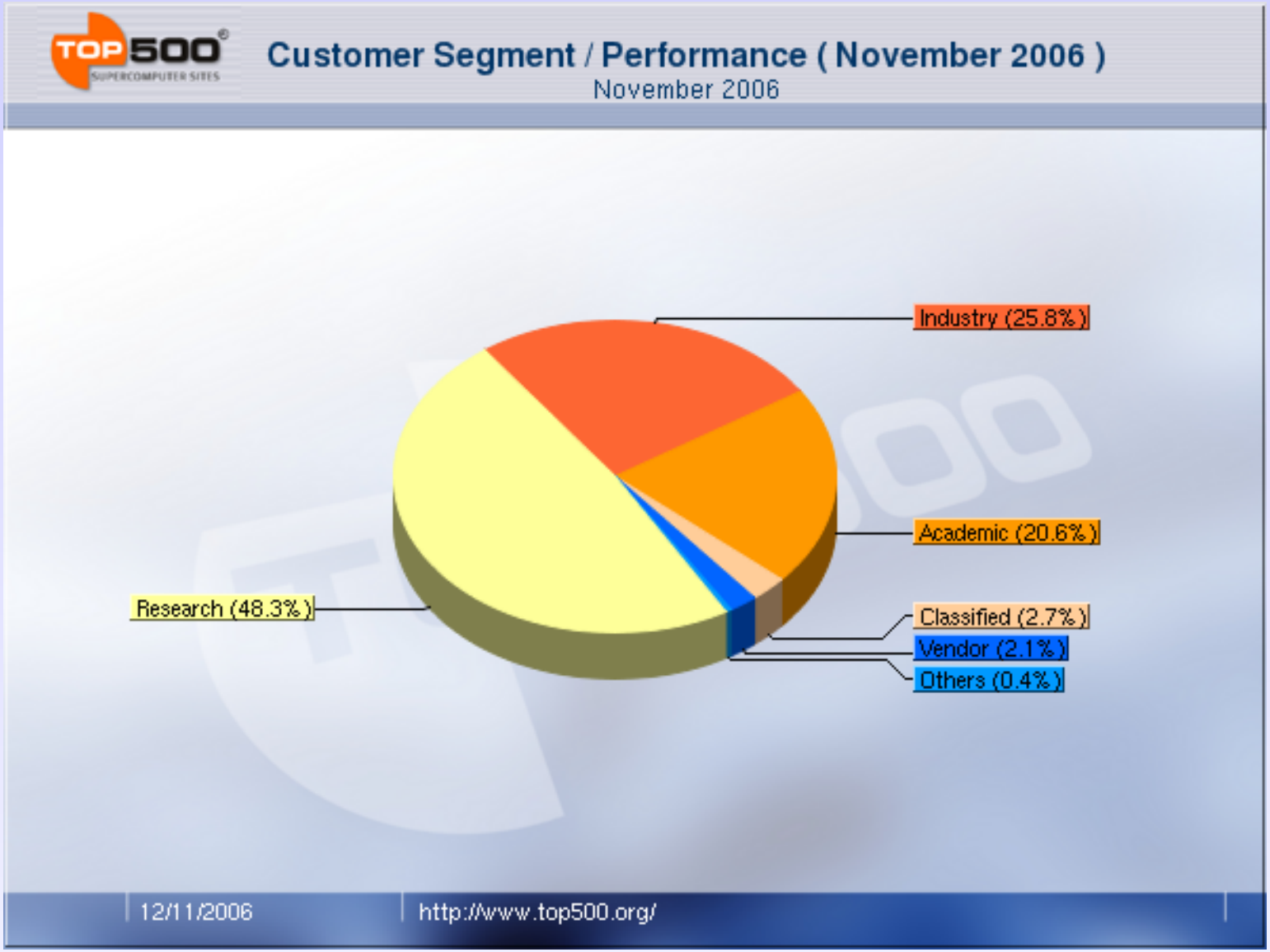


www.top500.org

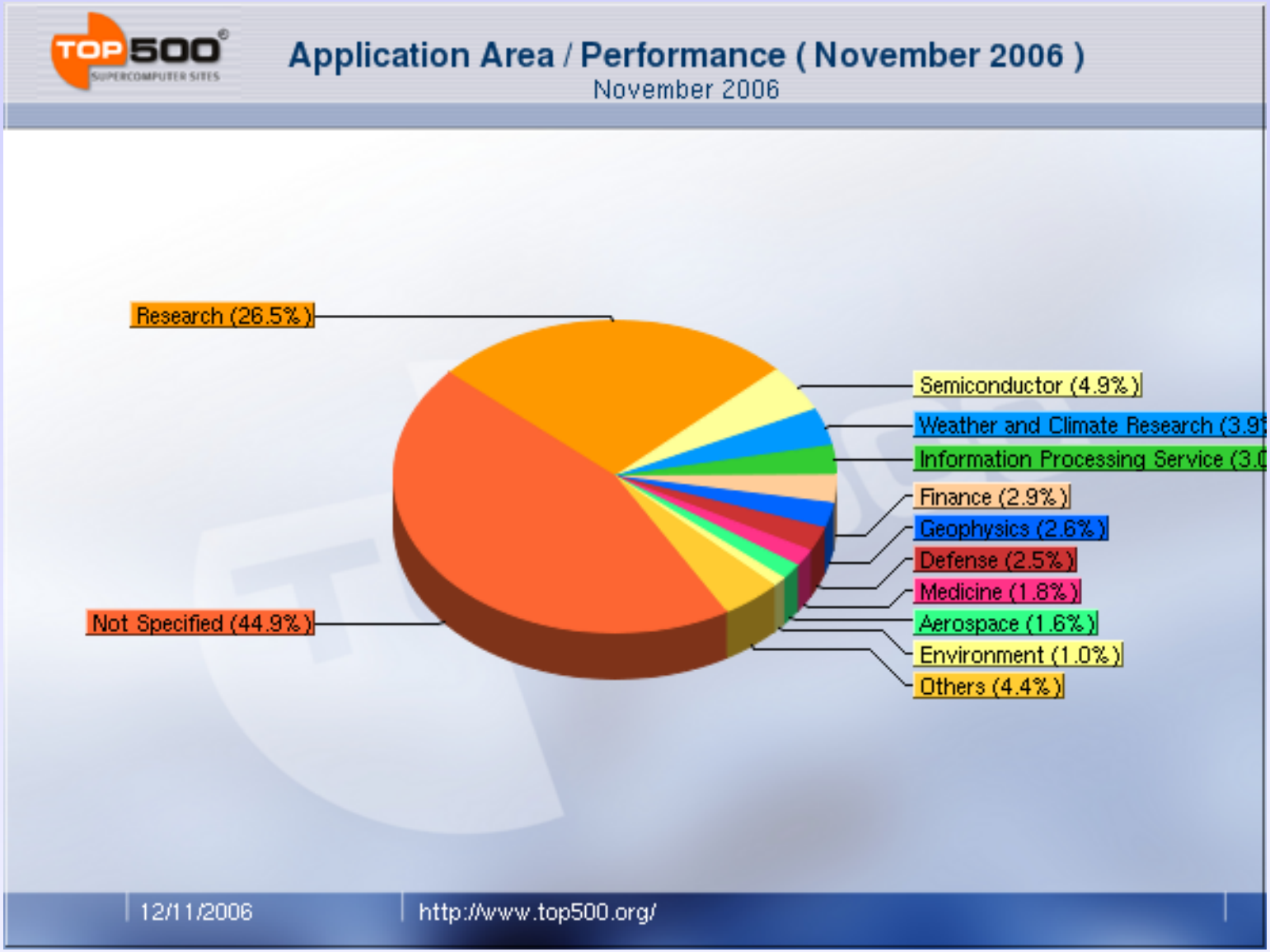
¿Dónde se realiza el cálculo en paralelo?



¿Quién programa en paralelo?



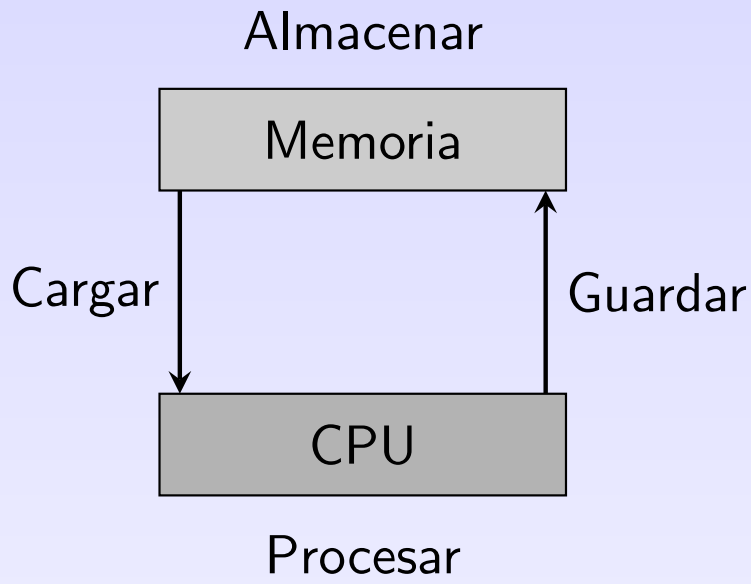
¿Quién programa en paralelo?



La arquitectura de von Neumann

Todos los ordenadores siguen el mismo patrón:

- La memoria almacena el programa y los datos
- El programa son de instrucciones a seguir por la CPU
- Los datos son información a utilizar por el programa

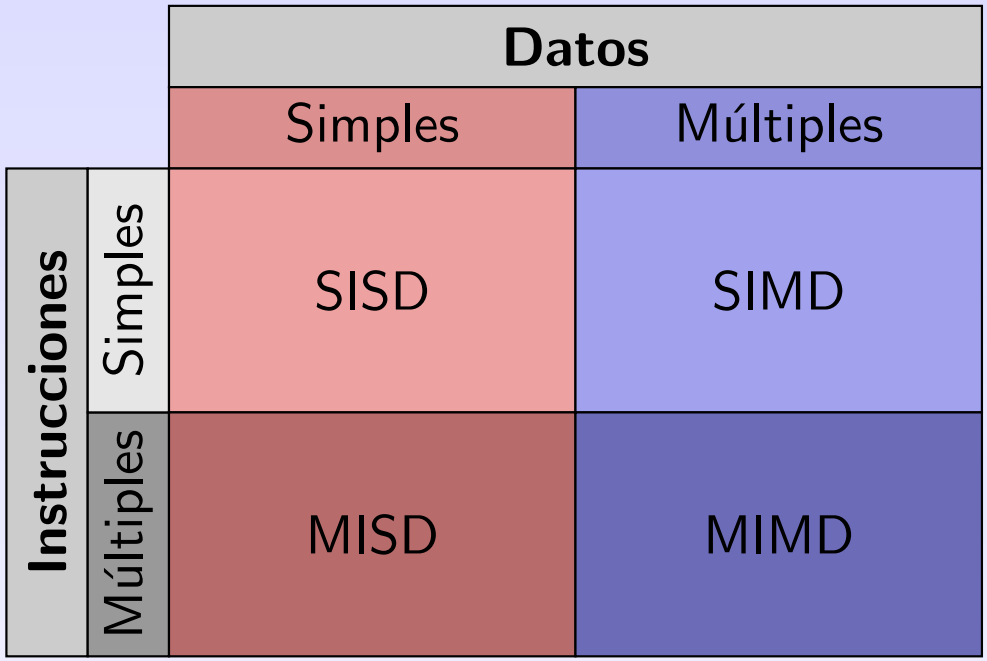


La CPU carga los datos, los procesa según el programa y luego los guarda otra vez en memoria

Taxonomía de Flynn

Es la **clasificación** más extendida del paralelismo:

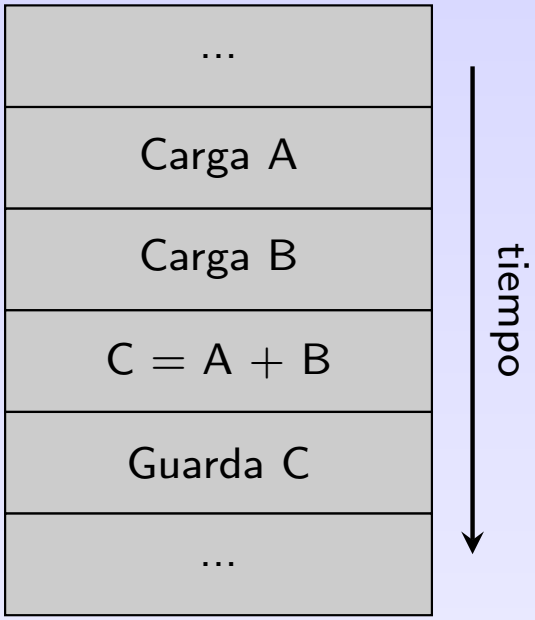
- Distingue entre instrucciones y datos
- Estos pueden ser simples o múltiples



SISD: Single Instruction, Single Data

Características del modelo SISD:

- La CPU procesa únicamente una instrucción por cada ciclo de reloj
- Únicamente un dato es procesado en cada ciclo de reloj
- Es el modelo más antiguo de ordenador y el más extendido
- Ejemplo: la mayoría de los PCs, servidores y estaciones de trabajo



SISD: Single Instruction, Single Data

Analogía con una imprenta:

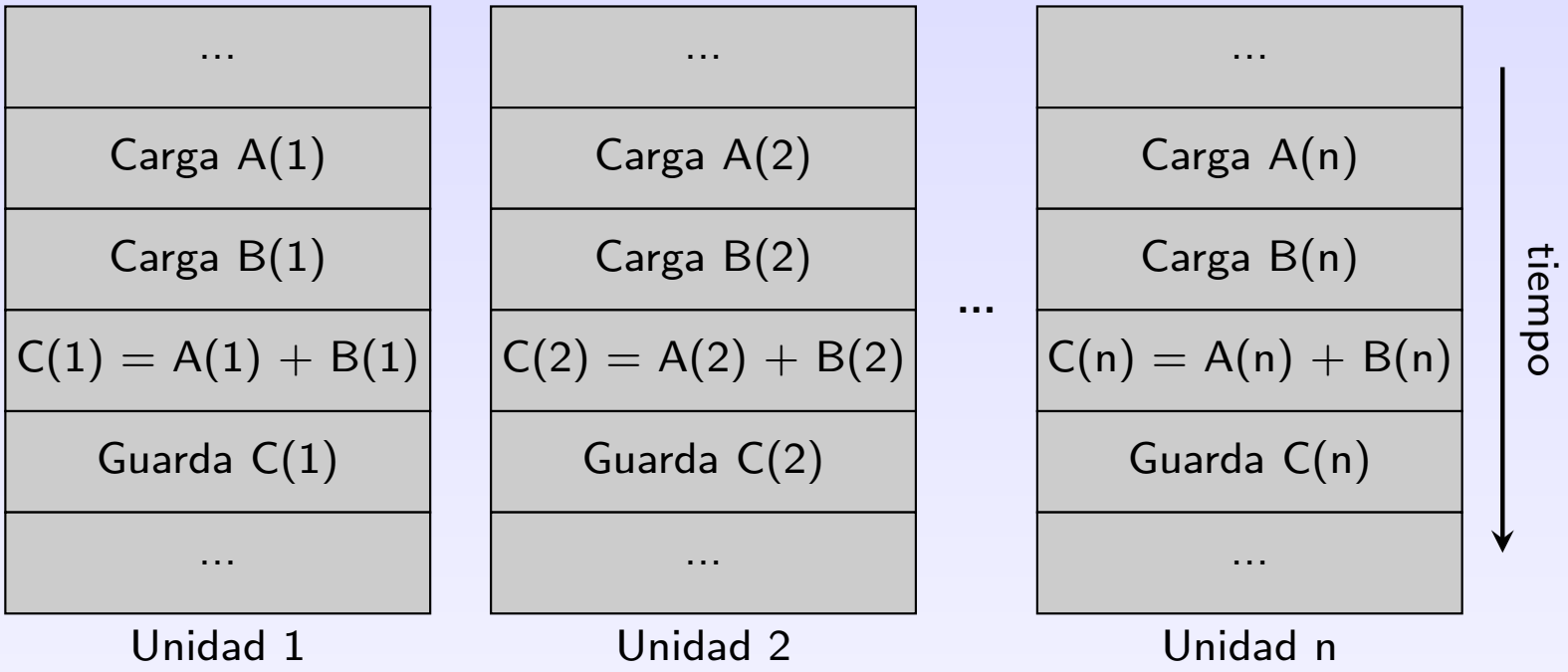
- El acto de imprimir son las instrucciones del programa
- La plantilla y el papel son los datos del programa

Cuanto más rápida la imprenta, más trabajo se realiza

SIMD: Single Instruction, Multiple Data

Características del modelo SIMD:

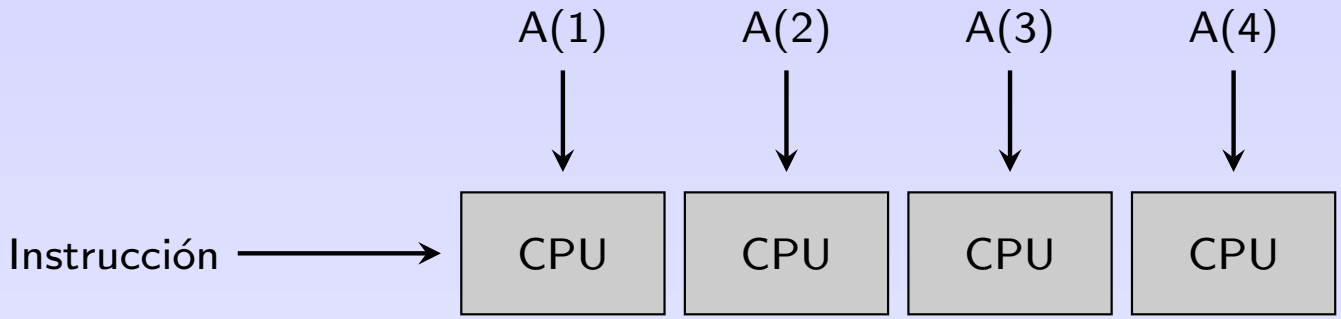
- Todas las unidades ejecutan la misma instrucción
- Cada unidad procesa un dato distinto
- Todas las unidades operan simultáneamente



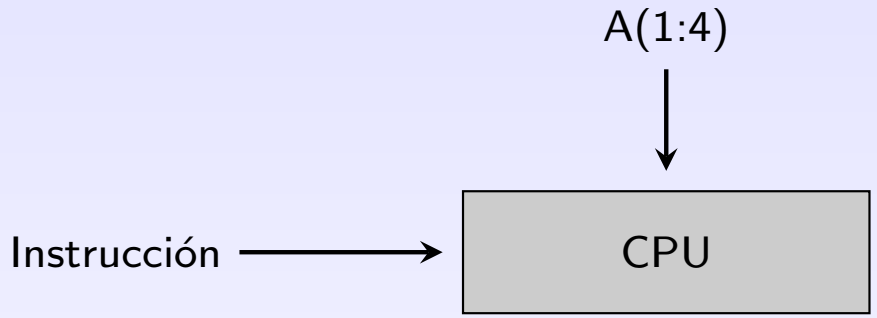
SIMD: Single Instruction, Multiple Data

Dos maneras de obtener SIMD:

- **Matrices de procesadores**:



- **Procesadores vectoriales**:



SIMD: Single Instruction, Multiple Data

Analogía con una imprenta:

- El acto de imprimir son las instrucciones del programa
- La plantilla y el papel son los datos del programa

La **matriz de imprentas** sería:

Cuantas más imprentas simultáneas,
más trabajo se realiza

SIMD: Single Instruction, Multiple Data

Analogía con una imprenta:

- El acto de imprimir son las instrucciones del programa
- La plantilla y el papel son los datos del programa

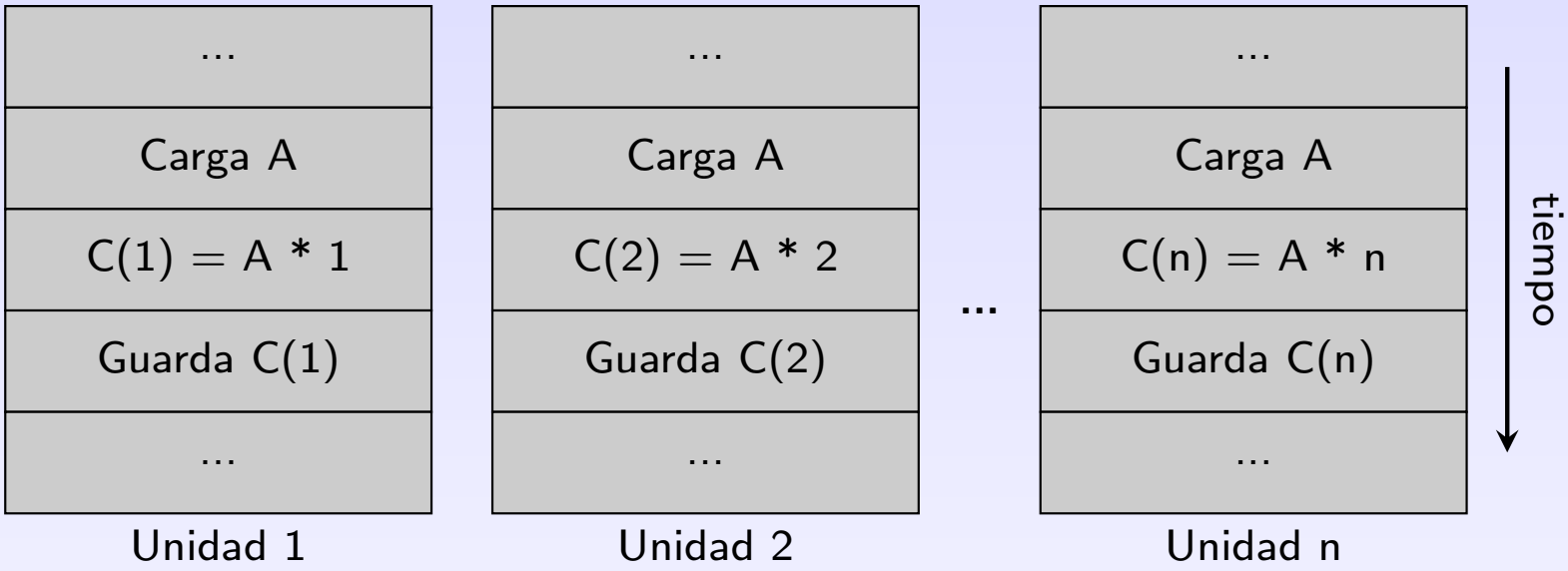
La **imprenta vectorial** sería:

Cuanto más vectorial sea la imprenta, más trabajo se realiza

MISD: Multiple Instruction, Single Data

Características del modelo MISD:

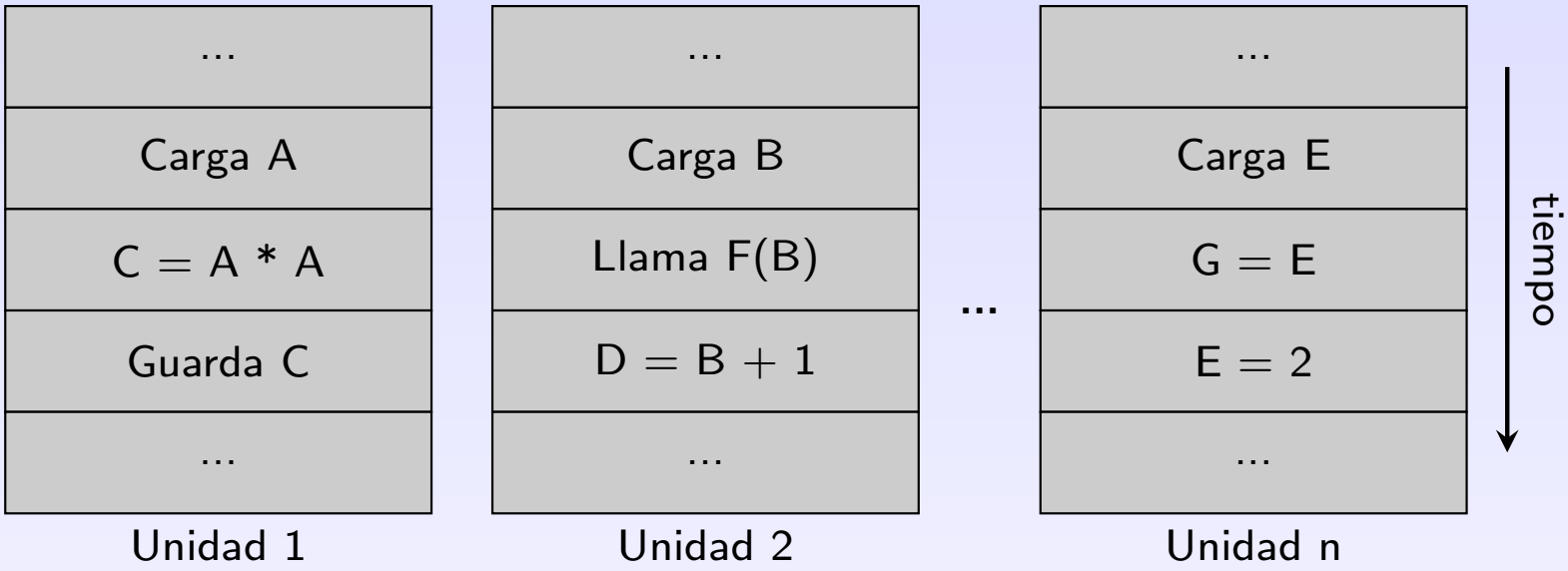
- Cada unidad ejecuta una instrucción distinta
- Cada unidad procesa el mismo dato
- Aplicación muy limitada en la vida real



MIMD: Multiple Instruction, Multiple Data

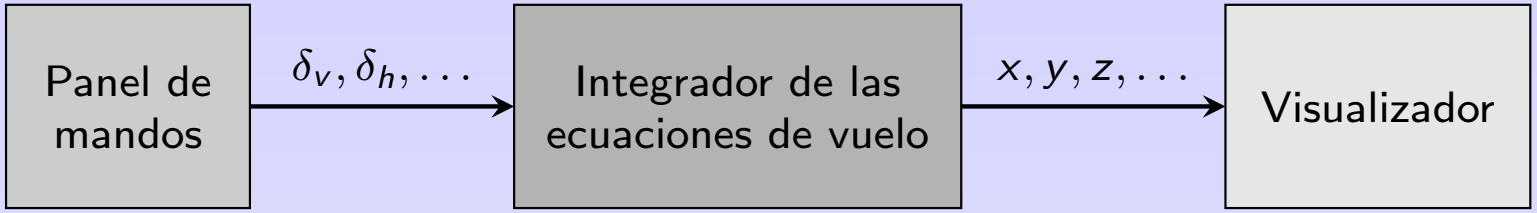
Características del modelo MIMD:

- Cada unidad ejecuta una instrucción distinta
- Cada unidad procesa un dato distinto
- Todas las unidades operan simultáneamente



MIMD: Multiple Instruction, Multiple Data

Analogía con un simulador de vuelo:



Clasificación de los ordenadores paralelos

Los ordenadores pueden clasificarse atendiendo a:

- El tipo de procesador que utilizan
- La manera de distribuir la memoria
- La arquitectura del ordenador

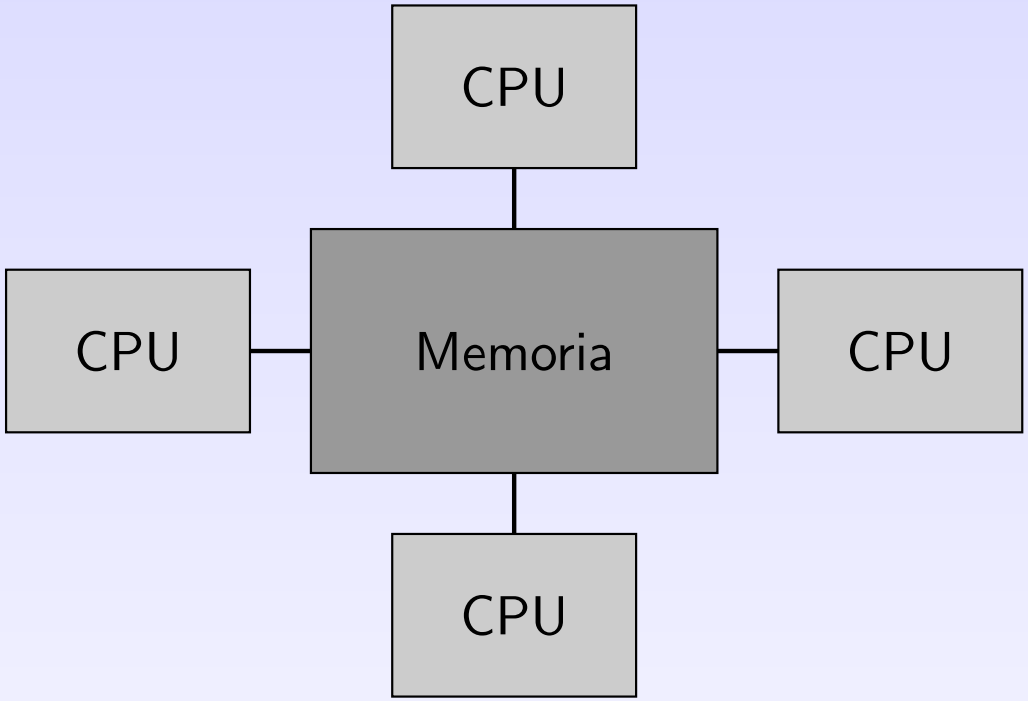
Lo más habitual es según la **distribución de memoria**:

- Ordenadores de memoria compartida
- Ordenadores de memoria distribuida
- Ordenadores híbridos

Ordenadores de memoria compartida

Características principales:

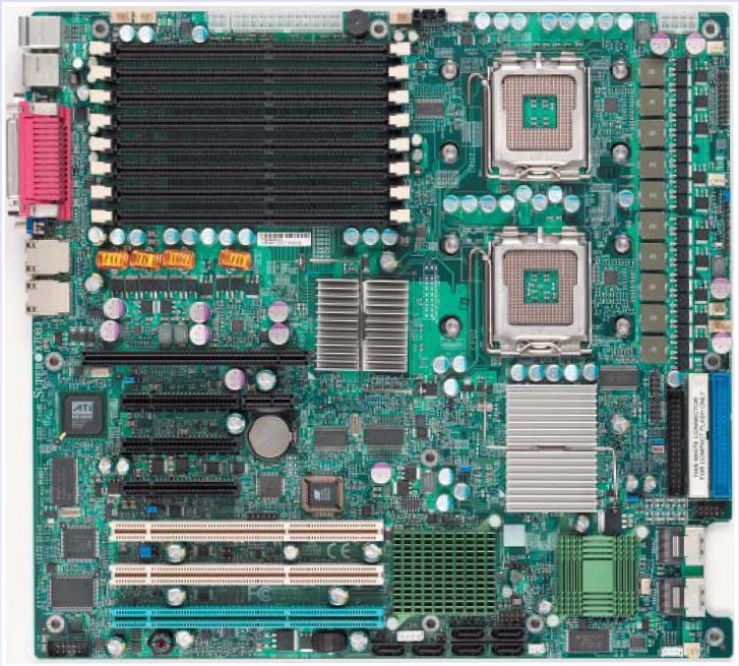
- Todas las CPUs acceden a la misma memoria
- Cambios en la memoria son visibles por todas las CPUs
- Hay principalmente dos tipos: UMA y NUMA



Ordenadores de memoria compartida

"Uniform Memory Access" (UMA):

- Todas las CPUs están igual de lejos de la memoria
- Máquinas SMP ("Symmetric Multiprocessor")
- Ejemplo típico actual: procesadores Intel Xeon

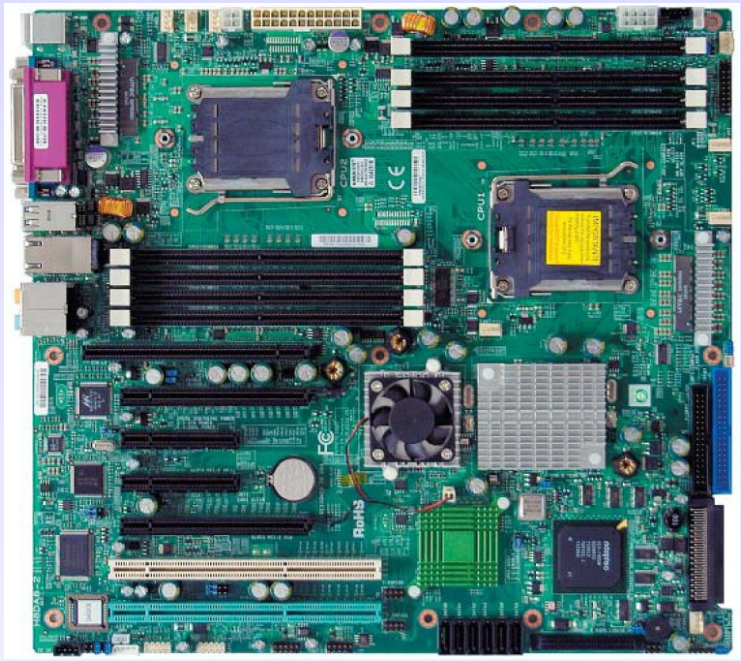


Placa base Supermicro X7DB3

Ordenadores de memoria compartida

"Non-Uniform Memory Access" (NUMA):

- Las CPUs no están todas igual de lejos de la memoria
- A menudo son máquinas SMP interconectadas
- Ejemplo típico actual: procesadores AMD Opteron



Placa base Supermicro H8DA8

Ordenadores de memoria compartida

IBM System p5:

- Arquitectura NUMA
- 64 CPUs con acceso a la misma memoria
- 2000 GB de memoria

Ordenadores de memoria compartida

Ventajas:

- Conceptualmente fáciles de programar
- Compartir datos entre *threads* es muy fácil y rápido

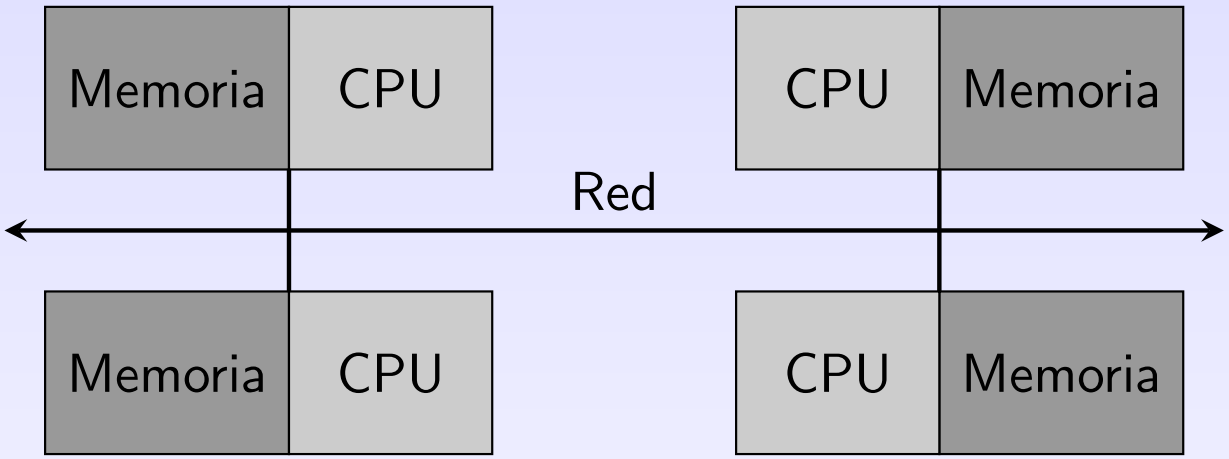
Desventajas:

- La escalabilidad entre CPUs y memoria es mala
- El programador es responsable de la sincronía
- Es muy costoso hacer ordenadores con muchas CPUs

Ordenadores de memoria distribuida

Características principales:

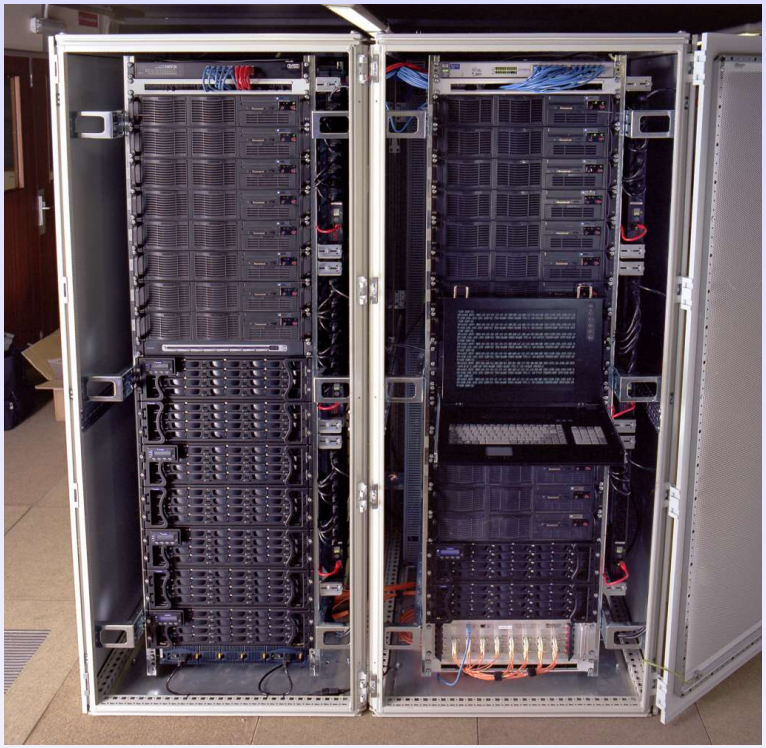
- Cada CPU tiene su propia memoria local
- La memoria local de una CPU no es visible por las demás CPUs
- Información es compartida entre CPUs por una red



Ordenadores de memoria distribuida

Clúster de ordenadores:

- Se compone de equipos informáticos éstandar
- Red de bajo o alto rendimiento
- Coste bajo para la potencia que se consigue



Ordenadores de memoria distribuida

Ventajas:

- La escalabilidad entre CPUs y memoria es muy buena
- Acceso rápido y en exclusiva a la memoria
- El coste es "lineal" con el número de CPUs

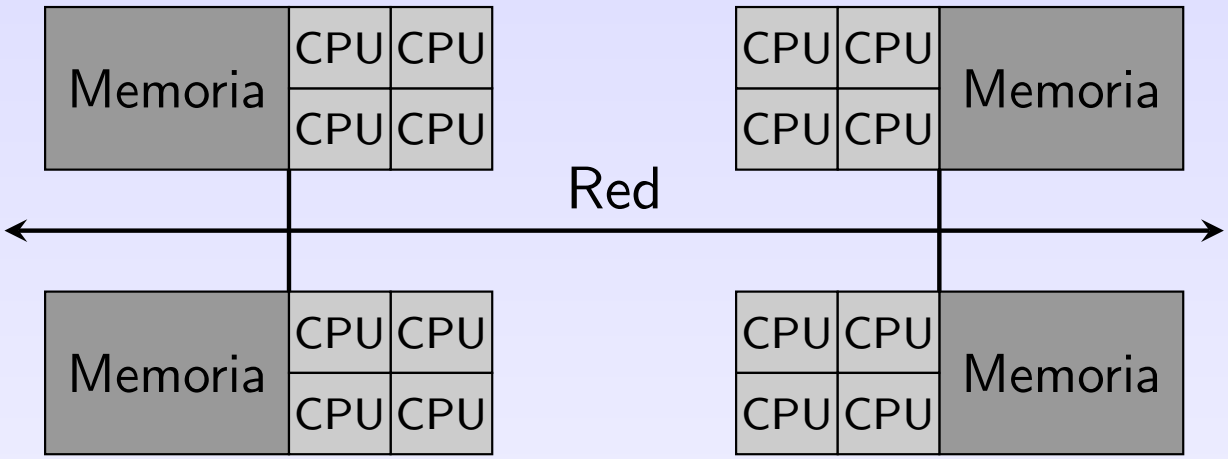
Desventajas:

- El programador es responsable de las comunicaciones
- La conversión de programas seriales puede no ser trivial
- La red de comunicaciones suele ser el cuello de botella

Ordenadores híbridos

Características principales:

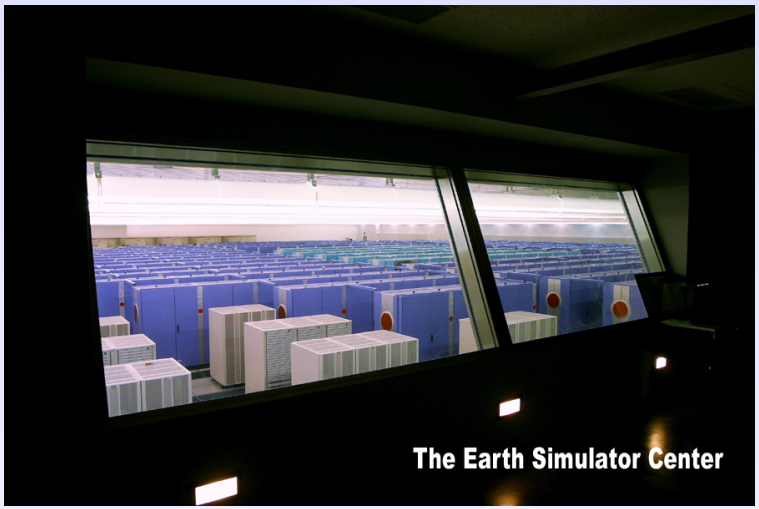
- Grupos de CPUs comparten una misma memoria
- Los grupos de CPUs se comunican por una red
- Suelen ser máquinas SMP interconectadas entre sí

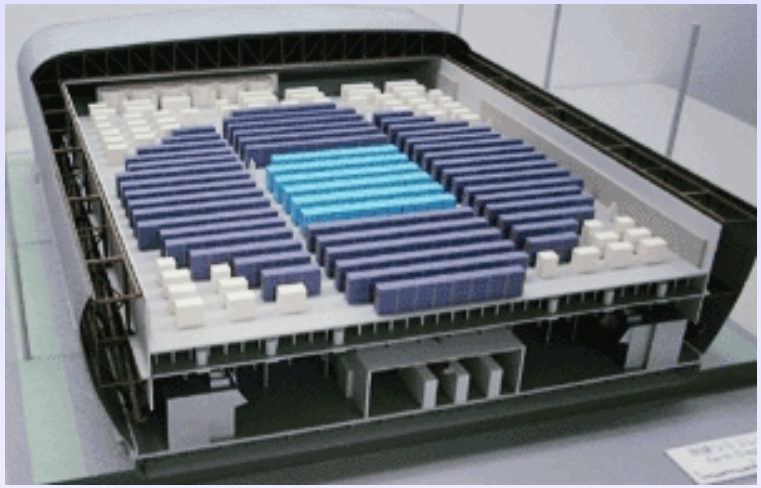


Ordenadores híbridos

NEC Earth Simulator (Japón):

- 640 nodos SMP con 8 CPUs por nodo
- 16GB de memoria por nodo
- Total: 5120 CPUs y 10240 GB de memoria

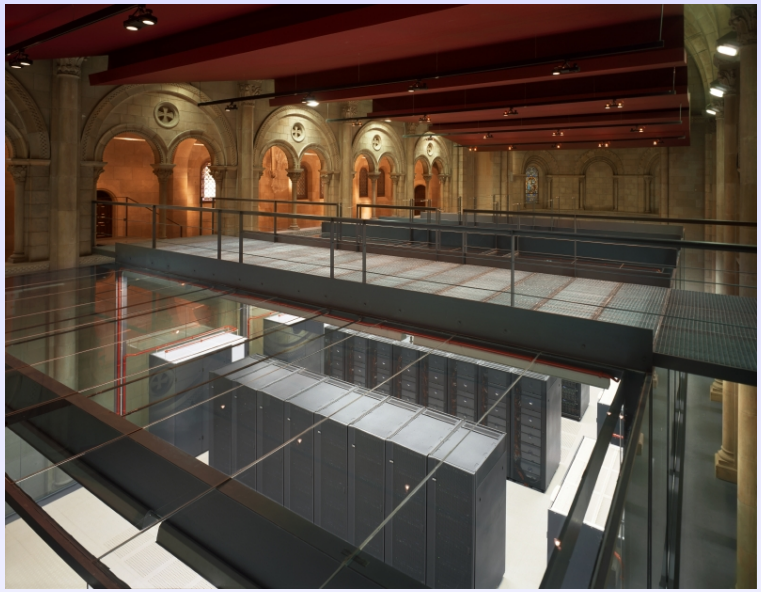




Ordenadores híbridos

Mare Nostrum (España):

- 5120 nodos SMP con 2 CPUs por nodo
- 4GB de memoria por nodo
- Total: 10240 CPUs y 20480 GB de memoria



Ordenadores híbridos

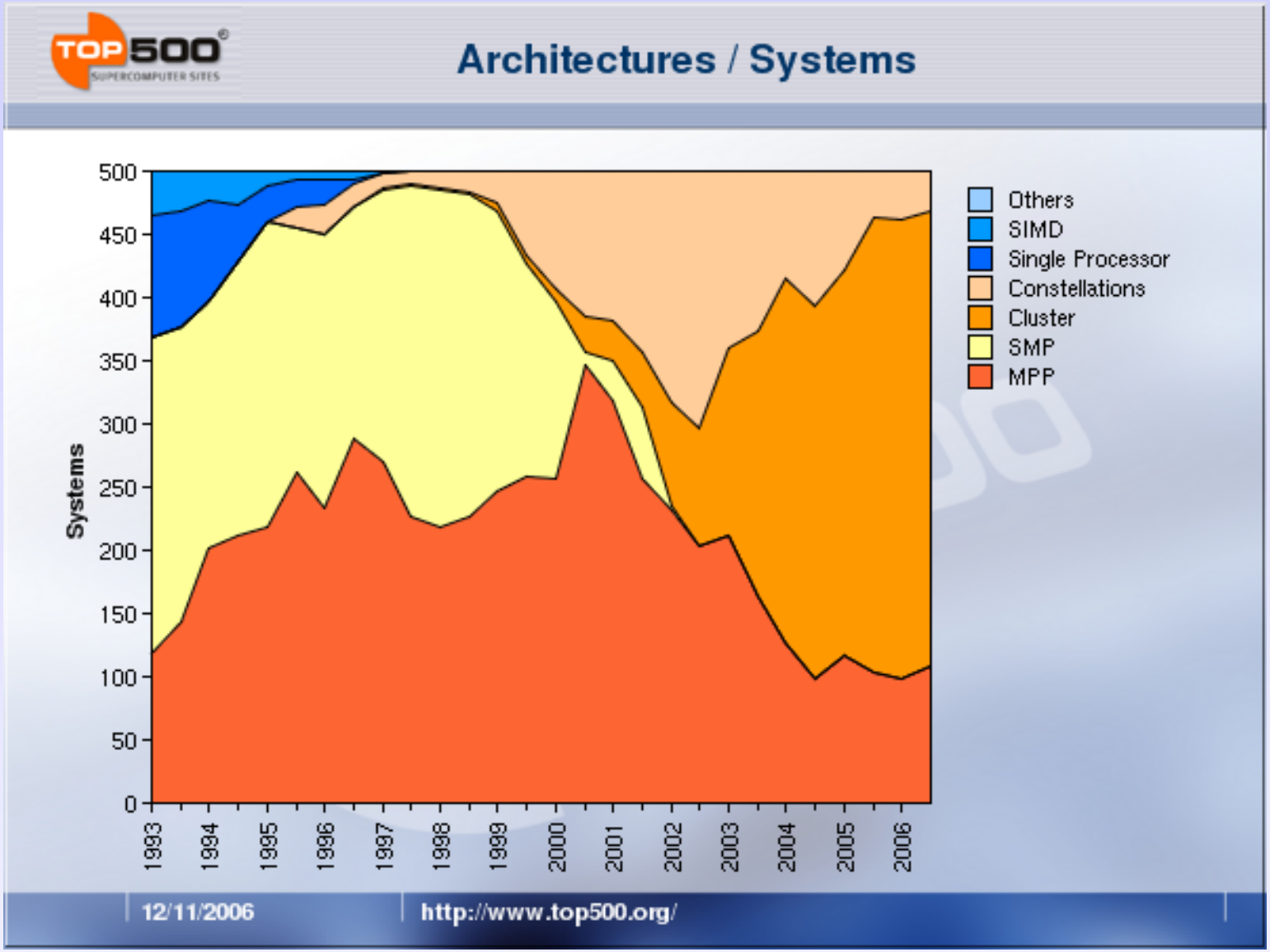
Ventajas:

- La escalabilidad entre CPUs y memoria es razonable
- El coste es "lineal" con el número de grupos de CPUs
- La red de comunicaciones es menos crítica

Desventajas:

- El programador es responsable de las comunicaciones
- El programador es responsable de la sincronía
- La conversión de programas seriales puede no ser trivial

Evolución histórica



Paradigmas de programación paralela

Existen varias formas de programar en paralelo:

- **Paso de mensajes**
- Memoria compartida
- **Tareas**
- Paralelismo de datos
- Operaciones remotas en memoria
- Modelos combinados

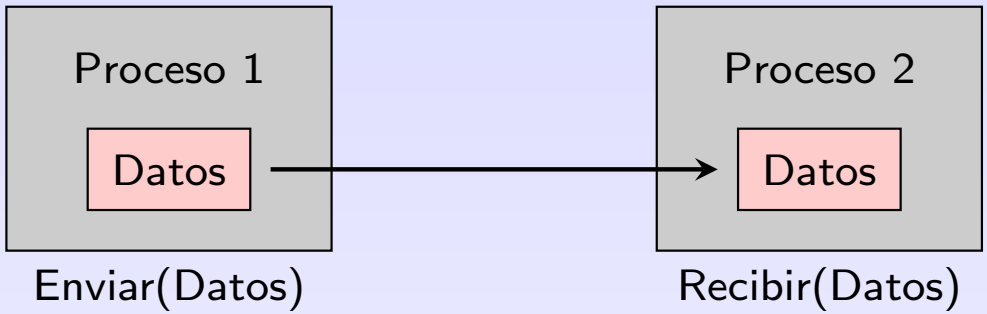
Todas estas formas:

- **NO** son excluyentes entre sí
- **NO** tienen nada que ver con el ordenador

Modelo de paso de mensajes

Características del paradigma de programación:

- Un conjunto de procesos que disponen de memoria local
- Los procesos intercambian datos mediante mensajes
- El emisor y receptor del mensaje tienen que colaborar



El **programador es responsable** del envío y la recepción de mensajes, que típicamente se hacen mediante llamadas a una librería

Modelo de paso de mensajes

La librería **Message Passing Interface (MPI)**:

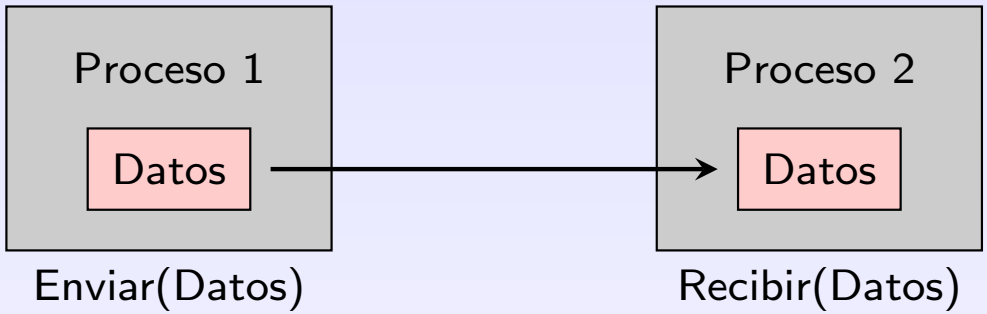
- Su funcionalidad es definida por el *MPI Forum*
- En 1994 se publicó la versión 1
- En 1996 se publicó la versión 2
- Es hoy por hoy la librería estándar
- Todas las plataformas paralelas la tienen

www.mpi-forum.org

Modelo de paso de mensajes

Ejemplo de programa con paso de mensajes:

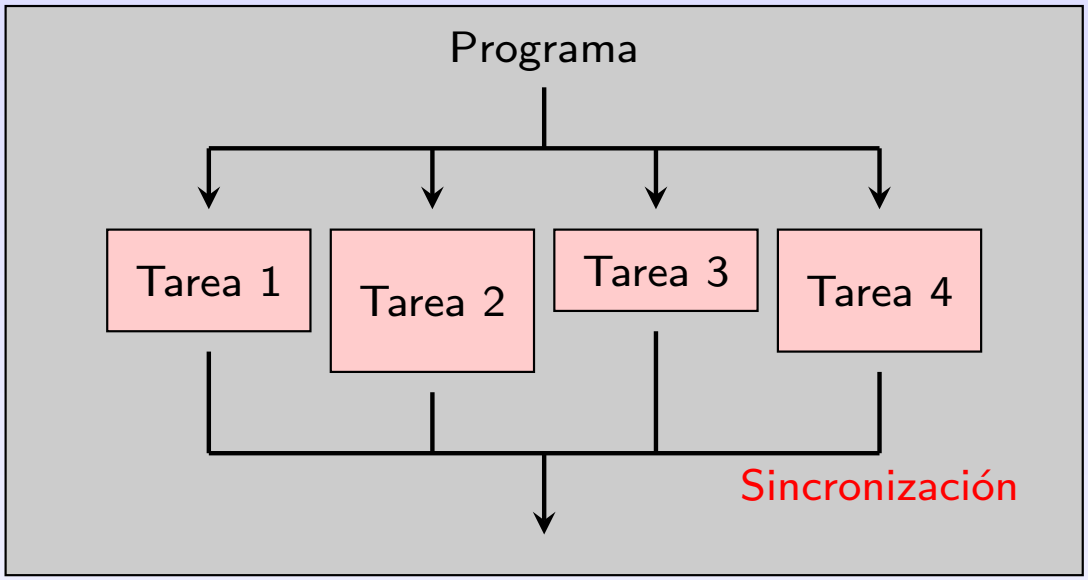
```
ID = Quien_soy_yo

Si soy ID = 1 entonces
  envio Datos a 2 y espero confirmacion

Si soy ID = 2 entonces
  recibo Datos de 1 y envio confirmacion
```



Modelo de tareas

Características del paradigma de programación:

- Un programa serial define un conjunto de tareas
- Cada tarea dispone de su memoria local
- Cada tarea tiene acceso a una memoria conjunta
- Las tareas son realizadas simultáneamente



El **programador es responsable** de la sincronización

Modelo de tareas

El estándar **OpenMP**:

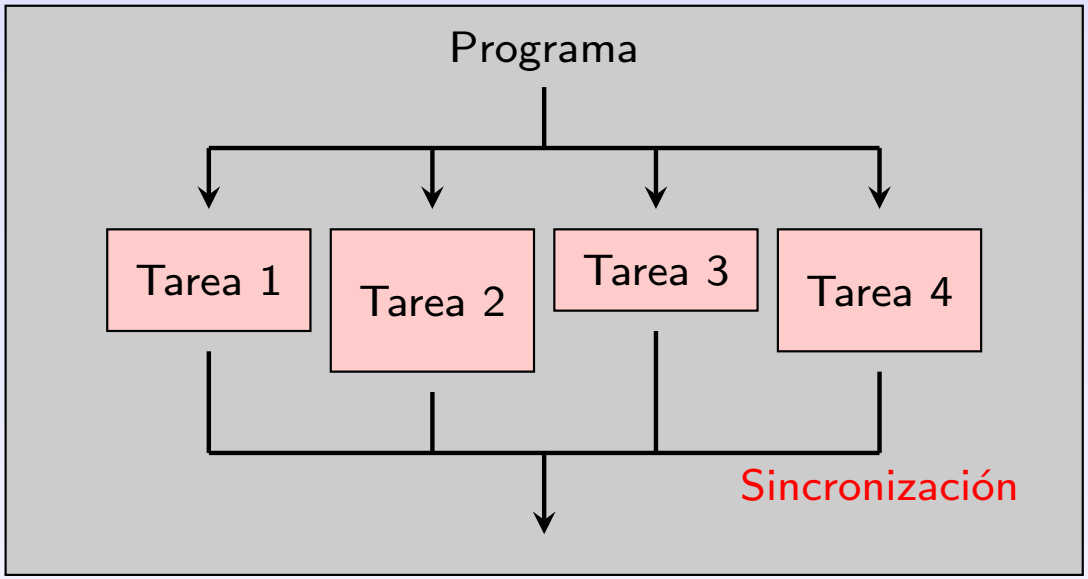
- Lo define el *OpenMP Architecture Review Board*
- En 1997 se publicó la versión 1.0
- En 2000 se publicó la versión 2.0
- En 2005 se publicó la versión 2.5
- Lo incorporan la mayoría de compiladores actuales

www.openmp.org

Modelo de tareas

Ejemplo de programa con tareas:

```
Hacer en paralelo el bucle con i = 1 hasta 25
  La tarea 1 es i =  1 hasta  5
  La tarea 2 es i =  6 hasta 13
  La tarea 3 es i = 14 hasta 18
  La tarea 4 es i = 19 hasta 25

Haced las tareas y os espero
```



Conceptos generales y terminología habitual

- **Ejecución serial**: las tareas/instrucciones de un programa son ejecutadas de manera secuencial, una cada vez
- **Ejecución paralela**: varias tareas/instrucciones de un programa son ejecutadas de manera simultánea
- **Memoria compartida**: las diferentes unidades de cálculo (CPU) comparten una memoria común a la cual tienen todos acceso en igualdad de condiciones
- **Memoria distribuida**: las diferentes unidades de cálculo (CPU) tienen una memoria propia a la cual las demás CPUs no tienen acceso directo

Conceptos generales y terminología habitual

- **Speedup**: la aceleración experimentada por un programa al hacer uso de N unidades de cálculo (CPU) en vez de una única:

$$\text{Speedup} = \frac{t_{\text{serie}}}{t_{\text{paralelo}}}$$

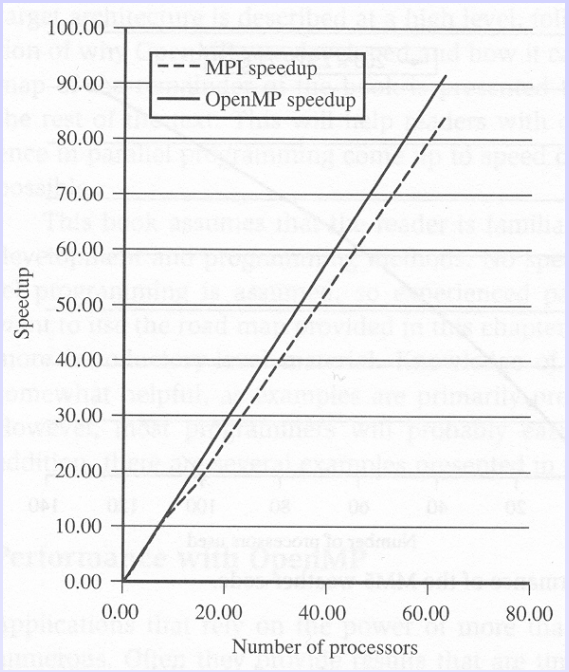
- **Eficiencia paralela**: es la aceleración alcanzada por un programa comparada con la que podría alcanzar en el caso ideal:

$$\text{Eficiencia paralela} = \frac{\text{Speedup}}{N}$$

- **Escalabilidad**: evolución del speedup de un programa con el número de unidades de cálculo (CPU)

Conceptos generales y terminología habitual

- **speedup superlineal**: habilidad de un programa de tener una eficiencia paralela mayor que la unidad:



- **Wall clock time**: tiempo físico t requerido por un programa para completar su ejecución

- **Tiempo de CPU**: tiempo lógico $t \times N$ requerido por un programa para completar su ejecución

Conceptos generales y terminología habitual

- **Proceso**: copia de un programa que se comunica con otros procesos/copias mediante pasos de mensajes
- **Tarea**: parte de un programa que se comunica con otras partes/tareas a través de una memoria compartida
- **Sincronización**: acción de poner en sincronía a dos o más tareas/procesos ejecutadas en paralelo
- **Thread safe**: una librería es *thread safe* cuando es capaz de ser utilizada de manera simultánea e independiente por varias tareas ejecutadas en paralelo

Conceptos generales y terminología habitual

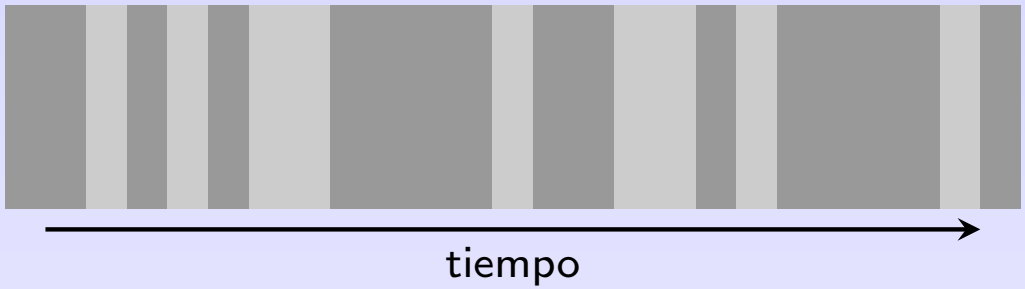
- **Parallel overhead**: sobrecoste debido a la ejecución paralela de un programa, en especial la creación y destrucción de tareas
- **Procesamiento paralelo masivo**: uso de un elevado número de procesadores para la resolución de un problema, hoy en día más de 1000
- **Embarazosamente paralelo**: programa/problema que no requiere de comunicaciones ni sincronismo para su ejecución/resolución:

Conceptos generales y terminología habitual

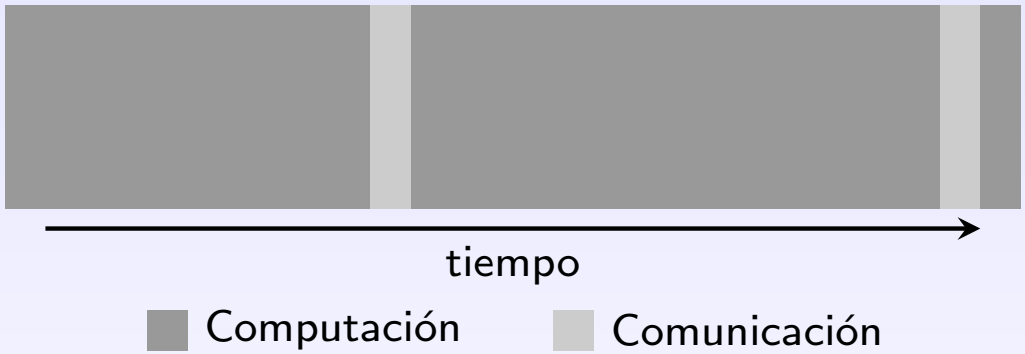
- **Comunicaciones**: acción de enviar de datos o instrucciones entre procesos pertenecientes a un mismo programa
- **Latencia**: el tiempo requerido por la red de comunicaciones en inicializar el envío de datos
- **Ancho de banda**: cantidad de datos que se pueden enviar en un segundo a través de la red de comunicaciones

Conceptos generales y terminología habitual

- **Granularidad**: medida cualitativa del cociente entre en tiempo dedicado a la computación y el tiempo dedicado a la comunicación:
  - **Granularidad fina**: se hace relativamente poca computación entre comunicaciones sucesivas:



  - **Granularidad gruesa**: se hace mucha computación entre comunicaciones sucesivas:



Límites a la paralelización: ley de Amdahl

Ley de Amdahl:

$$\textbf{Speedup teórico} = \frac{1}{(1-P) + P/N}$$

donde:

- P: fracción de código que es paralelizable
- N: número de procesadores a usar

La parte no paralelizable **limita la escalabilidad**:

N:P	0.50	0.90	0.99
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1000	1.99	9.91	90.99
10000	1.99	9.91	99.02
∞	2.00	10.00	100.00

Límites a la paralelización: ley de Amdahl

Muchos problemas **mejoran su escalabilidad** con el tamaño:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

- Resolución con Δx_1 y Δt_1:

Cálculos numéricos	85s	85.00 %
Fracción serial	15s	15.00 %

- Resolución con $\Delta x_2 = \Delta x_1/2$ y $\Delta t_2 = \Delta t_1/4$:

Cálculos numéricos	680s	97.84 %
Fracción serial	15s	2.16 %

Problemas en los que P crece con N son más **escalables**

Límites a la paralelización: las comunicaciones

La comunicación entre procesos lleva su tiempo:

$$t_{\text{com}} = \text{L} + \frac{\text{M}}{\text{B}}$$

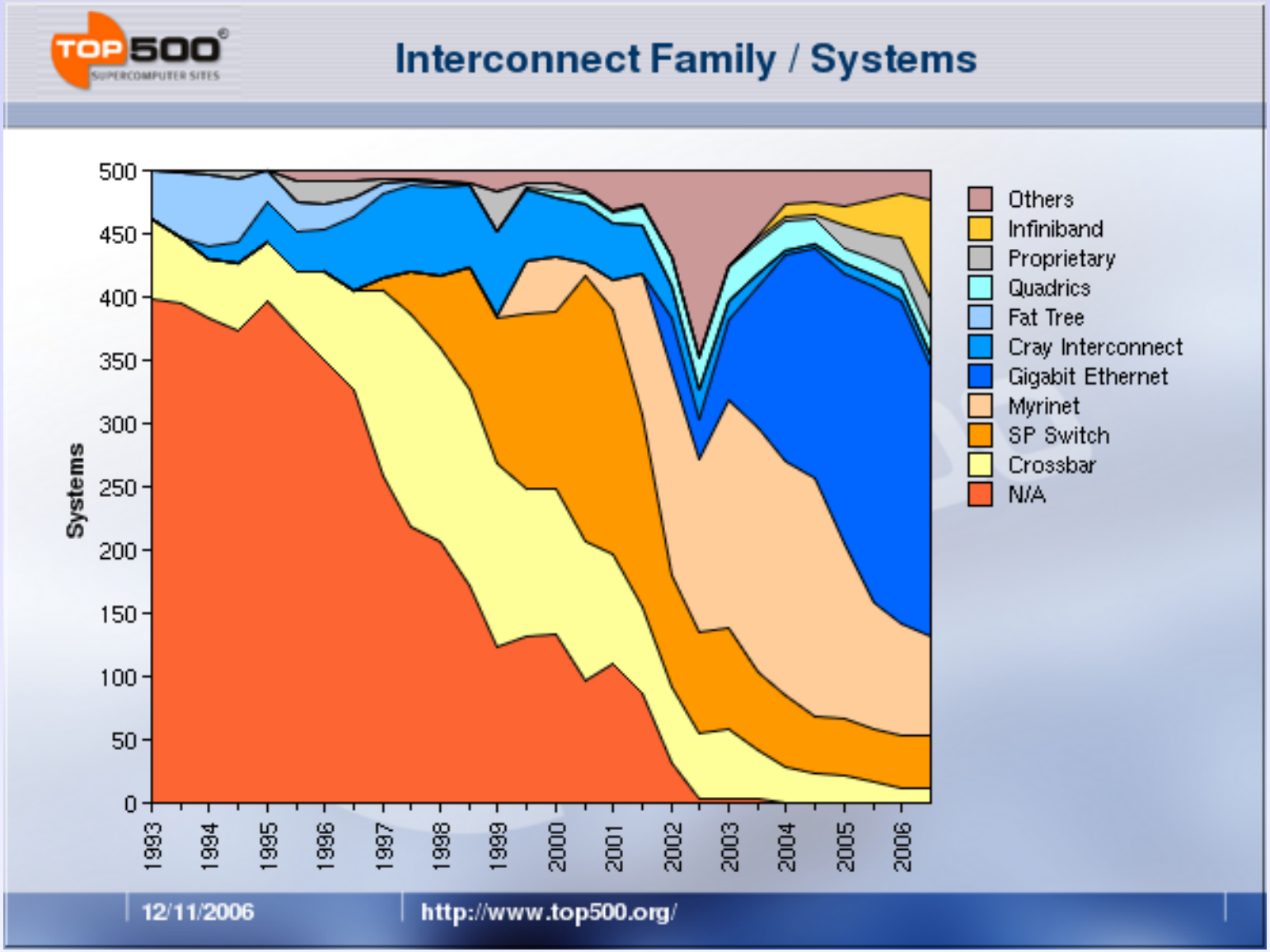
donde:

- L: Latencia de la red
- M: Tamaño del mensaje a enviar
- B: Ancho de banda de la red

	Latencia	Ancho de banda	Precio
Gigabit Ethernet	90μs	0.1 GBs	80€
Myrinet-10G	2μs	1-2 GBs	
Infiniband	2μs	1-2 GBs	800€
Quadrics	2μs	1-2 GBs	
Memoria RAM DDR2	0.02μs	4-8 GBs	

Límites a la paralelización: las comunicaciones

Evolución histórica de las redes de comunicaciones:



Límites a la paralelización: las comunicaciones

Estrategias para **reducir las comunicaciones**:

- Cambiar la formulación del problema
- Cambiar de método numérico
- Duplicar ciertos cálculos
- Modificar el método numérico
- Usar una granularidad más gruesa

Estrategias para **acelerar las comunicaciones**:

- Comprar una red con más ancho de banda
- Comprar una red con menos latencia
- Usar memoria compartida

Diseño de programas paralelos

1. Entender el problema y el programa
2. Inhibidores del paralelismo
3. Identificación de la carga y cuellos de botella
4. Estrategias de descomposición del problema
5. Balance de la carga
6. Requerimientos de comunicación
7. Selección del paradigma de programación

¿Paralelización manual o automática?

Existen herramientas capaces de **autoparalelizar** programas:

- Analizan y buscan oportunidades de paralelismo
- También identifican inhibidores del paralelismo

Pero estas herramientas presentan **serios problemas**:

- El programa puede proporcionar resultados erróneos
- La eficiencia paralela puede ser mala
- Limitado principalmente a bucles
- Estas herramientas son principalmente para Fortran

La paralelización es un trabajo manual y lo seguirá siendo durante muchos años

Entender el problema y el programa

Hay problemas que son paralelizables y otros que no:

- **Problema paralelizable**: calcular el potencial de energía de miles de conformaciones posibles de una misma molécula y determinar la conformación de mínimo potencial de energía

- **Problema no paralelizable**: Calcular la serie de Fibonacci mediante su fórmula de recurrencia:

$$F_{k+2} = F_{k+1} + F_k, \quad F_1 = 1,\ F_2 = 1$$

Carga computacional y cuellos de botella

En un problema/programa hay que identificar:

- Donde se produce la **carga computacional** y tratar de paralelizar sólo esa parte
- Donde están los **cuellos de botella** y tratar de reducirlos o de reordenar convenientemente el programa
- Donde van a hacer falta **comunicaciones** entre procesos (modelo de paso de mensajes)
- Donde van a hacer falta **sincronizaciones** entre tareas (modelo de tareas)

Inhibidores del paralelismo

El motivo más común es la **dependencia de datos**:

- Dependencia de datos **en bucles**:

```
do i = 2, 100
  A(i) = A(i-1) * 2
enddo
```

- Dependencia de datos **fuera de bucles**:

```
! Tarea 1          Tarea 2
  ...               ...
x = 2               x = 4
  ...               ...
y = x**2            z = x**3
```

x está en memoria compartida: ¿Cuál es su valor?

Inhibidores del paralelismo

El segundo motivo es la **entrada y salida de datos (I/O)**:

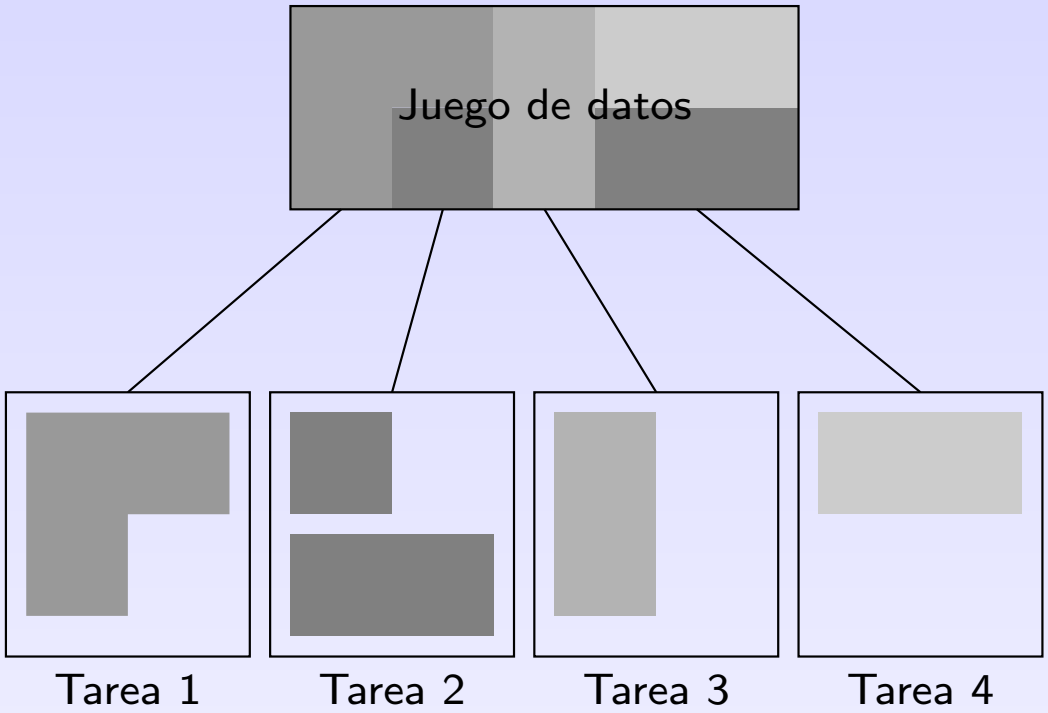
- Sólo un proceso/tarea puede escribir en un fichero
- Los anchos de banda de red y discos son críticos

Estrategias para reducir la entrada y salida de datos:

- **Reducir al máximo** las operaciones I/O
- Crear ficheros independientes para cada proceso
- Usar librerías de lectura y escritura paralela

Descomposición de dominios

Los datos a manipular son **divididos en bloques** a procesar por las diferentes tareas del programa:



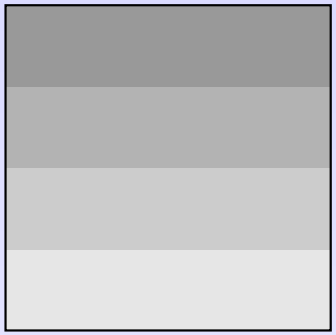
Estrategias de descomposición de dominios

1D:

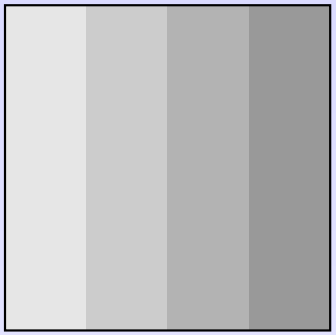
bloques

cíclico

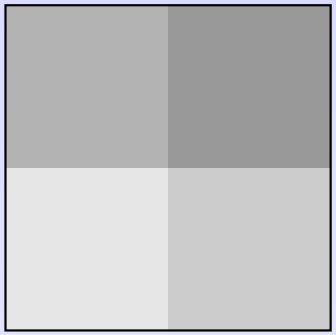
2D:



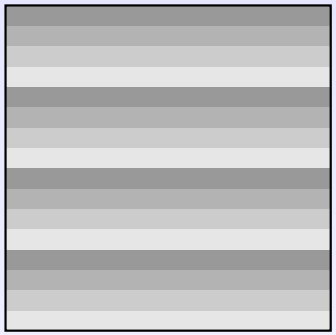
bloques,*



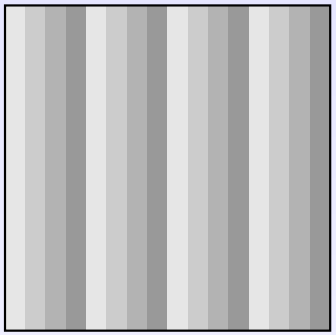
*,bloques



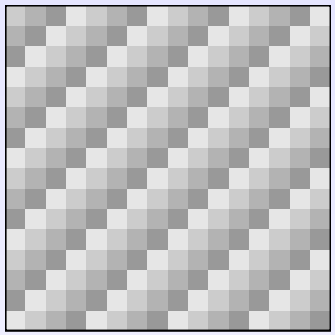
bloques,bloques



cíclico,*



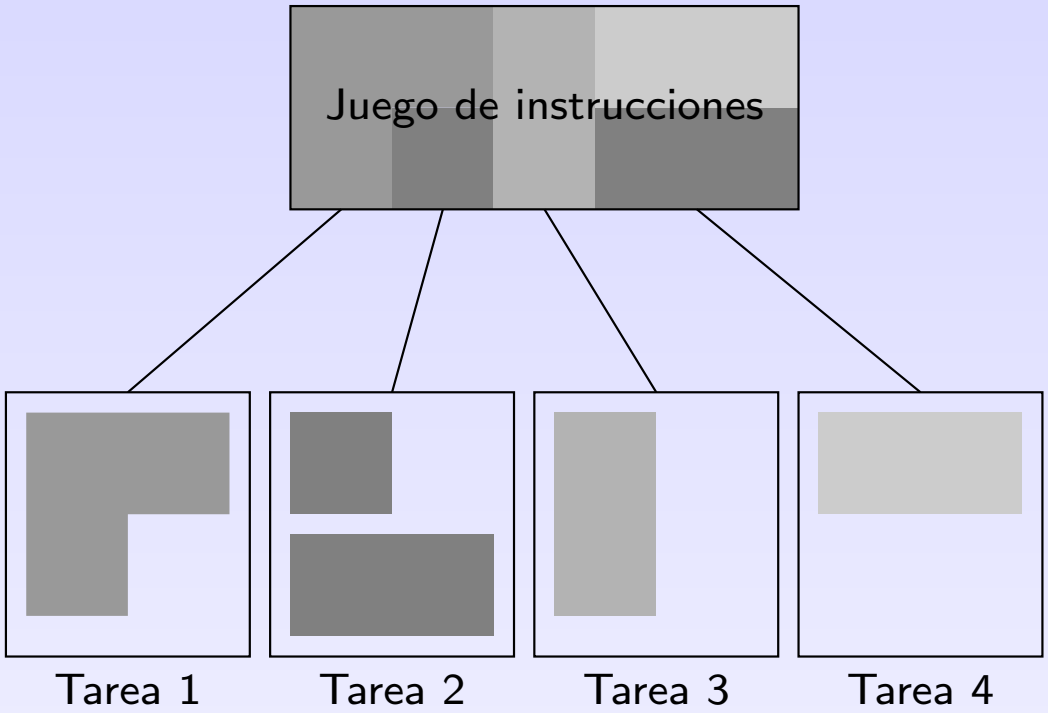
*,cíclico



cíclico,cíclico

Descomposición funcional

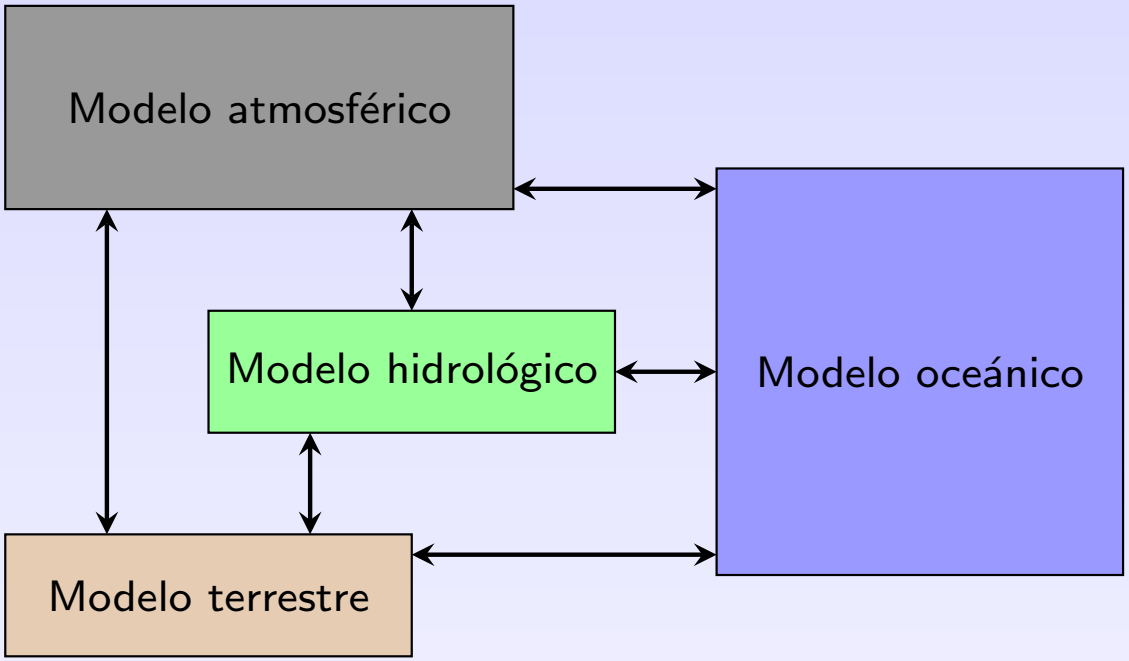
El problema es dividido atendiendo al **trabajo a realizar** en vez de a los datos a manipular:



Descomposición funcional: ejemplos

Modelos climáticos:

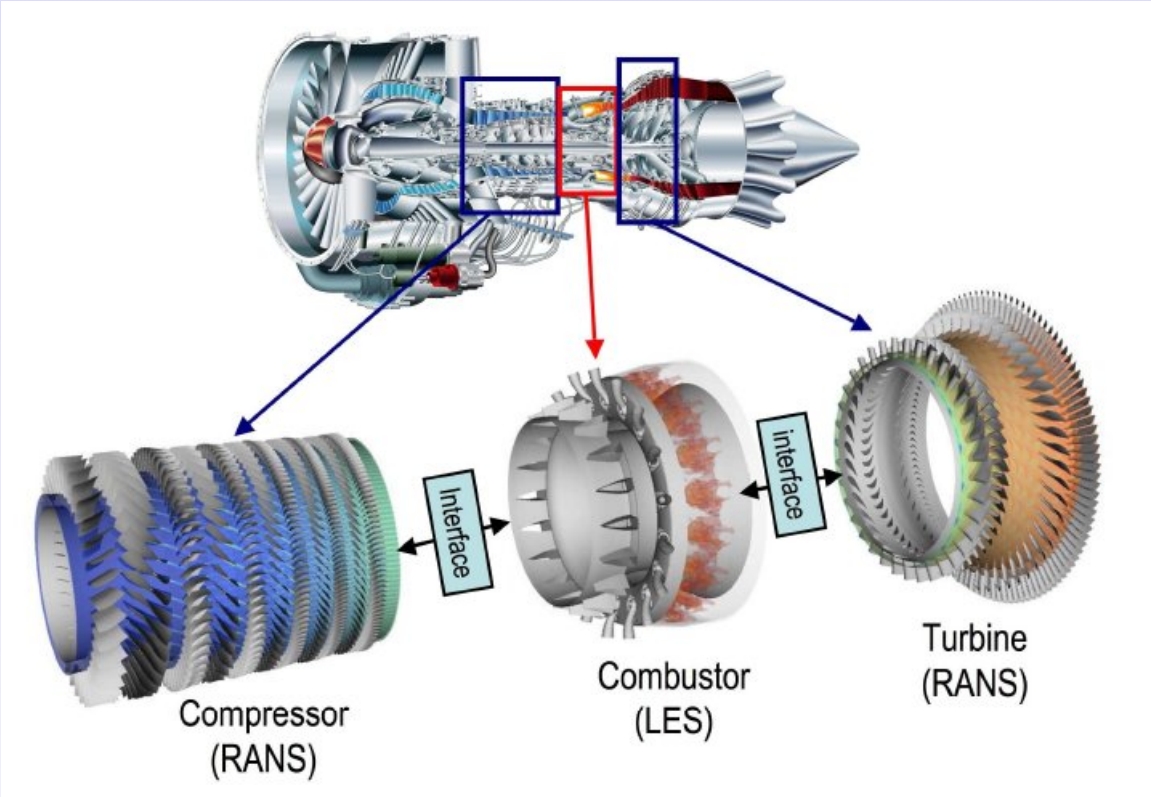
- Cada submodelo representa una tarea independiente
- Los submodelos intercambian información entre sí



Descomposición funcional: ejemplos

Turborreactor (Center for Turbulence Research):

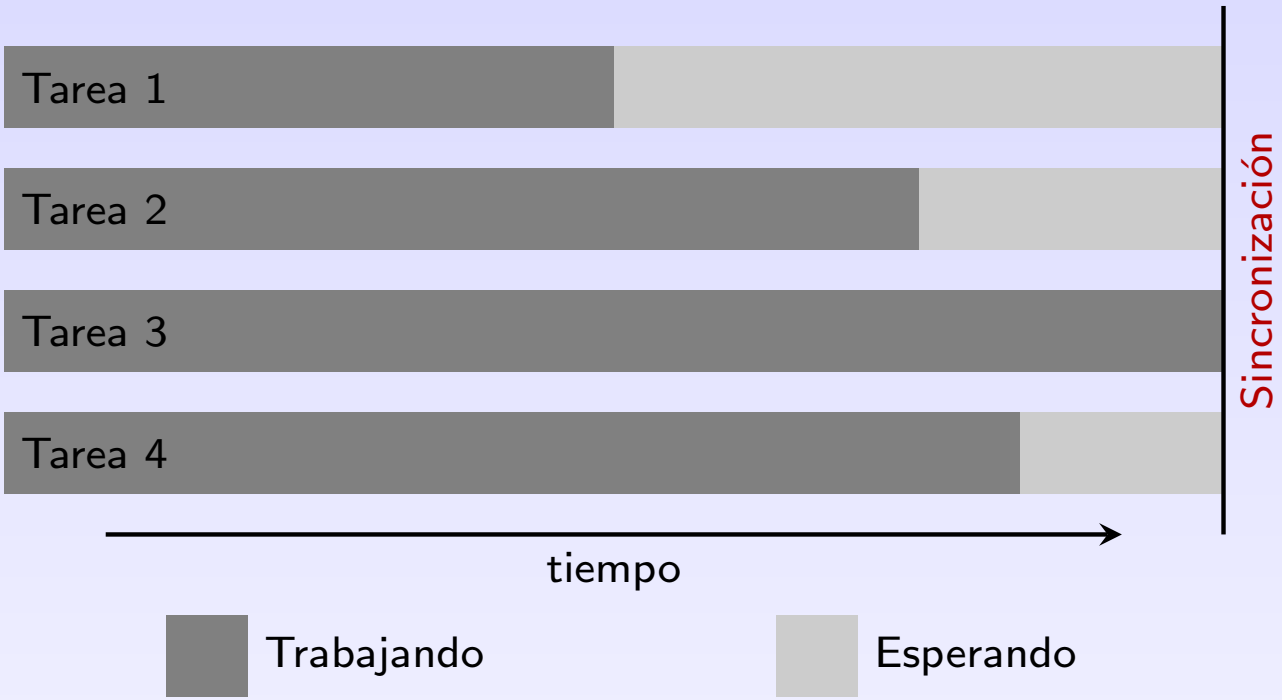
- El turborreactor es dividido en sus componentes
- Los componentes interactúan entre sí



Balance de la carga

La descomposición del problema debe ser **equilibrada**:

- Todos los procesos deben estar trabajando siempre
- Es fundamental para la eficiencia paralela



Balance de la carga

Maneras de equilibrar la carga:

- **Distribución equitativa** del trabajo entre los procesos:
  - Realizar la descomposición de dominios en partes iguales
  - Dividir los bucles en tramos iguales
  - En ordenadores heterogéneos, realizar medidas de velocidad para decidir la descomposición del problema
- **Asignación dinámica** del trabajo:
  - La distribución equitativa no siempre resuelve el problema del balance de la carga
  - El trabajo es asignado a los procesos a medida que terminan las asignaciones anteriores

Requerimientos de comunicación y sincronía

No todos los problemas requieren la misma comunicación:

- **No se necesita** comunicación o sincronía:
  - Renderización de películas
  - Plegado de proteínas
  - Procesamiento de imágenes y señales
- **Sí se necesita** comunicación o sincronía:
  - Mecánica de fluidos computacional
  - Astrofísica
  - Elasticidad y resistencia de materiales

Selección del paradigma de programación

Teniendo en cuenta todo lo anterior y además:

- El tipo de ordenador paralelo que se vaya a utilizar
- Las características del problema/programa a paralelizar
- El tiempo disponible para la implementación paralela

se **selecciona el paradigma** de programación apropiado:

- Paradigma de paso de mensajes: **MPI**
- Paradigma de tareas: **OpenMP**
- Otro paradigma ...

Ejemplos de paralelización

Problemas a paralelizar:

- Procesamiento de imágenes
- Cálculo de una integral
- Ecuación de ondas unidimensional
- Ecuación del calor bidimensional

Objetivos a alcanzar:

- Asentar los conceptos del paralelismo
- Ver el efecto de la naturaleza de los problemas
- Ver el efecto de cambiar el método numérico
- Ver el efecto de cambiar la estrategia de descomposición

Procesamiento de imágenes

- **Problema**: aplicar un filtro $f(x, y, c)$ a una imagen dada
- **Versión serial** del algoritmo:

```
do j = 1, Numero_Pixels_en_j
  do i = 1, Numero_Pixels_en_i
    Color        = Imagen(i,j)
    Imagen(i,j) = f(i,j,Color)
  enddo
enddo
```

- La aplicación del filtro a un pixel no depende de los pixels vecinos
- Es **embarazosamente paralelo**

$\downarrow$

$f(x, y, c)$

$\downarrow$

Procesamiento de imágenes: solución 1

Descomposición de dominios

- División equitativa de la imagen
- Nula comunicación entre procesos
- La estrategia de descomposición depende del lenguaje (Fortran o C)
- Cada proceso o tarea realiza:

```
j1 = Mi_Primera_Columna
j2 = Mi_Ultima_Columna

do j = j1, j2
  do i = 1, Numero_Pixels_en_i
    Color        = Imagen(i,j)
    Imagen(i,j) = f(i,j,Color)
  enddo
enddo
```

1 2 3 4

(Fortran)

Procesamiento de imágenes: solución 1

Versión paralela del algoritmo:

```
ID = Quien_soy_yo

Si soy ID = Jefe entonces

  Cargo Imagen desde fichero
  Mando a cada Currito los valores j1 y j2
  Mando a cada Currito su parte de Imagen
  Recibo de cada Currito los resultados

Si soy ID = Currito entonces

  Recibo j1 y j2 del Jefe
  Recibo mi parte de Imagen del Jefe

  do j = j1, j2
    do i = 1, Numero_Pixels_en_i
      Color        = Imagen(i,j)
      Imagen(i,j) = f(i,j,Color)
    enddo
  enddo

  Devuelvo mi parte de Imagen al Jefe
```

Procesamiento de imágenes: solución 2

Banco de tareas (pool of tasks):

- Es una manera **dinámica** de balancear la carga
- Se crea un banco de tareas/trabajos a realizar
- Un **proceso maestro**:
  - Gestiona el banco de tareas
  - Envía tareas a los procesos que lo requieren
  - Recolecta los resultados enviados por los procesos
- Un **proceso esclavo**:
  - Recibe tareas del proceso maestro
  - Realiza las tareas encomendadas
  - Devuelve los resultados al proceso maestro

Procesamiento de imágenes: solución 2

Versión paralela del algoritmo (1 maestro y 1 esclavo):

```
ID = Quien_soy_yo

Si soy ID = Jefe entonces

  Mientras haya tareas en el banco
    Mando al Currito la siguiente tarea (i,j)
    Recibo del Currito los resultados

  Avisa al Currito de que ha terminado

Si soy ID = Currito entonces

  Mientras haya tareas que realizar
    Recibo del Jefe la tarea (i,j)

    Color       = Imagen(i,j)
    Imagen(i,j) = f(i,j,Color)

    Mando los resultados al Jefe
```

Procesamiento de imágenes: solución 2

Características del método del banco de tareas:

- Cada tarea consiste en aplicar el filtro al pixel (i,j)
- El **balance de carga es bueno**
- Poco cálculo entre comunicaciones: **granularidad fina**
- Una mejora sería definir tareas con más trabajo
- El **tamaño óptimo de tarea** depende del problema

Cálculo de una integral

- **Problema**: calcular la integral definida siguiente:

$$I = \int_a^b f(x)\,\mathrm{d}x$$

- Las integrales definidas tienen la siguiente propiedad:

$$I = \int_a^{x_1} f(x)\,\mathrm{d}x + \int_{x_1}^{x_2} f(x)\,\mathrm{d}x + \ldots + \int_{x_N}^{b} f(x)\,\mathrm{d}x$$

- El propio problema indica la estrategia paralela a seguir: **descomposición de dominios**

- Calcular la integral es **embarazosamente paralelo**

Cálculo de una integral

Versión paralela del algoritmo de integración:

```
ID = Quien_soy_yo

Si soy ID = Jefe entonces

  Subdivido [a,b] en subintervalos

  Mando a los Curritos sus valores a_i y b_i
  Recibo de los Curritos los resultados

  Sumo todos los resultados para obtener la integral

Si soy ID = Currito entonces

  Recibo del Jefe los valores a_i y b_i

  Integro f(x) desde a_i hasta b_i

  Mando el resultado al Jefe
```

Ecuación de ondas unidimensional

- **Problema**: resolver la ecuación diferencial

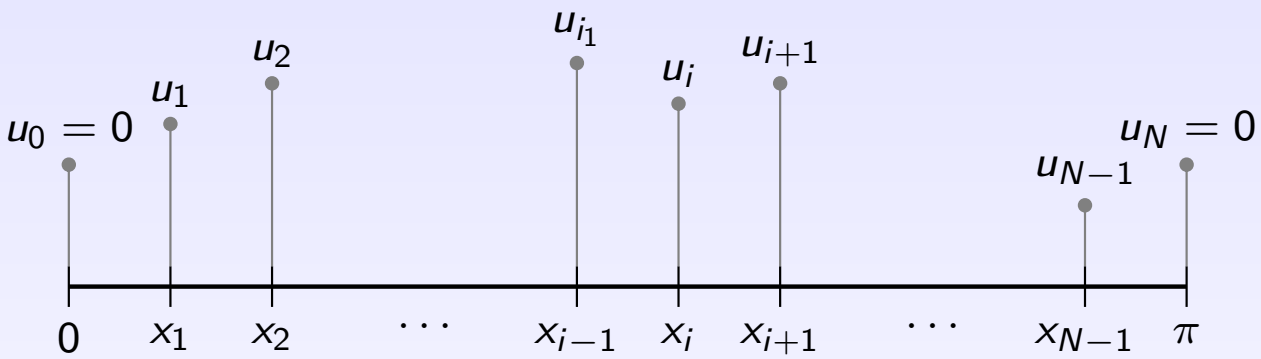
$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

con las condiciones de contorno e iniciales

$$u(0,t) = 0, \quad u(\pi,t) = 0, \quad u(x,0) = f(x)$$

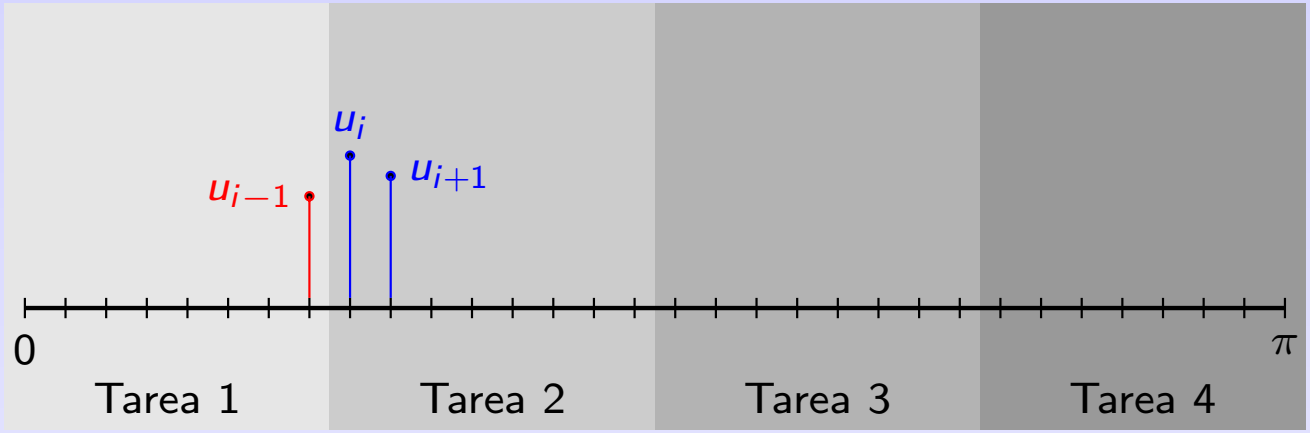
- Con **diferencias finitas** de segundo orden:

$$\frac{\mathrm{d}^2 u_i}{\mathrm{d}t^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}, \qquad \Delta x = \frac{\pi}{N}$$



Ecuación de ondas unidimensional

Estrategia paralela: **descomposición del dominio**



Hacen falta **comunicaciones** para resolver el problema:

$$\frac{\mathrm{d}^2 u_i}{\mathrm{d}t^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

La carga está **balanceada** si los bloques de nodos son iguales

Ecuación de ondas unidimensional

Versión paralela de las diferencias finitas:

```
ID  = Quien_soy_yo
NID = Cuantos_somos

Divido el intervalo [0,pi] en NID partes
Me quedo con la parte ID del intervalo

Leo la condicion inicial que me corresponde

Mientras que t < t_final

  Avanzo un paso temporal dt

  Comunico a los procesos vecinos mis valores frontera
  Recibo de los procesos vecinos sus valores frontera

  Si ID = 0 o N impongo condiciones de contorno
```

¿Se puede hacer mejor que esto?

Ecuación de ondas unidimensional

Cambio de método numérico: Fourier-Galerkin

$$u(x,t) = \sum_{k=1}^{\infty} \hat{u}_k(t) \sin(kx)$$

El problema a resolver es:

$$\frac{\mathrm{d}^2 \hat{u}_k}{\mathrm{d}t^2} = -k^2 \hat{u}_k, \qquad \hat{u}_k(0) = \frac{1}{\pi} \int_{-\pi}^{\pi} u(x,0) \sin(kx)\, \mathrm{d}x$$

- Estrategia paralela: descomposición en frecuencias
- **No** hacen falta comunicaciones
- Este problema es **embarazosamente paralelo**

¡La estrategia paralela **depende del método numérico**!

Ecuación del calor bidimensional

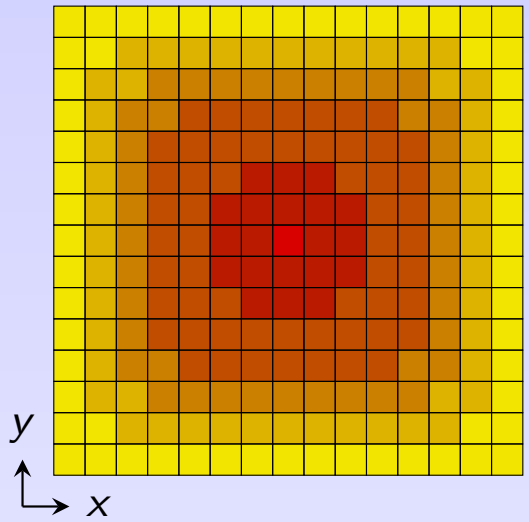
Problema: resolver la ecuación

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}$$

con las condiciones

$$T(x, y, t)|_{\partial\Omega} = g(s)$$

$$T(x, y, 0) = f(x, y)$$



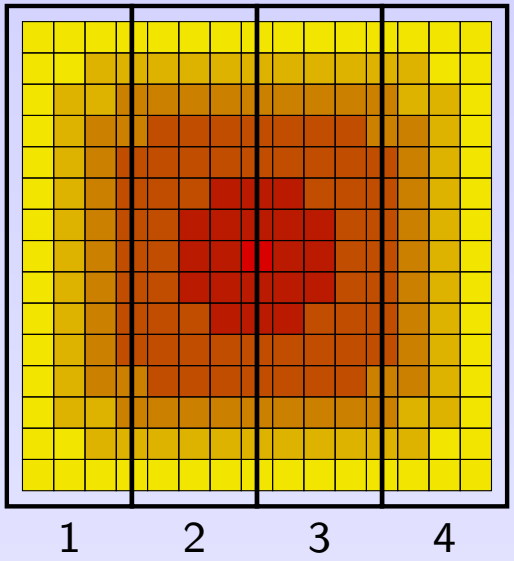
Método numérico: diferencias finitas de 2º orden

$$\frac{\mathrm{d}T_{i,j}}{\mathrm{d}t} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2}$$

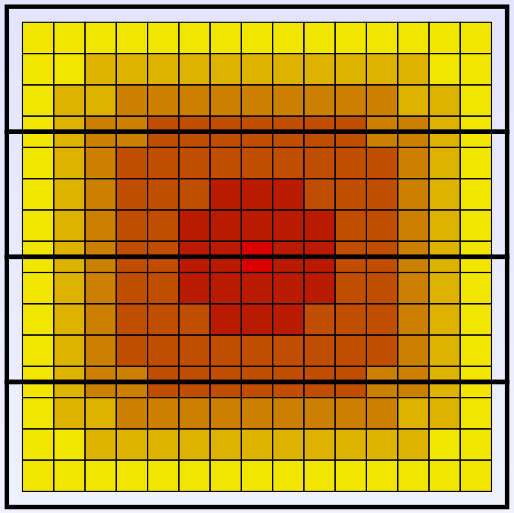
Estrategia paralela: descomposición de dominios

Ecuación del calor bidimensional

Tres maneras de hacer la descomposición de dominios:



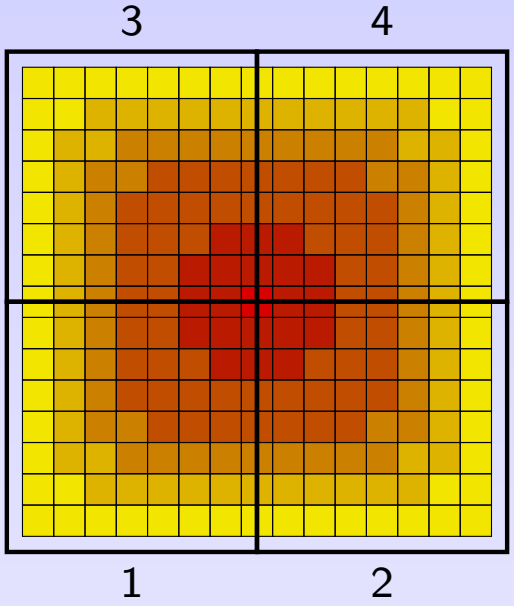
- Organización de memoria buena para Fortran
- El volumen de comunicaciones es del orden de $N_y N_{\text{proc}}$



- Organización de memoria buena para C
- El volumen de comunicaciones es del orden de $N_x N_{\text{proc}}$

Ecuación del calor bidimensional

Tres maneras de hacer la descomposición de dominios:



- La organización de memoria no es óptima, pero es buena
- El volumen de comunicaciones es del orden de $(N_x + N_y)\sqrt{N_{\text{proc}}}$

NO exite la descomposición perfecta:

- Depende del lenguaje de programación
- Depende del ordenador paralelo
- Depende del método numérico
- Depende del problema ...

Resumen

1. Motivación de la programación en paralelo
2. Clasificación lógica del paralelismo
3. Clasificación física de ordenadores paralelos
4. Paradigmas de programación paralela
5. Conceptos generales y terminología habitual
6. Diseño de programas paralelos
7. Ejemplos de paralelización

Agradecimientos

Oscar Flores

Universidad Politécnica de Madrid