Ingeniería de los Computadores

Unidad 3. Computación paralela

Introducción

¿Dónde se usa la supercomputación?

- **Defensa**
  - Lucha antiterrorista
  - Programas de armas
  - Programa espacial
- **Ingeniería/Academia/Investigación**
  - Prospección de nuevas energías y simulación
  - Nanotecnología
  - Modelado biológico
  - *Deep learning*
  - Investigación genoma
  - Investigación proteínas
  - Simulaciones astrofísicas
  - Predicción meteorológica
  - Diseño de fármacos
  - Modelado geológico
  - ...

Procesamiento paralelo versus procesamiento distribuido

Procesamiento paralelo

- Estudia los aspectos relacionados con la **división de una aplicación en unidades independientes** y su ejecución en múltiples procesadores para reducir tiempo y/o aumentar la complejidad del problema.

Procesamiento distribuido

- Estudia los aspectos relacionados con la ejecución de **múltiples aplicaciones** al mismo tiempo utilizando múltiples recursos (procesadores, memorias, discos y bases de datos) situados en distintas localizaciones físicas.

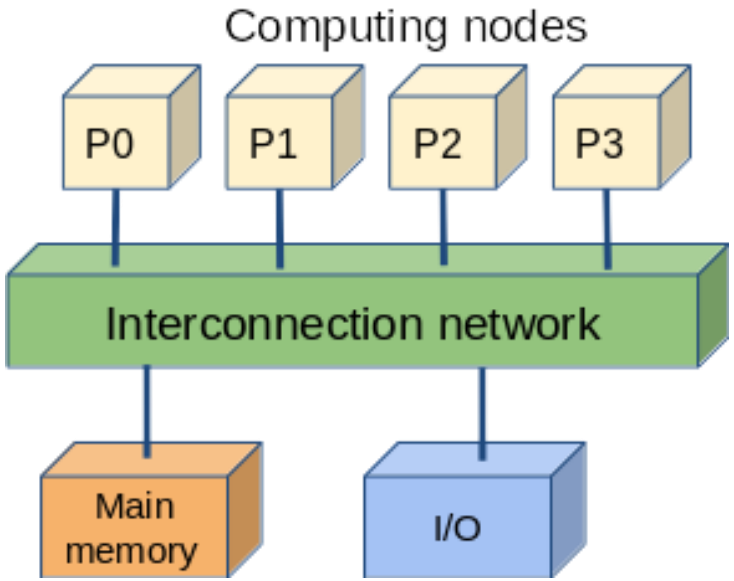
Clasificación de computadores paralelos (según el sistema de memoria):

- **Multiprocesadores**: Comparten el mismo espacio de memoria (el programador no necesita saber dónde están los datos)

- **Multicomputadores**: Cada procesador (nodo) tiene su propio espacio de direcciones (el programador necesita saber dónde están los datos)

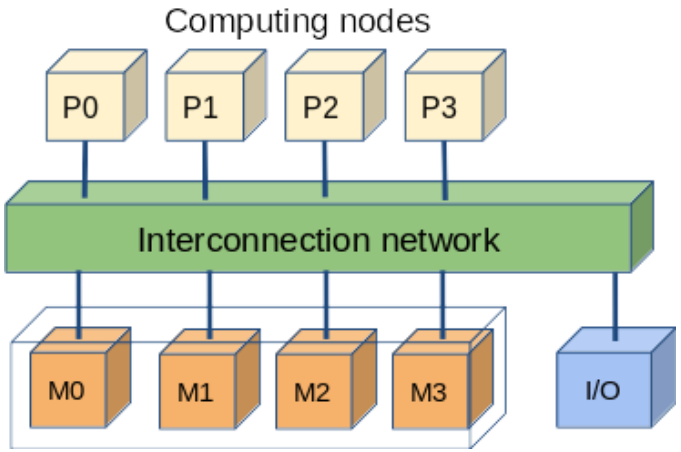
Diseño de un computador paralelo:

- Nodos de cómputo
- Sistema de memoria
- Sistema de comunicación
- Sistema de entrada/salida

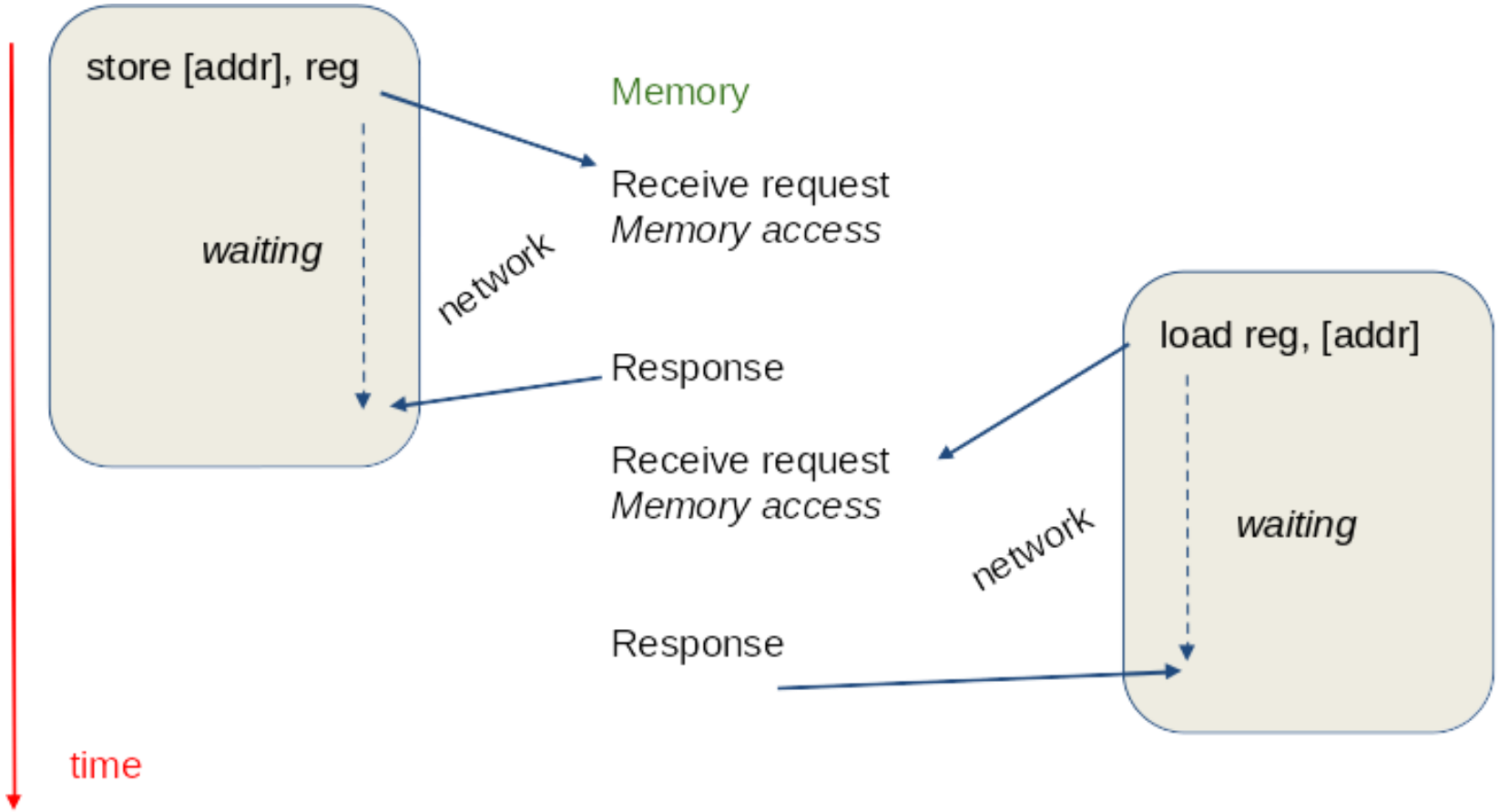


Multiprocesador con memoria centralizada (SMP – *Symmetric MultiProcessor*)

- Mayor latencia
- Poco escalable
- Comunicación mediante variables compartidas (datos no duplicados en memoria principal)
- Necesita implementar primitivas de sincronización
- No necesita distribuir código y datos
- Programación más sencilla (generalmente)

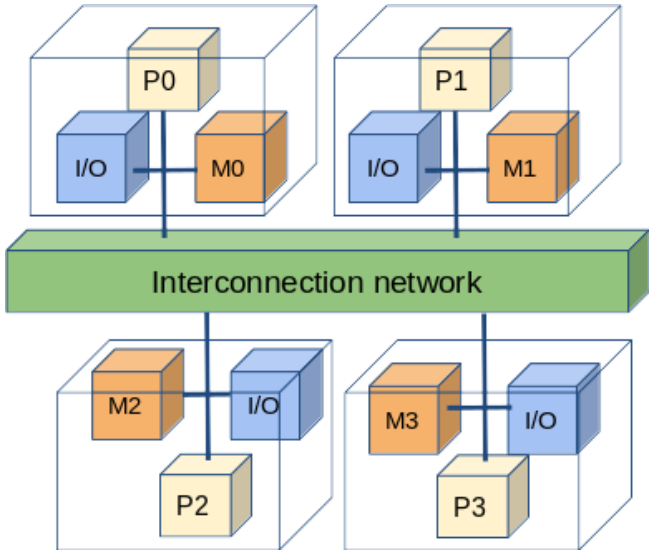


Comunicación en un multiprocesador

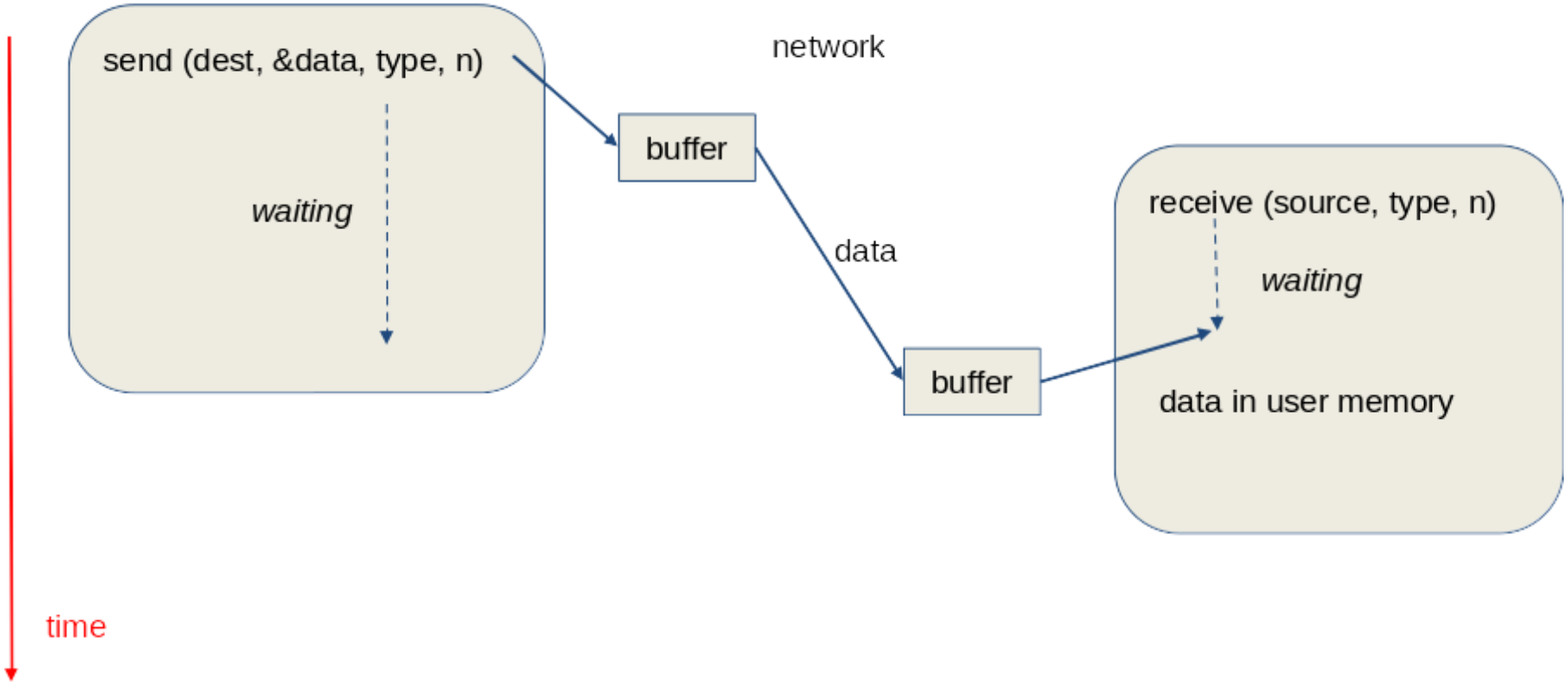


Multicomputador:

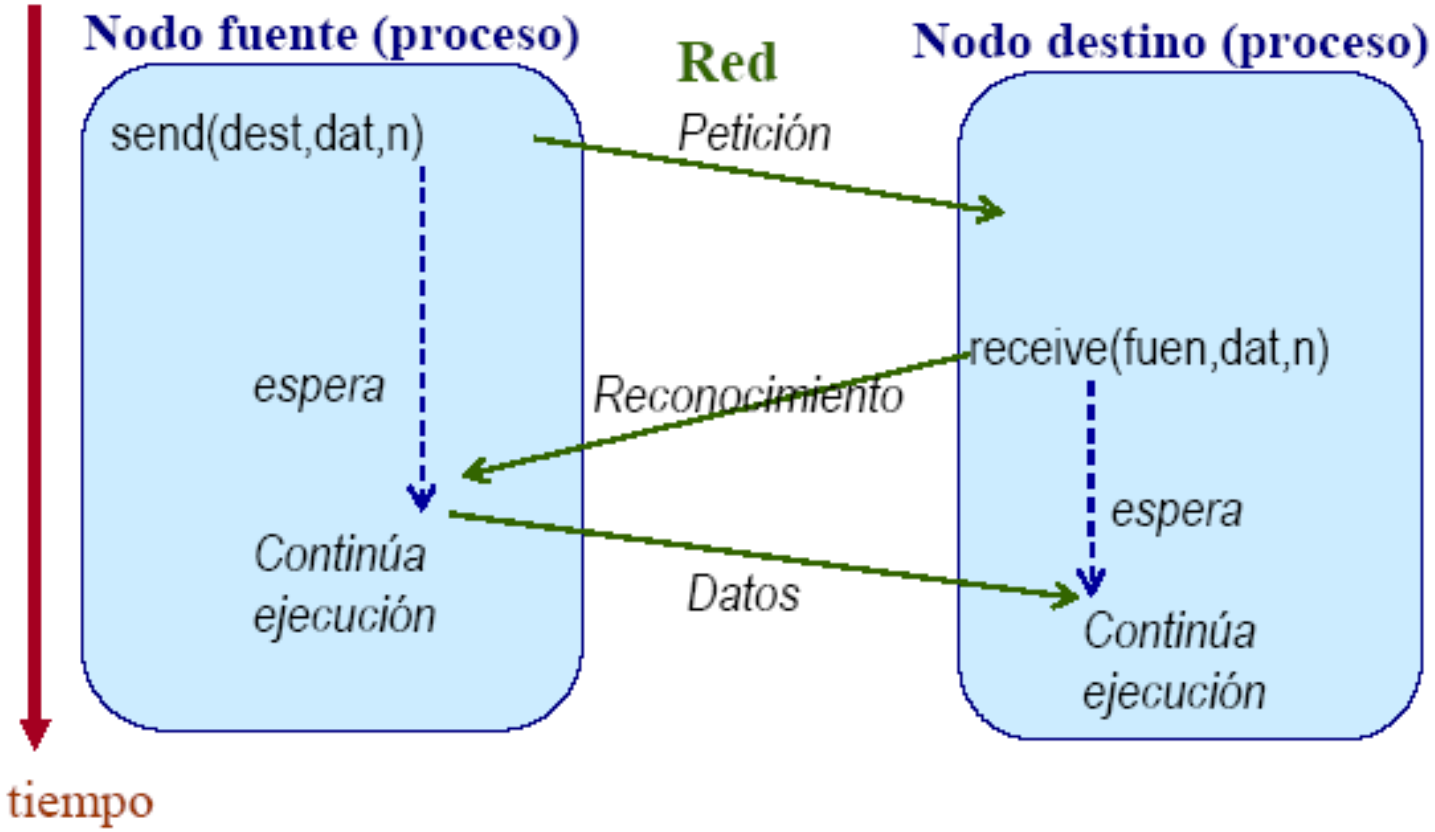
- Menor latencia
- Mayor escalabilidad
- Comunicación mediante **paso de mensajes** (datos duplicados en memoria)
- Sincronización mediante mecanismos de comunicación
- Distribución de carga de trabajo (código y datos) entre procesadores
- Programación más difícil



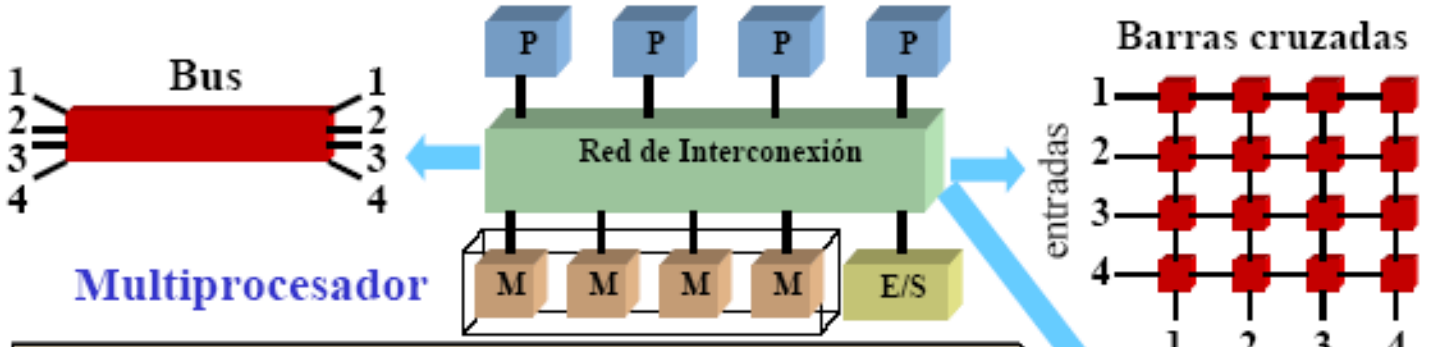
Comunicación asíncrona en un multicomputador



Comunicación síncrona en un multicomputador



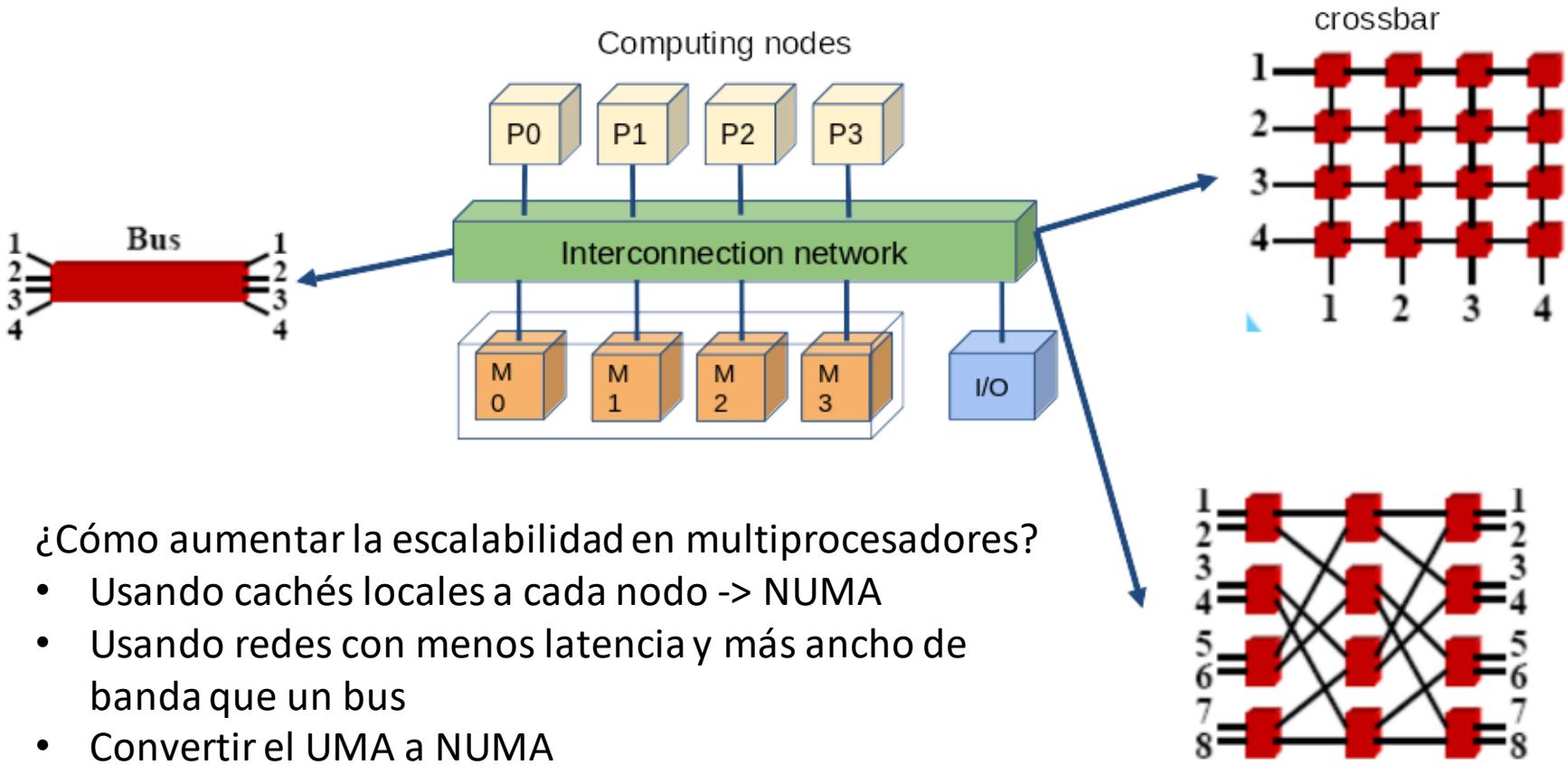
Redes de interconexión en multiprocesadores



Incremento escalabilidad multiprocesadores:

- Añadir **cache local** a cada 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).

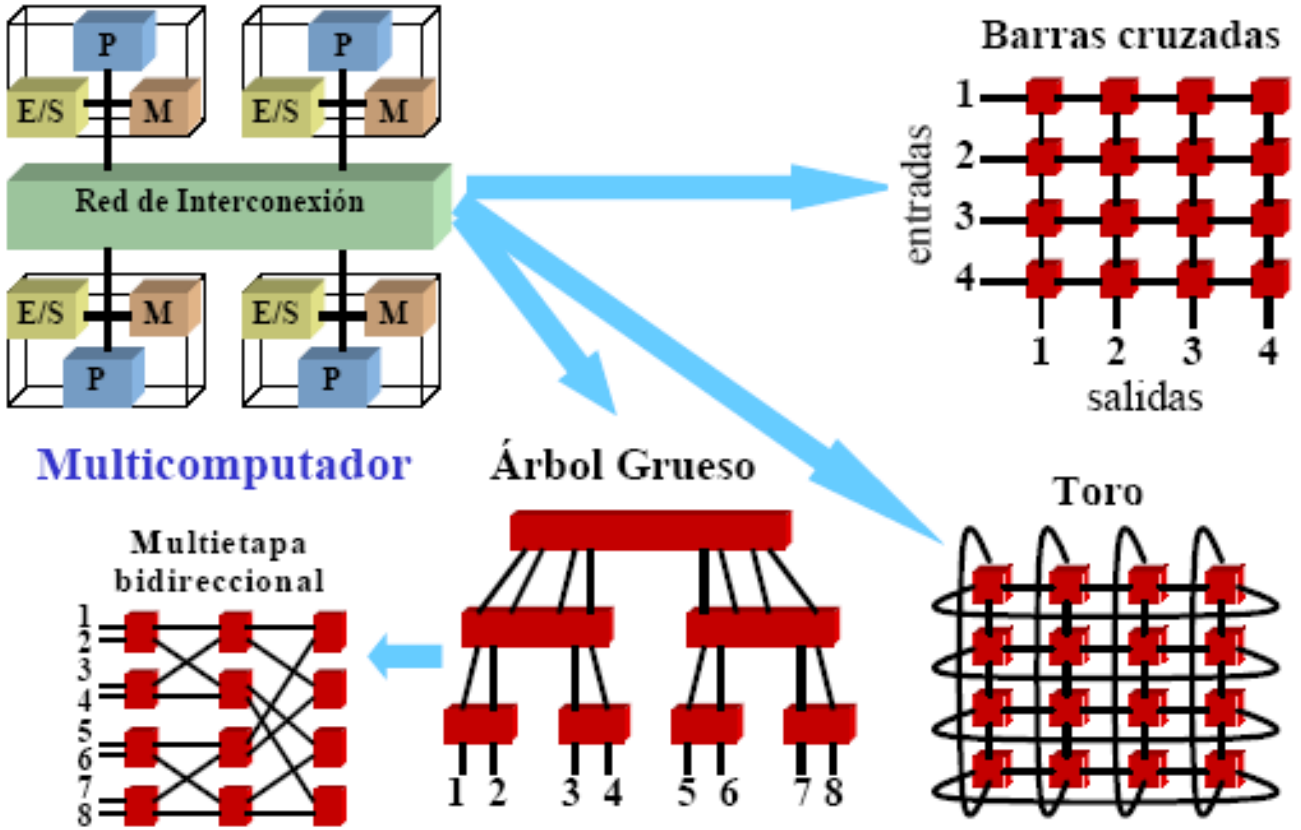
Redes de interconexión en multiprocesadores



¿Cómo aumentar la escalabilidad en multiprocesadores?

- Usando cachés locales a cada nodo -> NUMA
- Usando redes con menos latencia y más ancho de banda que un bus
- Convertir el UMA a NUMA

Redes de interconexión en multicomputadores



Clasificación de computadores paralelos

<table>
<tr><td>Multicomputadores
Memoria no compartida
Múltiples espacios de direcciones.</td><td colspan="3">IBM-SP2
HP AlphaServer SC45 (cluster de SMP)</td><td rowspan="2">Memoria físicamente distribuida

Escalables</td><td rowspan="4">+
Escalabilidad
−</td></tr>
<tr><td rowspan="3">Multiprocesadores
Memoria compartida
Un único espacio de direcciones.</td><td rowspan="3">NUMA
(Non-Uniform Memory Access)</td><td>NUMA</td><td>Cray T3E, Cray X1</td></tr>
<tr><td>CC-NUMA</td><td>Origin 3000
HP 9000 Superdome</td></tr>
<tr><td>COMA</td><td>KSR-1</td></tr>
<tr><td>UMA
(Uniform Memory Access)</td><td>SMP</td><td>WS,
servidores básicos</td><td>Memoria físicamente centralizada</td></tr>
</table>

Conceptos

Computadores paralelos y escalabilidad

Podemos ordenar los computadores paralelos en función de su escalabilidad creciente:

UMA/SMP < COMA < CC-NUMA < NUMA < Multicomputers

Ejemplos:

Multicomputers:	IBM-SP2, HP AlphaServer SC45 (clúster o SMP)
NUMA :	Cray T3E, Cray X1
CC-NUMA :	Origin 3000, HP 9000 Superdome
COMA:	KSR-1
SMP:	¡Intentad adivinar!

Otros tipos de computadores paralelos

- **MPP (Massive Parallel Processor)**. Número de procesadores superior a 100. Sistemas con redes diseñadas a medida.
- **Cluster**. Computadores completos (PC's, servidores, etc.) conectados con alguna red comercial tipo LAN que se utiliza como recurso de cómputo único. El tráfico de la red se reduce al ocasionado por la aplicación ejecutada (no hay tráfico externo)
- **Cluster Beowulf**. Cluster con sistema operativo libre (Linux) y hardware y software de amplia difusión
- **Constelaciones**. Cluster de SMP con un nº de procesadores en un nodo mayor que el nº de nodos del cluster

Otros tipos de computadores paralelos

- **Redes de computadores**. Conjunto de computadores conectados por una LAN. Por la red circula tanto tráfico de la aplicación paralela como externo
- **GRID**. Conjunto de recursos autónomos distribuidos geográficamente conectados por infraestructura de telecomunicaciones y que conforman un sistema de altas prestaciones virtual

Tipos de paralelismo

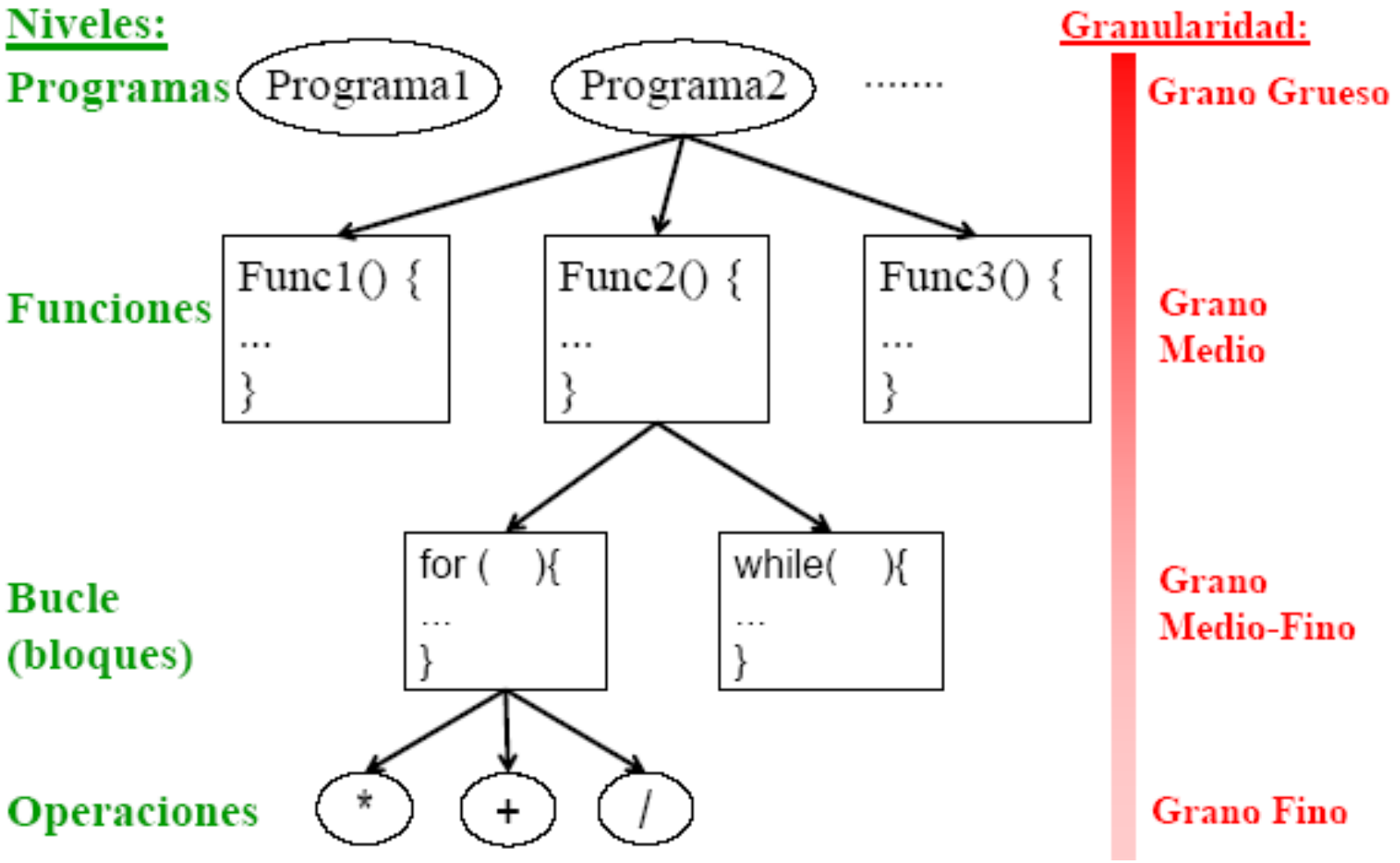
Paralelismo funcional:

- Se obtiene a través de la **reorganización de la estructura lógica** de una aplicación. Existen diferentes niveles de paralelismo funcional según las estructuras que se reorganicen.

Paralelismo de datos:

- Implícito en operaciones con estructuras de datos tipo **vector** o **matriz**. Está relacionado con operaciones realizadas sobre grandes volúmenes de datos que sean independientes entre si.

Niveles y granularidad del paralelismo funcional:



Tipos de paralelismo con respecto a su visibilidad:

Paralelismo explícito

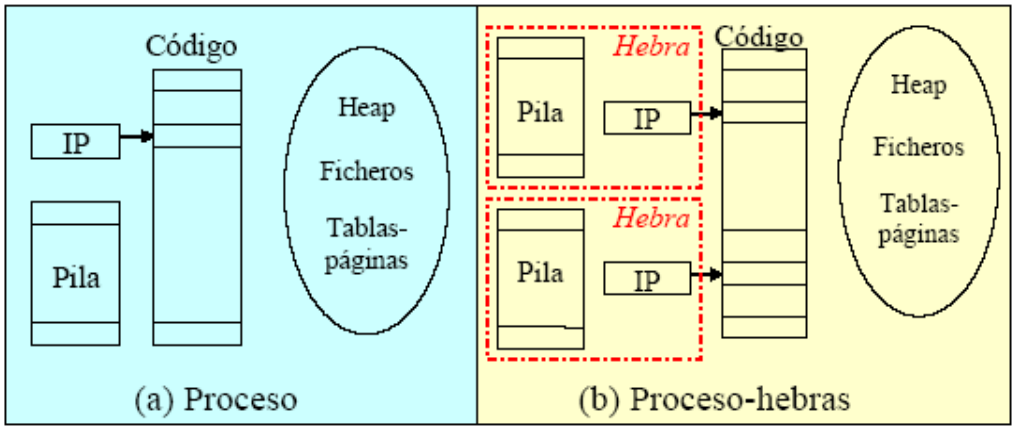
- Paralelismo no presente de forma inherente en las estructuras de programación y que **se debe indicar expresamente**.

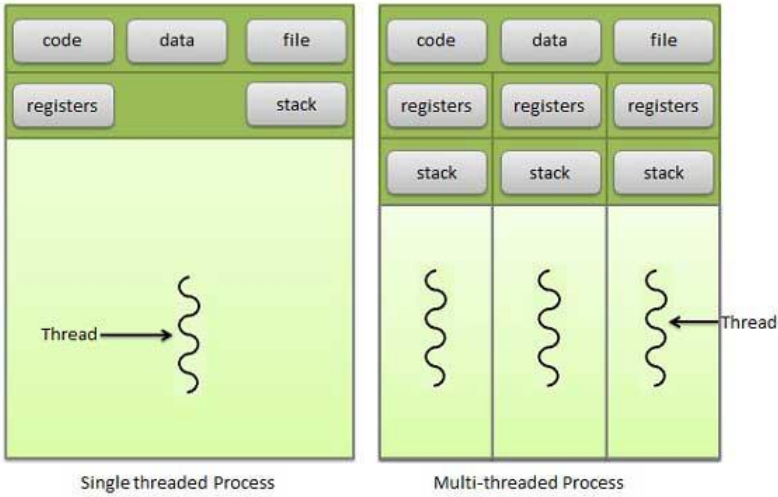
Paralelismo implícito

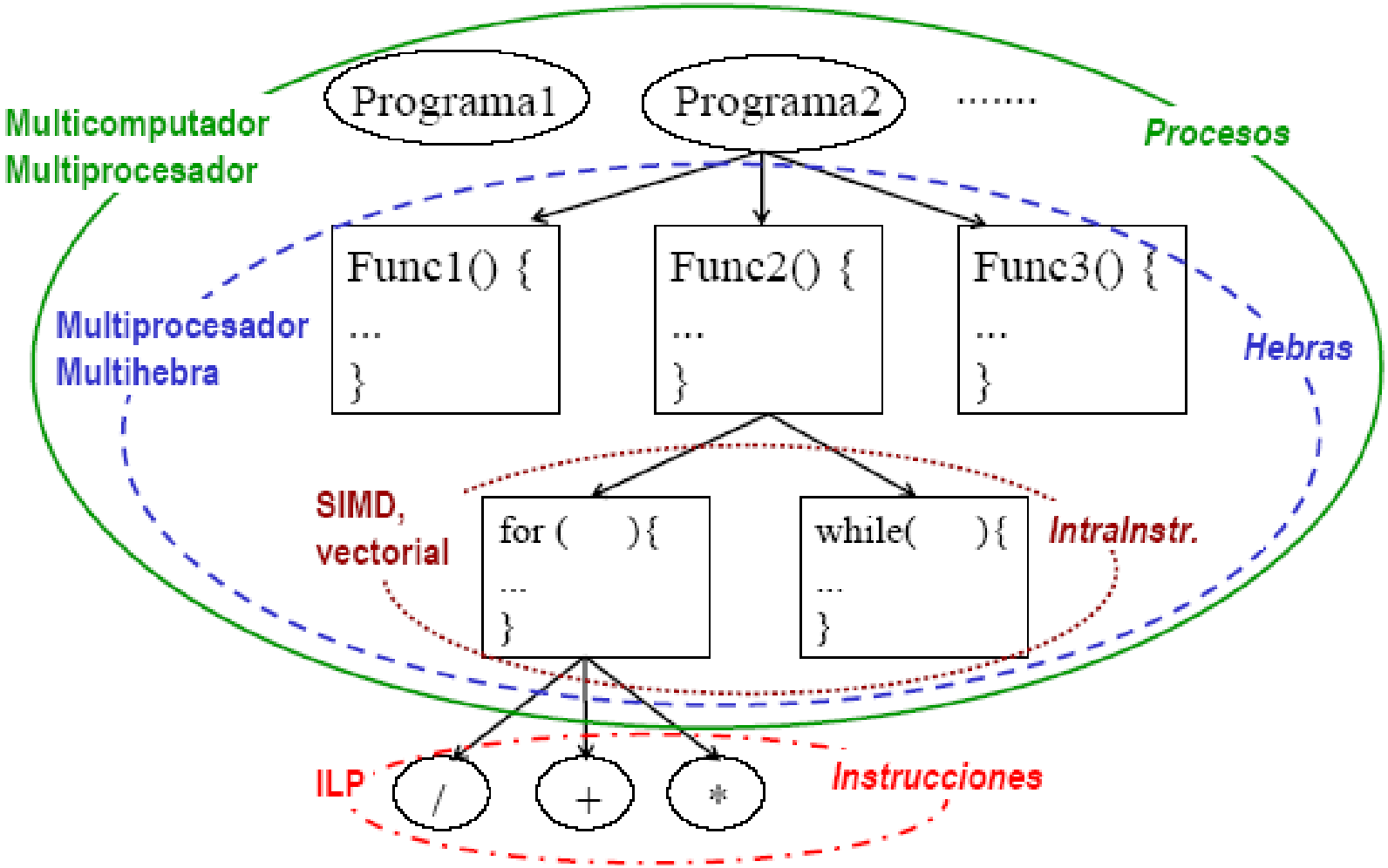
- Paralelismo presente (aunque puede no estar ejecutándose de forma paralela) debido a la propia estructura de los datos (vectores) o de la aplicación (bucle)

Unidades de ejecución: hilos y procesos

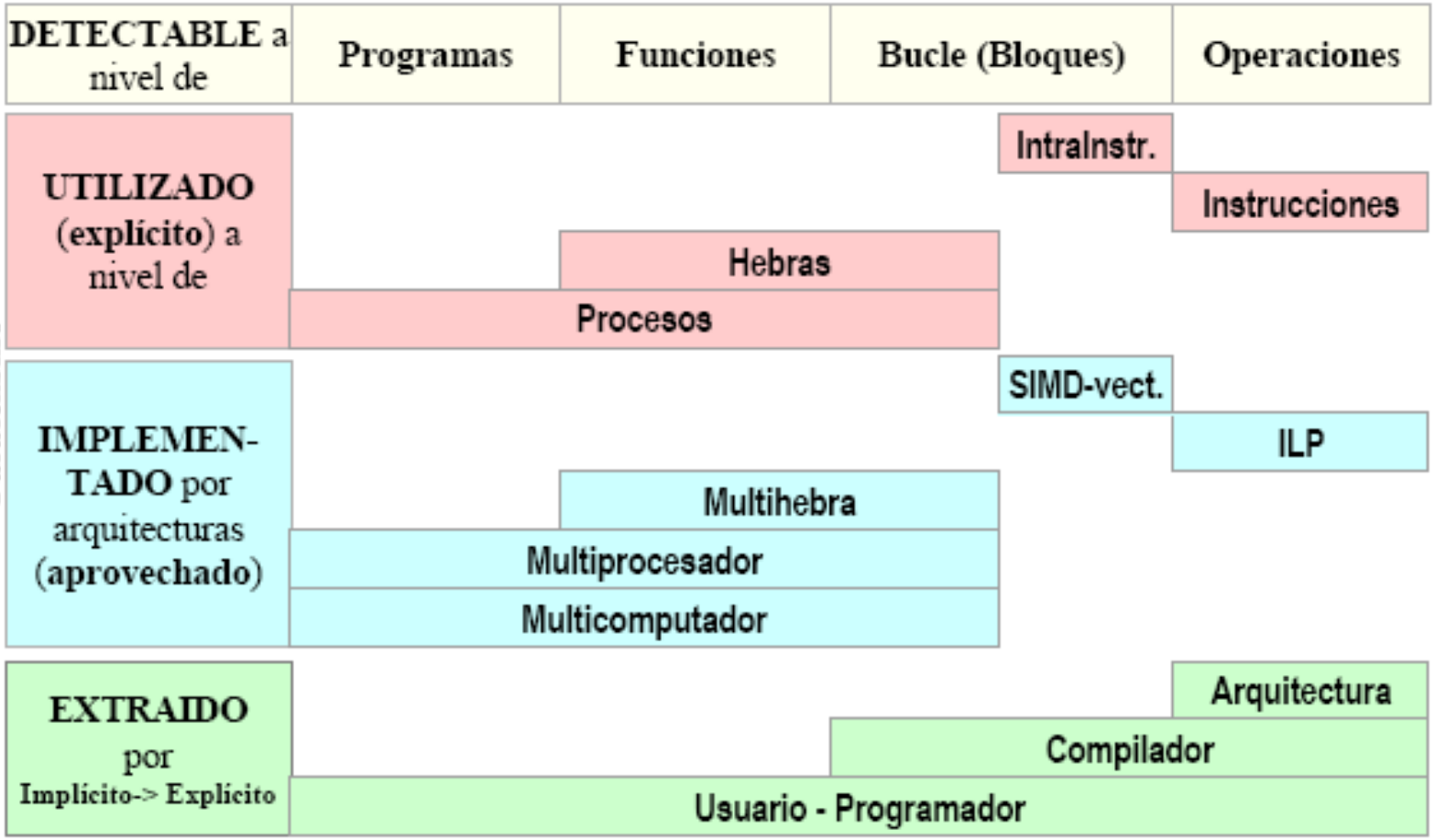
- Hardware (procesador): Gestiona la ejecución de las instrucciones
- Software (SO): Gestiona la ejecución de hilos y procesos
  - **Proceso**: espacio de direcciones virtuales propio
  - **Hilos/hebra/*thread***: comparten direcciones virtuales, se crean y destruyen más rápido y la comunicación también es más rápida






Multicomputador
Multiprocesador
Programa1
Programa2
.....
Procesos
Multiprocesador
Multihebra
Func1() {
...
}
Func2() {
...
}
Func3() {
...
}
Hebras
SIMD,
vectorial
for ( ){
...
}
while( ){
...
}
IntraInstr.
ILP
/
+
*
Instrucciones

Detección y extracción de paralelismo



Problemas introducidos por la programación paralela

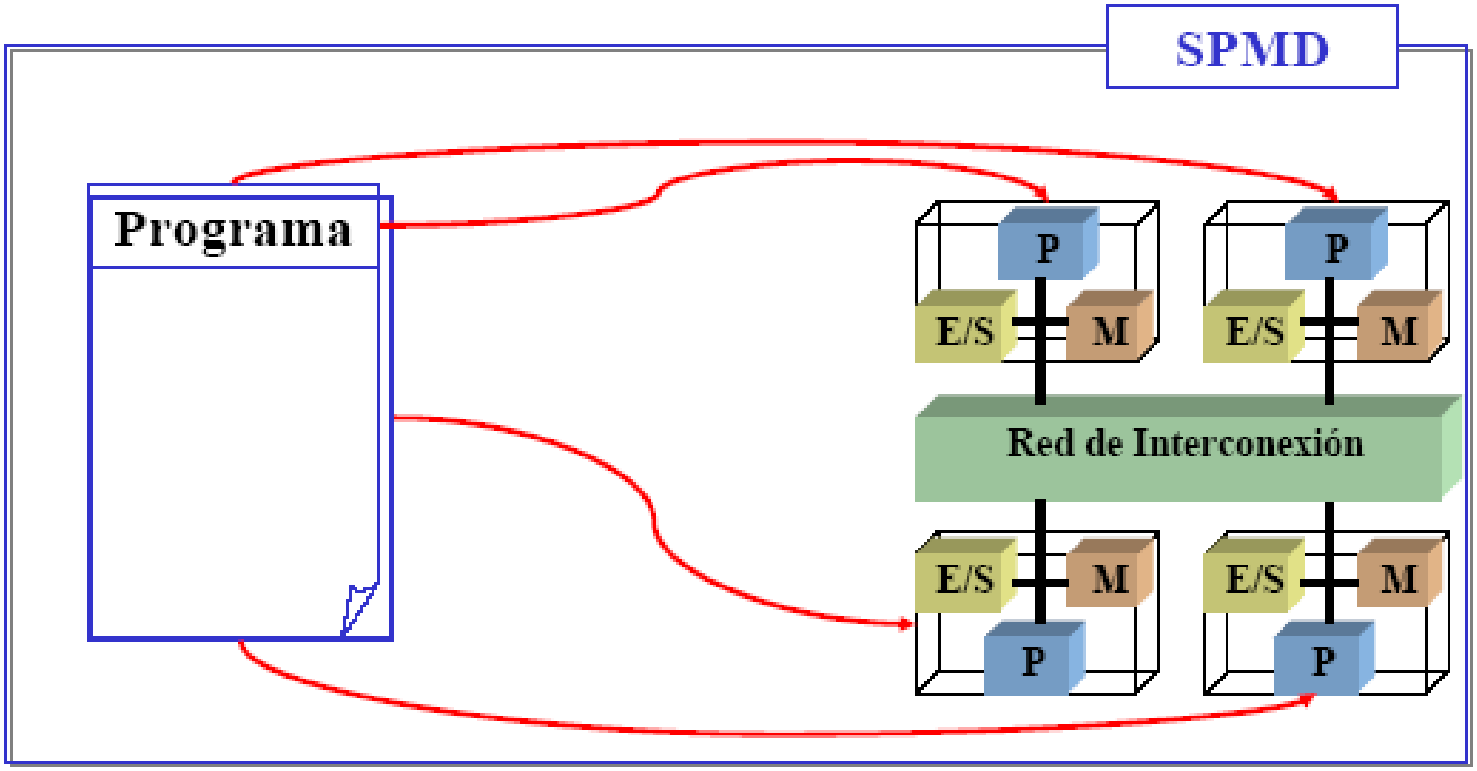
- División en unidades de cómputo independientes (tareas)
- Agrupación de tareas (código y datos) en procesos/hebras
- Asignación a procesadores
- Sincronización y comunicación

Situación inicial (normalmente)

- Se parte de una versión secuencial (no paralela)
- Se parte de una descripción de la aplicación
- Elementos de apoyo:
  - Programa paralelo que resuelva un problema semejante
  - Librerías de funciones paralelas (BLAS, ScalaPack, OpenMP, Intel TBB)

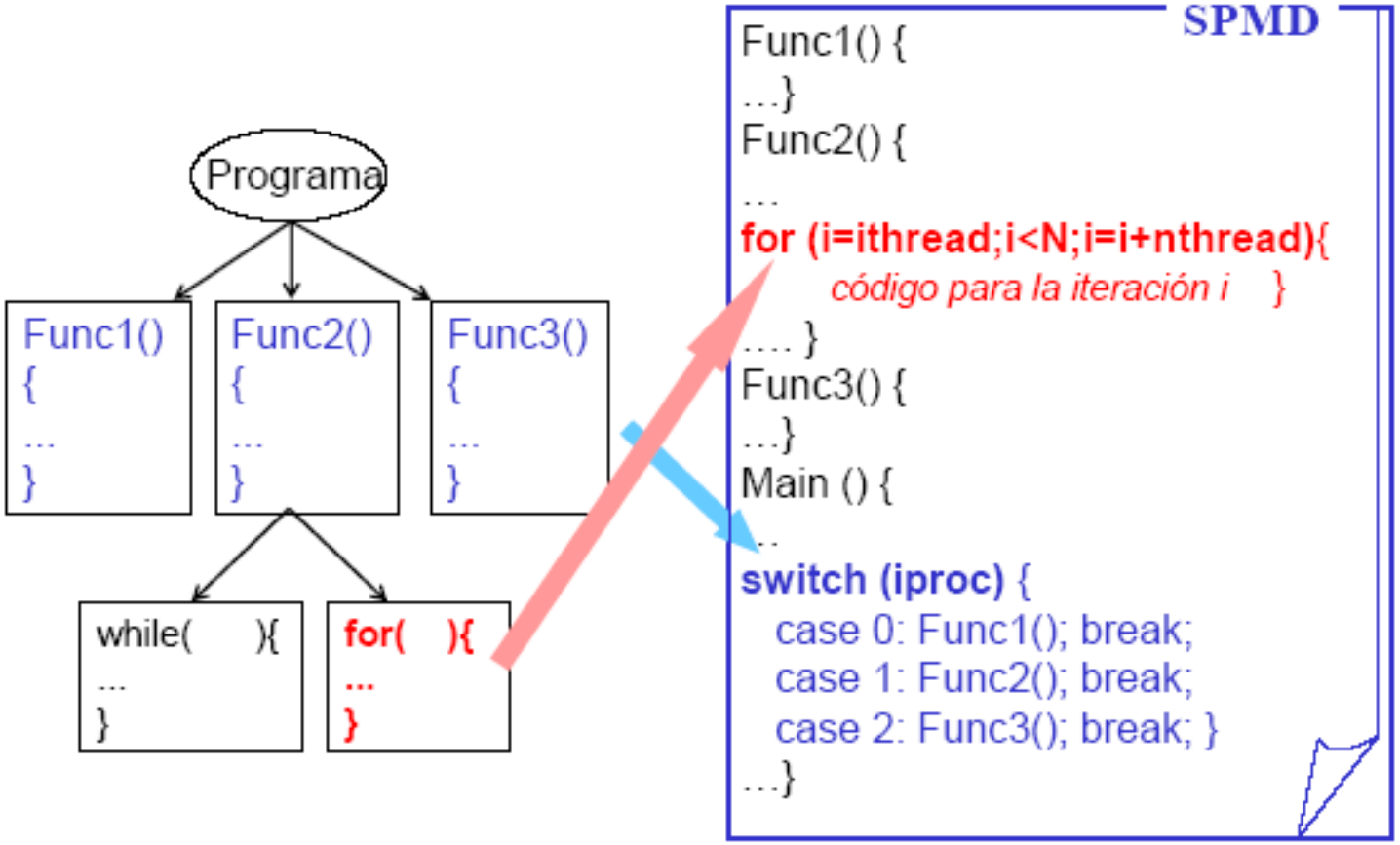
Modos de programación paralela:

- **SPMD** (Single Program Multiple Data)



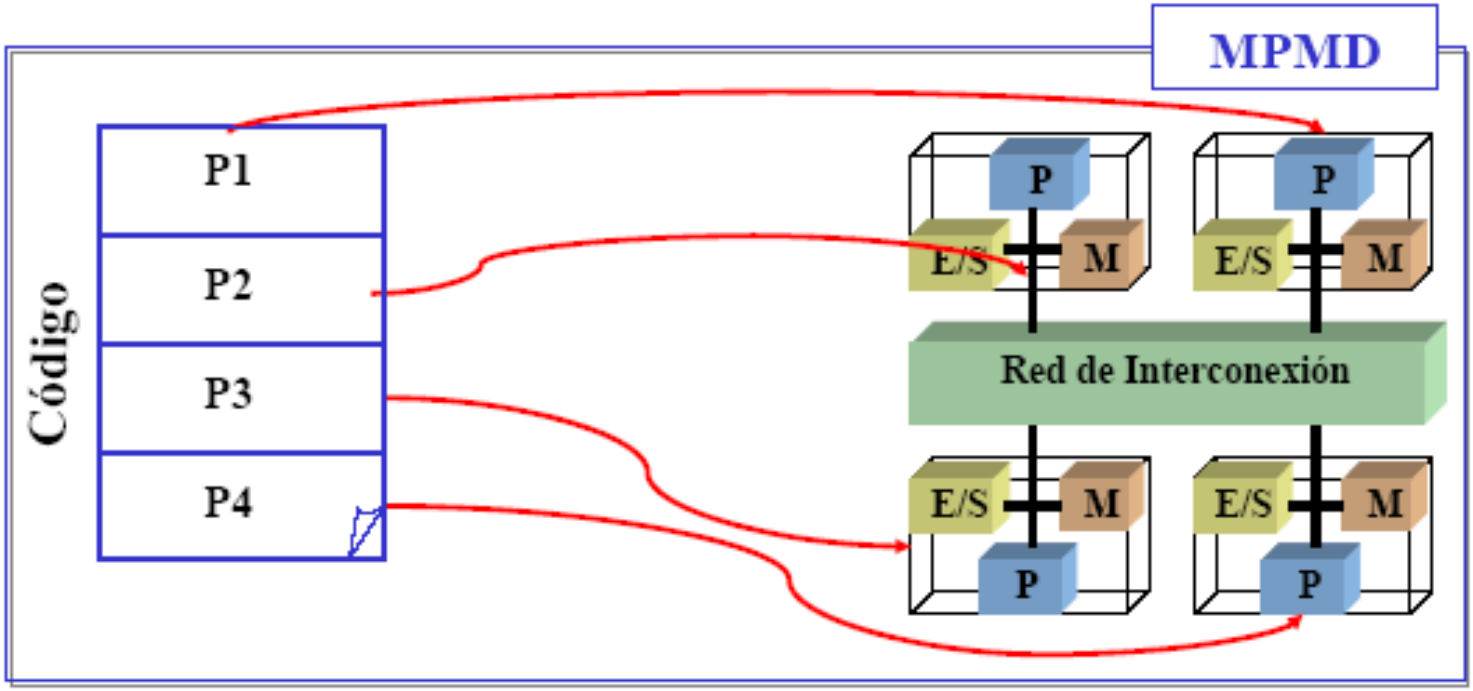
Modos de programación paralela:

- **SPMD** (Single Program Multiple Data). Ejemplo



Modos de programación paralela

- **MPMD** (Multiple Program Multiple Data)

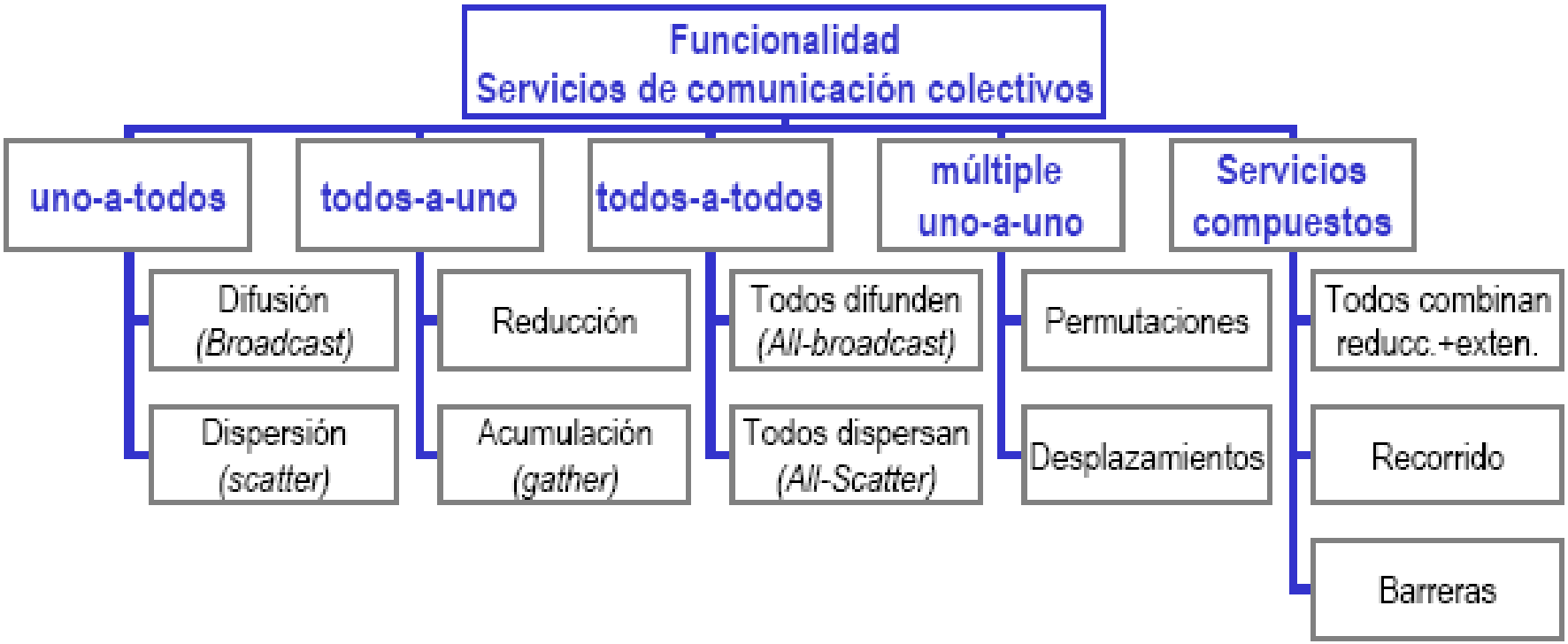


- Modo **Mixto** SPMD-MPMD

Herramientas que facilitan la programación paralela

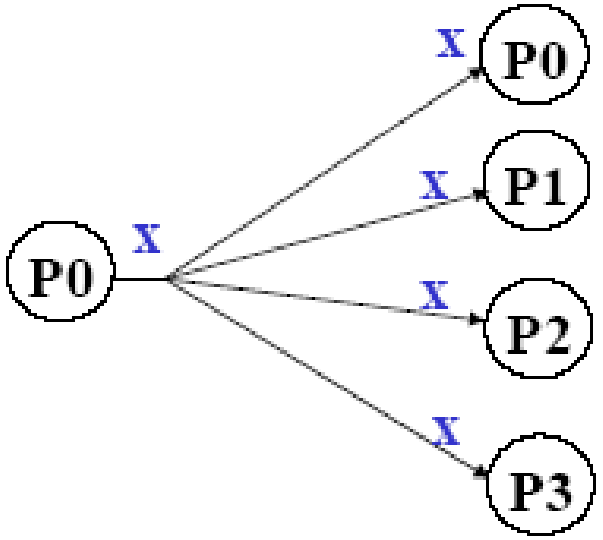
- Compiladores paralelos. Extracción automática del paralelismo
- Directivas del compilador (OpenMP, Intel TBB): lenguaje secuencial + directivas
- Lenguajes paralelos (HPF, Occam, Ada)
- Bibliotecas de funciones (*Pthreads*, MPI, PVM): lenguaje secuencial + funciones de biblioteca como interfaces

Alternativas de comunicación:

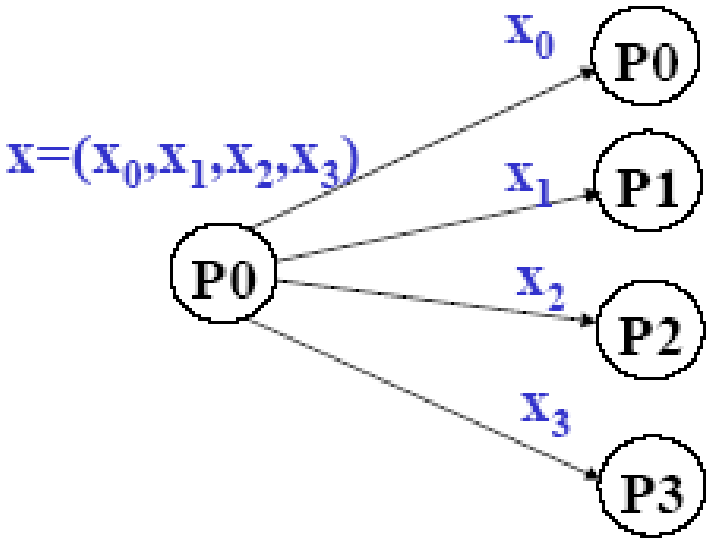


Comunicación: uno a todos

Difusión (*broadcast*)

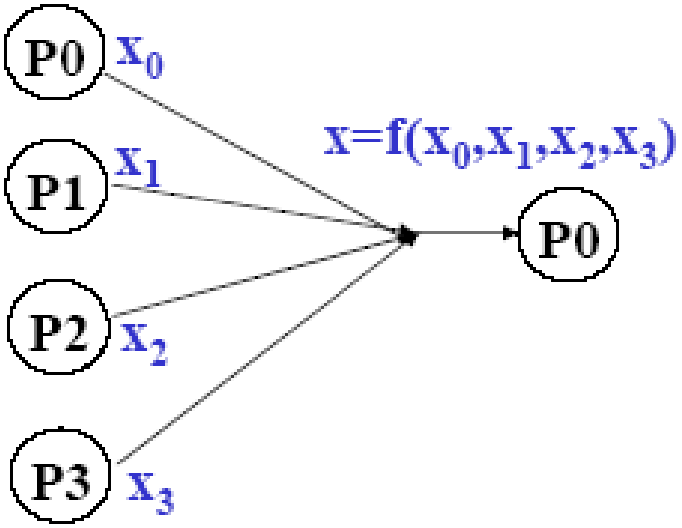


Dispersión (*scatter*)

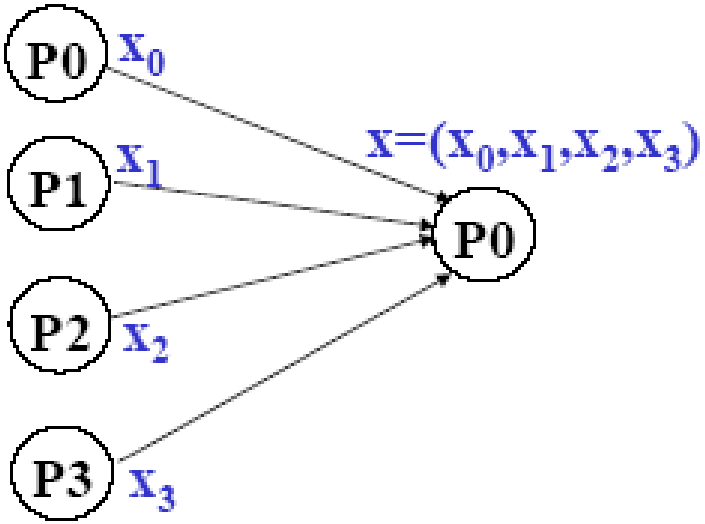


Comunicación: todos a uno

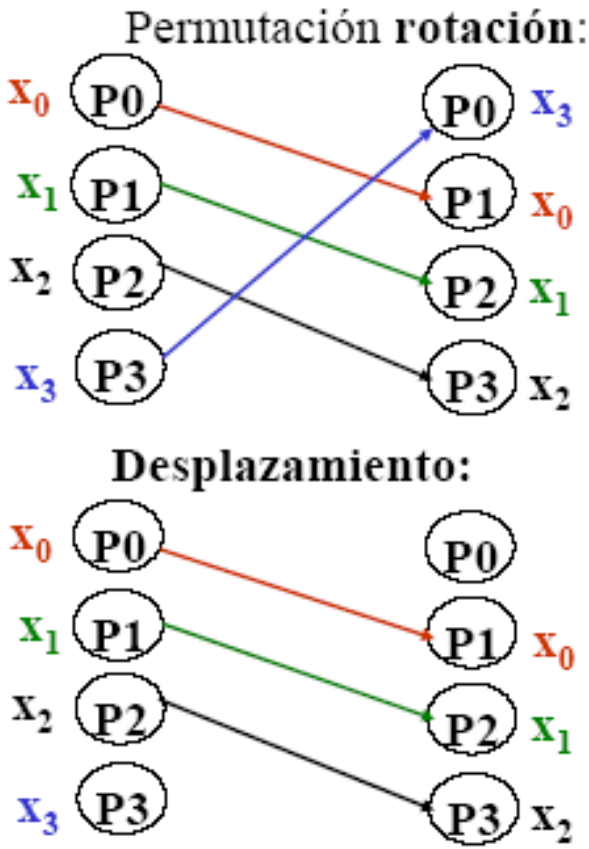
Reducción

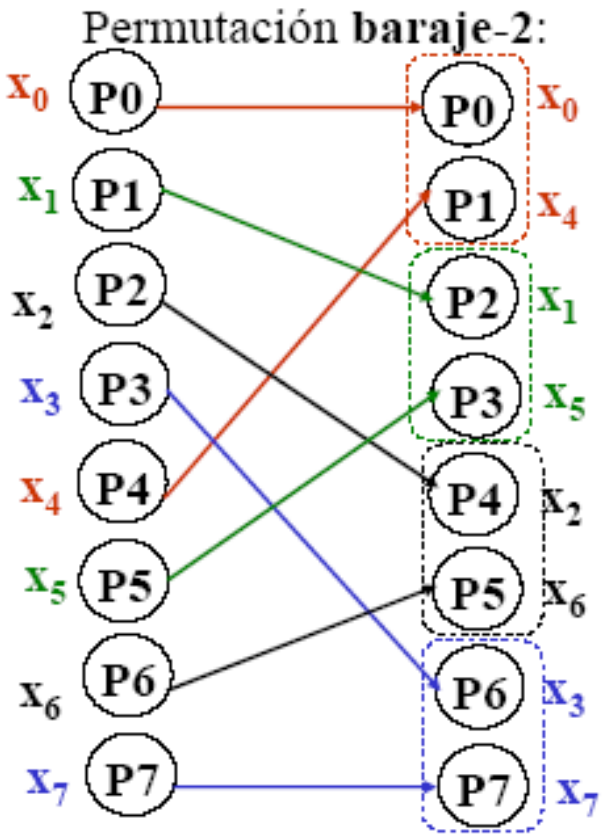


Acumulación (*gather*)



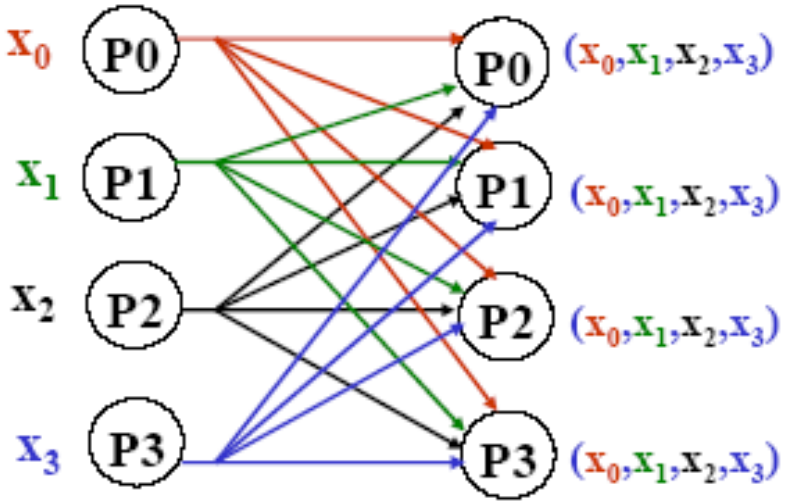
Comunicación: múltiple uno a uno



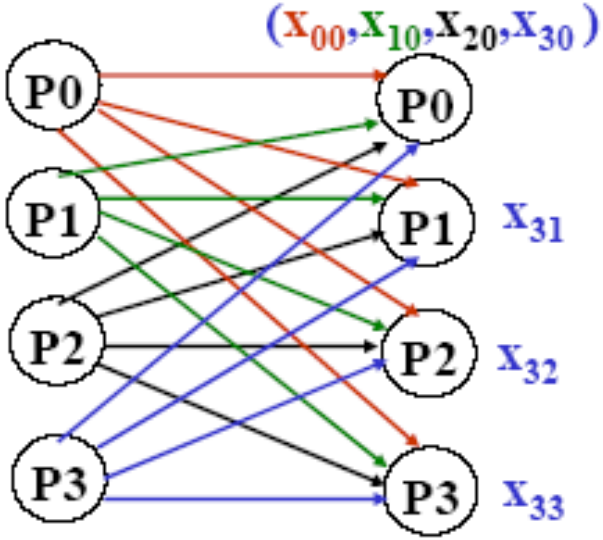


Comunicación: todos a todos

Todos Difunden (*all-broadcast*) o chismorreo (*gossiping*)



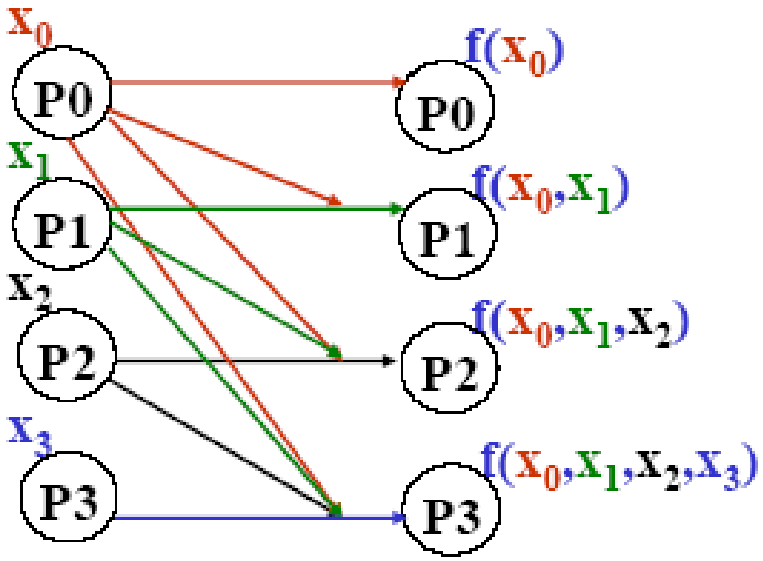
Todos Dispersan (*all-scatter*)



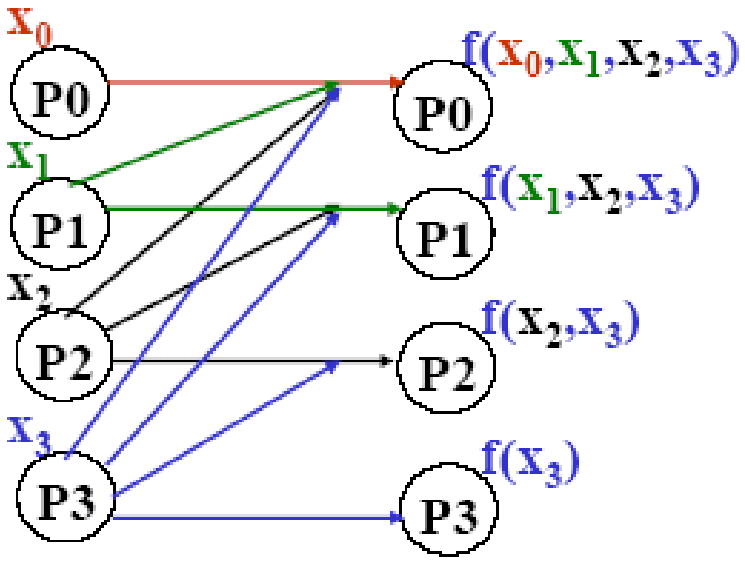
$x=(x_{30},x_{31},x_{32},x_{33})$

Comunicación: servicios compuestos

Recorrido prefijo paralelo



Recorrido sufijo paralelo



Comunicación: Ejemplos de funciones colectivas usando MPI

Uno-a-todos	**Difusión**	MPI_Bcast()
	Dispersión	MPI_Scatter()
Todos-a-uno	**Reducción**	MPI_Reduce()
	Acumulación	MPI_Gather()
Todos-a-todos	**Todos difunden**	MPI_Allgather()
Servicios compuestos	**Todos combinan**	MPI_Allreduce()
	Barreras	MPI_Barrier()
	Scan	MPI_Scan

Estilos de programación paralela: paso de mensajes

- Herramientas software
  - Lenguajes de programación (Ada) y librerías (MPI, PVM)
- Distribución de carga de trabajo
  - Uso de librerías: explícita con construcciones del lenguaje secuencial (if, for, ...)
  - Uso de lenguaje paralelo: explícita con construcciones del propio lenguaje (Occam: sentencia par)
- Primitivas básicas de comunicación : Send y Receive
- Funciones de comunicación colectivas
- Sincronización
  - Receive bloqueante. Barreras (MPI_Barrier)
  - Send-receive bloqueante en ADA

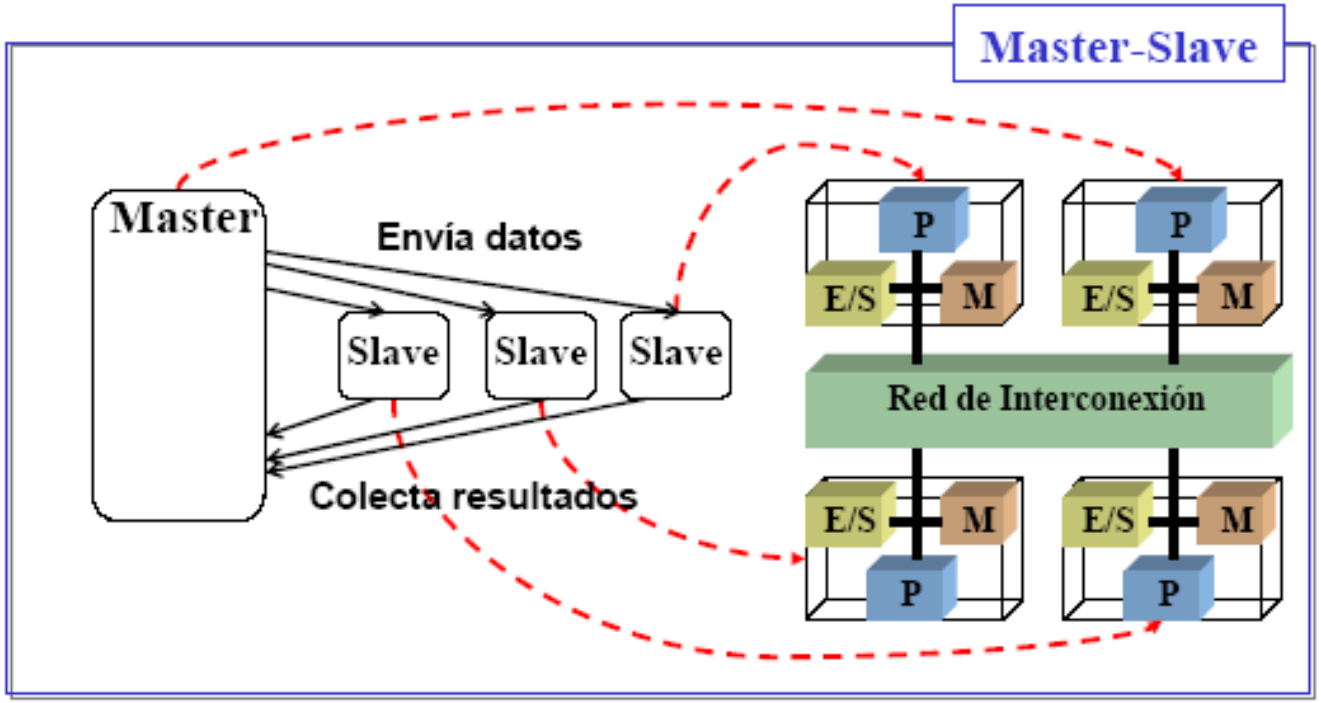
Estilo de programación paralela: variables compartidas

- Lenguajes de programación (Ada), Funciones de librerías (OpenMP), directivas del compilador (OpenMP)
- Distribución de la carga
  - Librerías: explícita con el lenguaje secuencial
  - Directivas: sincronización implícita (mayor nivel de abstracción)
  - Lenguajes: construcciones del lenguaje
- Comunicación básica: load y store
- Funciones de comunicación colectiva
- Sincronización
  - Semáforos, cerrojos, variables condicionales

Estilos de programación paralela: paralelismo de datos

- Herramientas
  - Lenguaje de programación (HPF – High Performance Fortran)
- Distribución de carga
  - Directivas para distribuir datos entre procesadores
  - Directivas para paralelizar bucles
- Comunicación básica: implícita -> A(i) = A(i-1)
- Funciones de comunicación colectivas
  - Implícitas en funciones usadas explícitamente por el programador (rotaciones – CSHIFT)
- Sincronización: implícita

Estructuras de paralelismo: *master-slave*



Estructuras de paralelismo: master slave

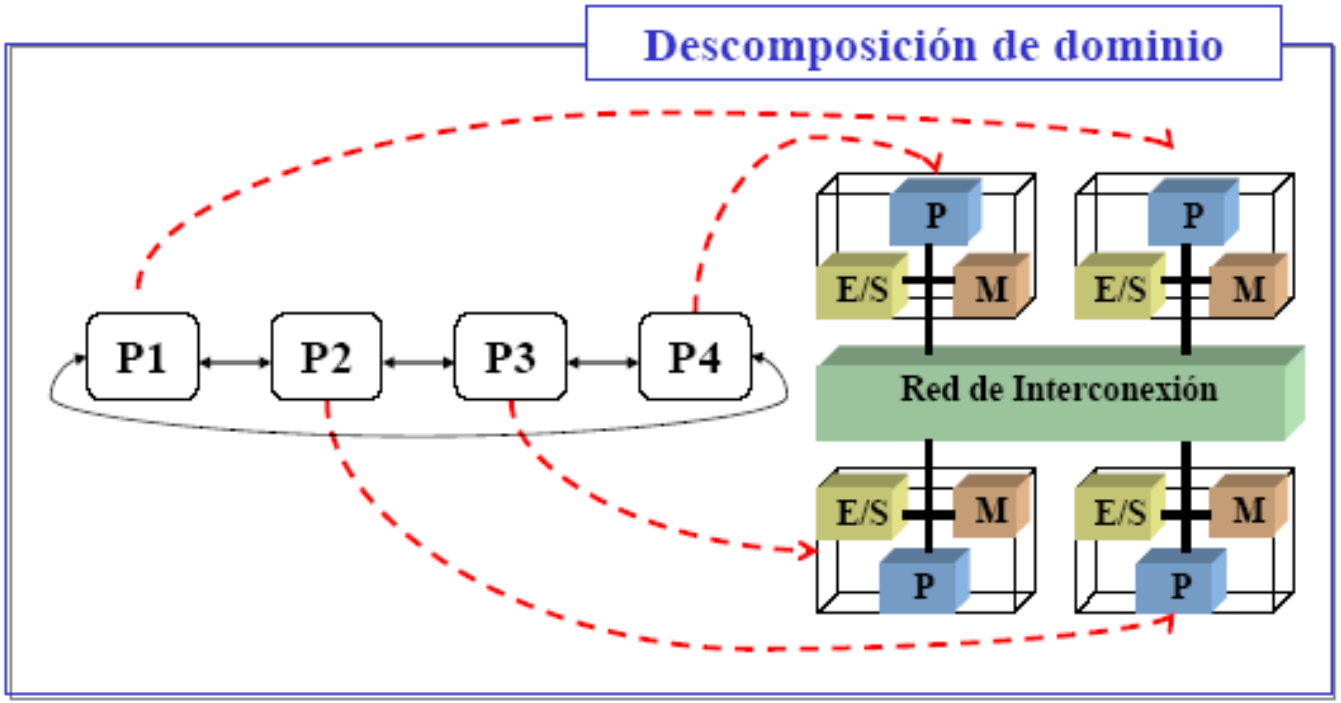
Master-Slave como MPMD– SPMD

```
main ()
{    código Master
}
----------------
main ()
{   código Slaves
}
```

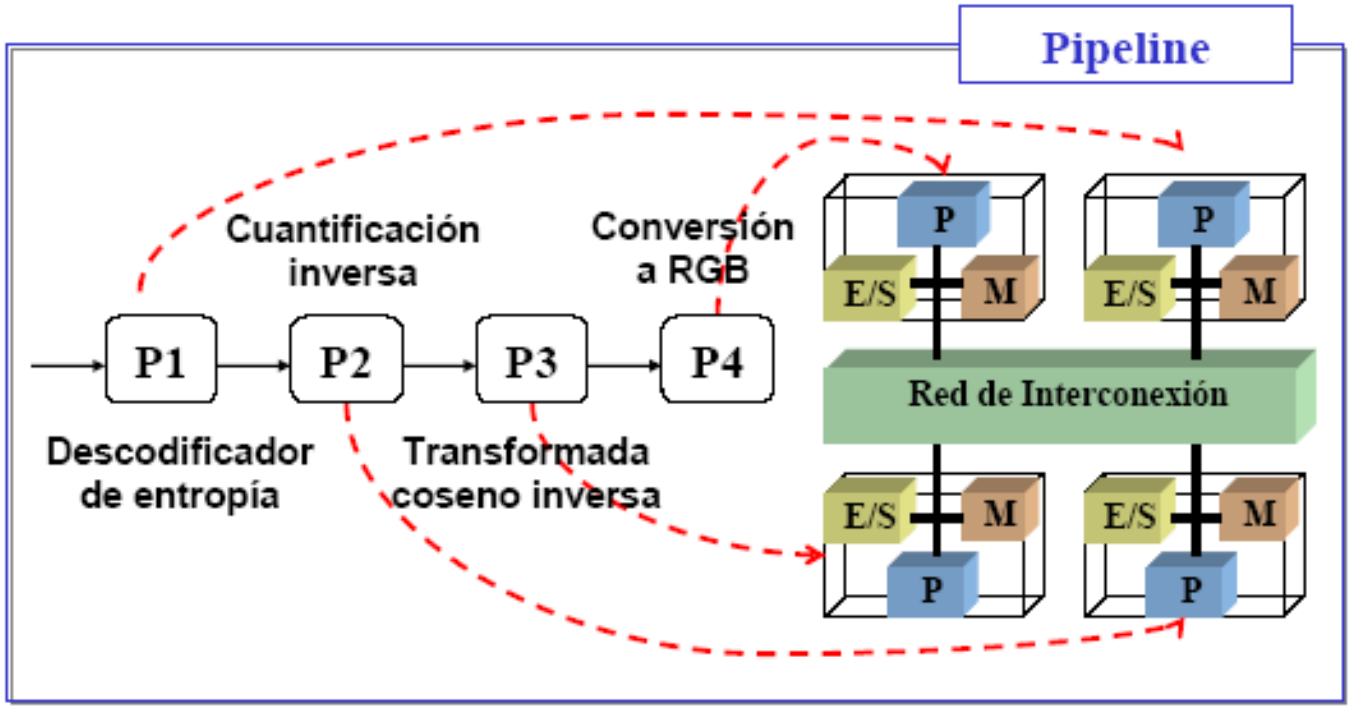
Master-Slave como SPMD

```
main ()
{ if (iproc=id_Master) {
     código Master
  } else {
     código Slaves
  }
}
```

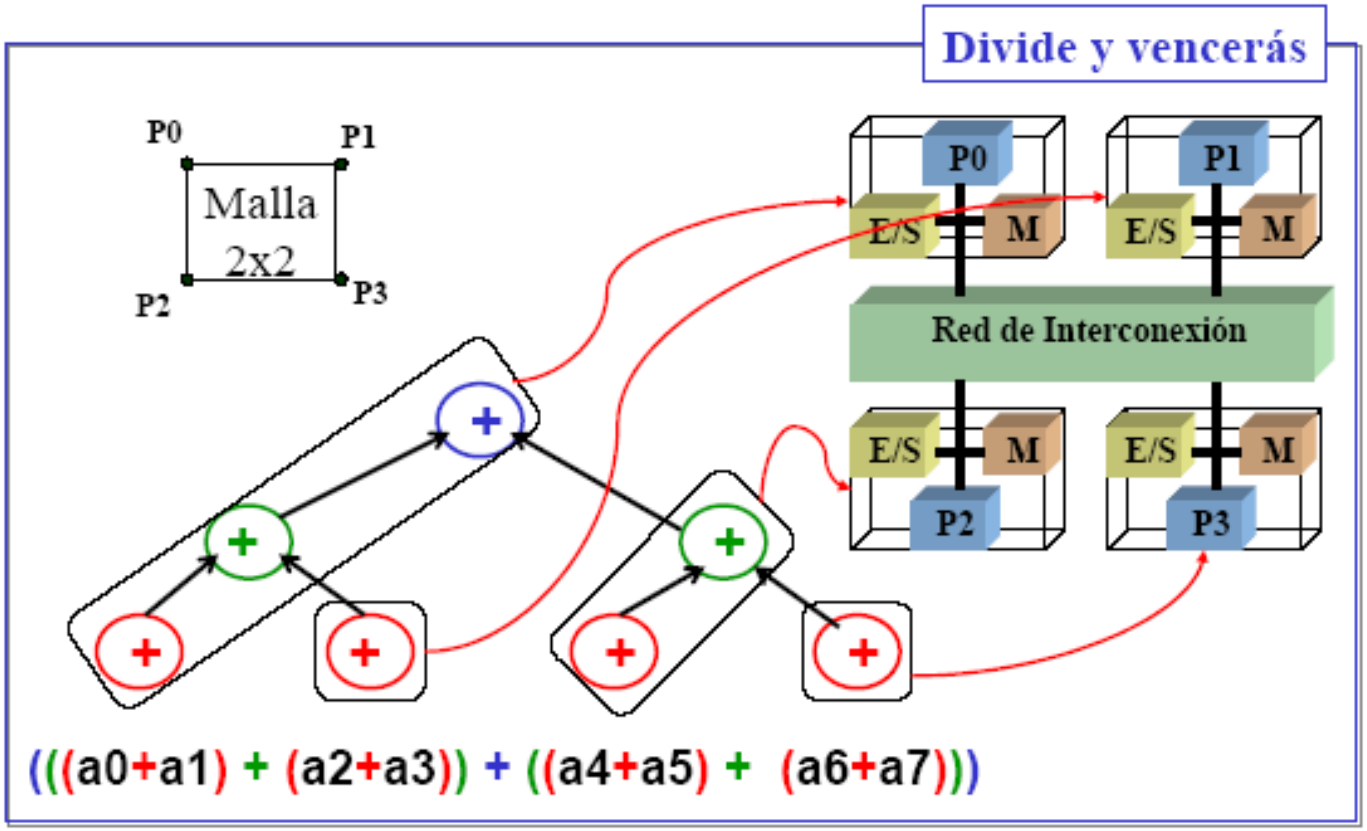
Estructuras de paralelismo: descomposición de dominio (paralelismo de datos)



Estructuras de paralelismo: segmentada (flujo de datos)

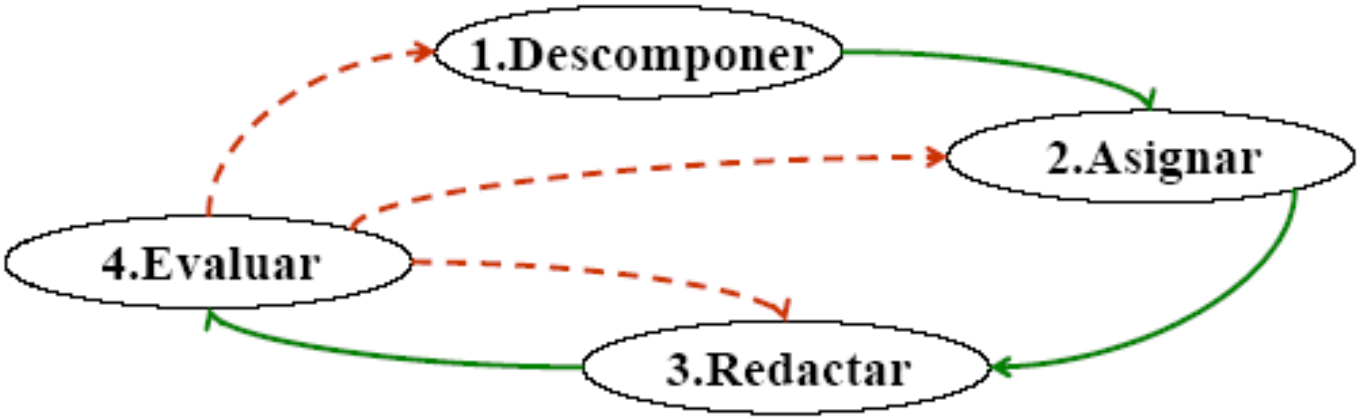


Estructuras de paralelismo: divide y vencerás (descomposición recursiva)

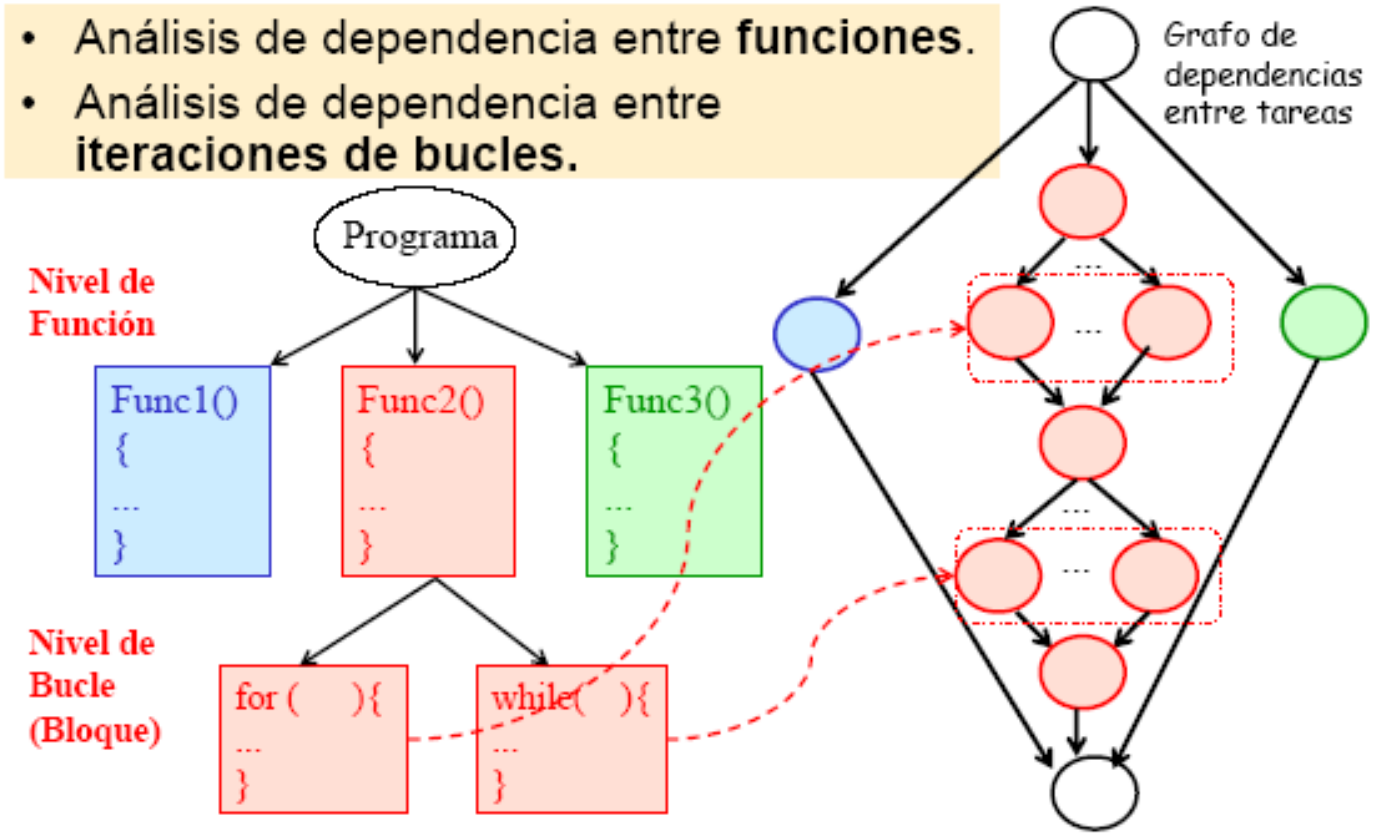


Proceso de paralelización

- Descomposición en tareas independientes
- **Asignación** de tareas a procesos y/o hebras
- Redacción de código paralelo
- Evaluación de prestaciones



Proceso de paralelización: descomposición en tareas independientes

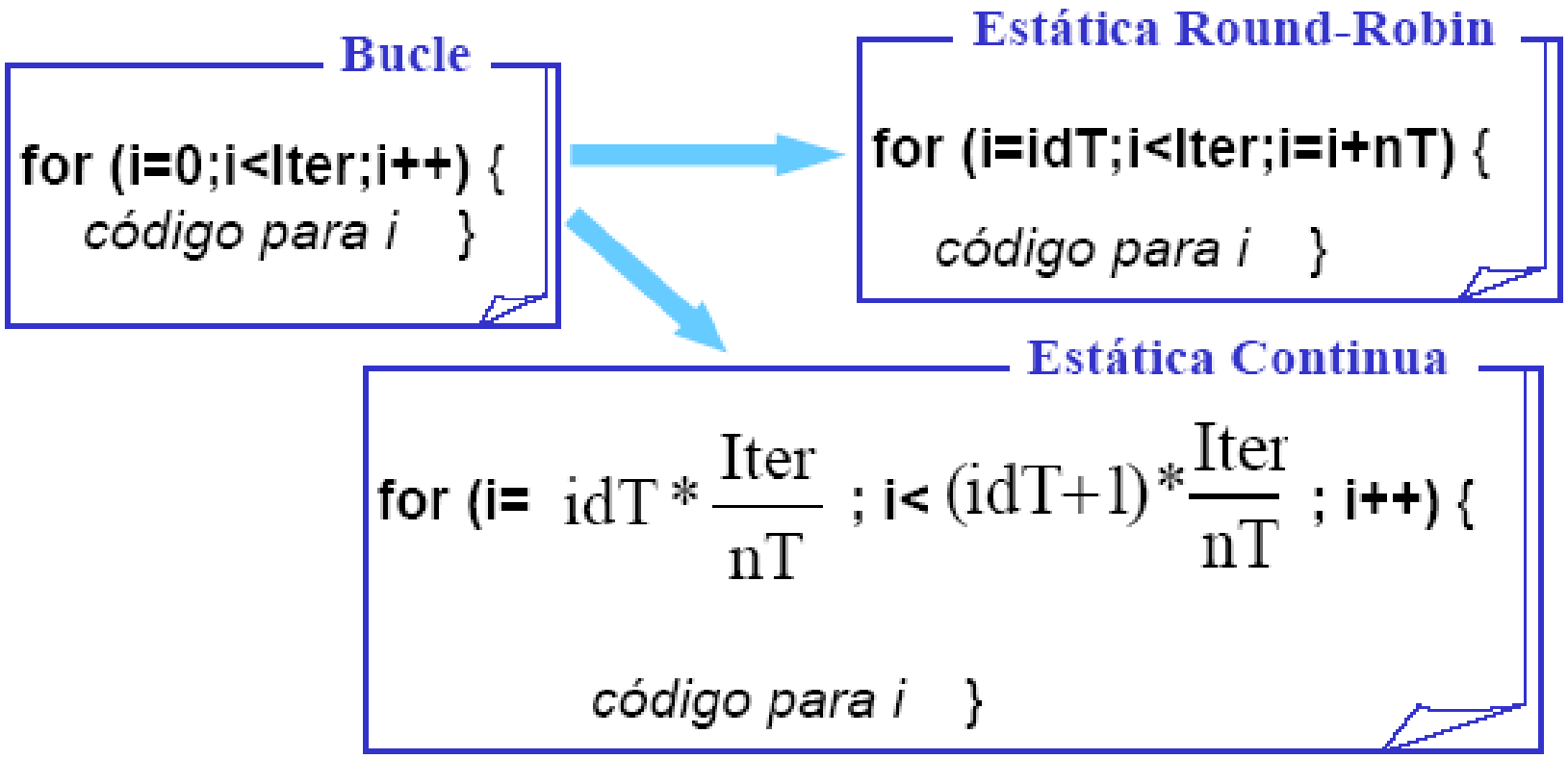


Proceso de paralelización: asignación de tareas

- Normalmente se asignan iteraciones de un ciclo a hebras y funciones a procesos
- La granularidad depende de
  - El número de procesadores
  - El tiempo de comunicación/sincronización frente al tiempo de cálculo
- Equilibrio en la carga de trabajo (que unos procesadores no esperen a otros)
- Tipos de asignación
  - Dinámica (en tiempo de ejecución). Si no se conoce el número de tareas
  - Estática (programador o compilador)

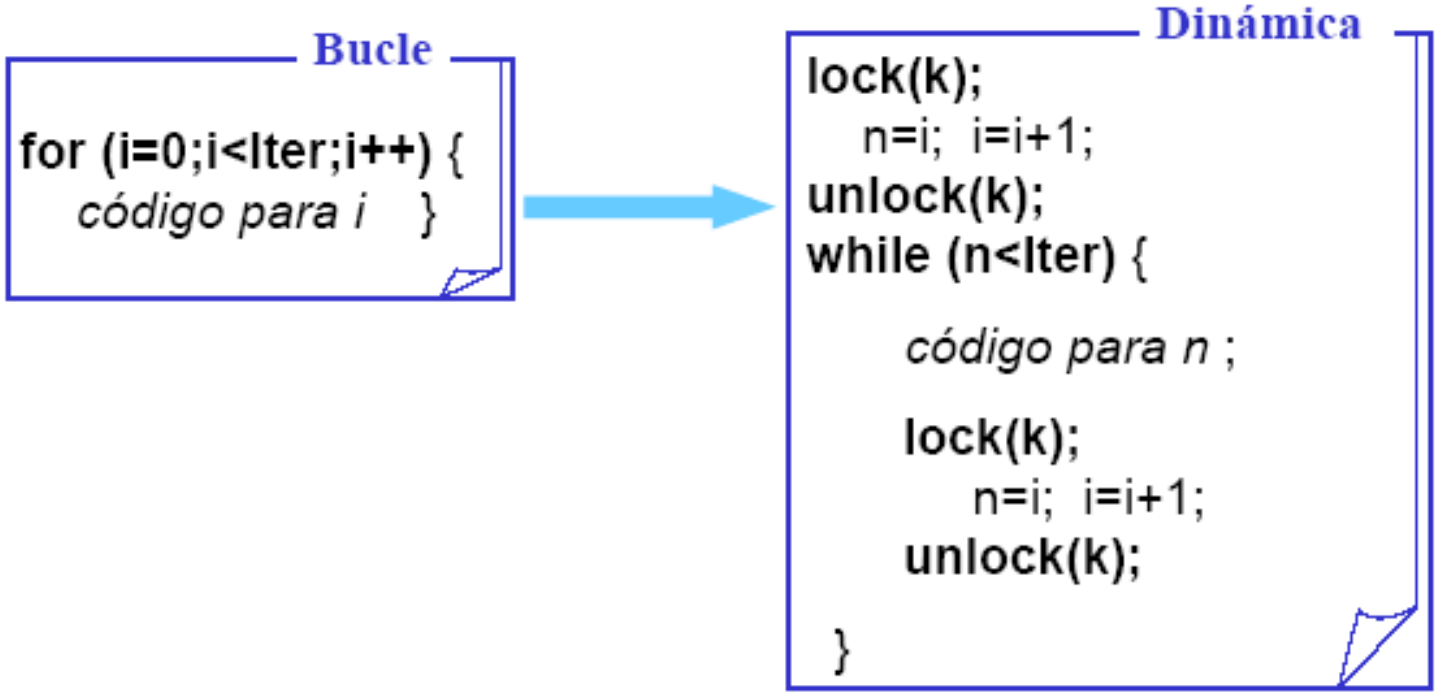
Proceso de paralelización: asignación de tareas

- Asignación estática



Proceso de paralelización: asignación de tareas

- Asignación dinámica



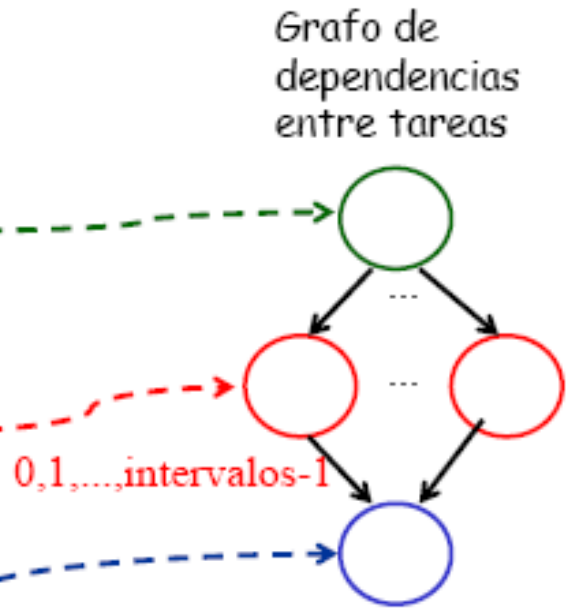
Proceso de paralelización: redactar código paralelo

- Depende de:
  - Estilo de programación: paso de mensajes, etc.
  - Modo de programación
  - Situación inicial
  - Herramienta utilizada para explicitar el paralelismo
  - Estructura del programa

- Un programa paralelo debe incluir
  - Creación y destrucción de procesos/hebras
  - Asignación de carga de trabajo
  - Comunicación y sincronización

Cálculo de Pi por descomposición de tareas

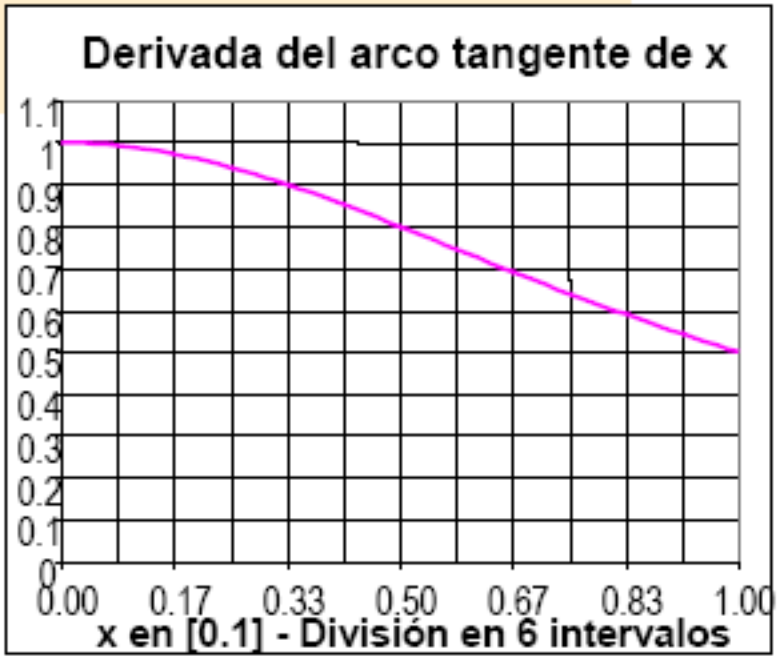
```
main(int argc, char **argv) {
double ancho, sum;
int intervalos, i;

   intervalos = atoi(argv[1]);
   ancho = 1.0/(double) intervalos;
   for (i=0;i< intervals; i++){
         x = (i+0.5)*ancho;
         sum = sum + 4.0/(1.0+x*x);
   }
   sum* = ancho;
}
```



Cálculo de Pi por descomposición de tareas

$$\left.\begin{aligned} \text{arctg}'(x) &= \frac{1}{1+x^2} \\ \text{arctg}(1) &= \frac{\pi}{4} \\ \text{arctg}(0) &= 0 \end{aligned}\right\} \Rightarrow \int_0^1 \frac{1}{1+x^2} = \text{arctg}(x)\,|_0^1 = \frac{\pi}{4} - 0$$

- PI se puede calcular por integración numérica.



- Cálculo de Pi por descomposición de tareas
  - Asignación estática de iteraciones del bucle (asignación *Round Robin*)
  - Redacción de código paralelo
    - Estilo de programación: paso de mensajes
    - Modo de programación: SPMD
    - Situación inicial: versión secuencial
    - Herramienta: MPI
    - Estructura del programa: paralelismo de datos o divide y vencerás

Cálculo de Pi por descomposición de tareas

```
#include <mpi.h>
main(int argc, char **argv)  {
double ancho,x, sum, tsum; int intervalos, i;  int nproc, iproc;
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) exit(1);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
    intervalos = atoi(argv[1]);   ancho = 1.0 /(double) intervalos;  lsum = 0;
    for (i=iproc; i<intervalos; i+=nproc) {
            x = (i + 0.5) * ancho;  lsum += 4.0 / (1.0 + x * x);
    }
    lsum *= ancho;
    MPI_Reduce(&tsum, &sum, 1, MPI_DOUBLE,
                        MPI_SUM,0,MPI_COMM_WORLD);
    MPI_Finalize();
}
```

Enrolar

Asignar/
Localizar

Comunicar/
sincronizar

Desenrolar

Cálculo de Pi por descomposición de tareas

- Asignación dinámica de iteraciones del bucle
- Redacción de código paralelo
  - Estilo de programación: directivas
  - Modo de programación: SPMD
  - Situación inicial: versión secuencial
  - Herramienta: OpenMP
  - Estructura del programa: paralelismo de datos o divide y vencerás

Cálculo de Pi por descomposición de tareas

```
#include <omp.h>
#define NUM_THREADS 4
main(int argc, char **argv)  {
    double ancho,x, sum;  int intervalos, i;
    intervalos = atoi(argv[1]);   ancho = 1.0/(double) intervalos;
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
    #pragma omp for reduction(+:sum) private(x)
                    schedule(dynamic)
            for (i=0;i< intervalos; i++) {
                x = (i+0.5)*ancho; sum = sum + 4.0/(1.0+x*x);
            }
    sum* = ancho;
}
```

Crear/ Terminar

Comunicar/ sincronizar

Localizar

Asignar

Ejemplos del Top500

#1 TOP500 desde 2013 (10/2019) - Tianhe 2A (Milkiway-2)

- Fabricante: NUDT (*National University of Defense Technology*)
- #Cores: 4.981.760
- OS: Linux
- Red de interconexión: propietaria
- Máximo rendimiento: 61,444 TF
- Pico de rendimiento: 100,678 TF
- Potencia disipada: 18,482 kW
- Situación: *National Supercomputing Center* en Guangzhou
- http://www.nscc-tj.gov.cn/en/
- Especificación detallada: https://www.top500.org/site/50365

#5 TOP500 (6/2012) – Tianhe 1A

- #1 TOP500 (6/2012)
- Fabricante: NUDT (National University of Defense Technology)
- #cores: 186368
- OS: Linux
- Red de interconexión: propietaria
- Rendimiento máximo: 2566 TF
- Rendimiento pico: 4701 TF
- Potencia disipada: 4040 kW
- Situsción: *National Supercomputing Center* en Tianjin
- http://www.nscc-tj.gov.cn/en/
- Especificaciones detalladas: http://www.nscc-tj.gov.cn/en/resources/resources_1.asp

Introducción

- Visita a la web del Top500: https://www.top500.org

Prácticas

- Competencias adquiridas en las prácticas de la asignatura
- Debate: técnicas y problemas en la evaluación del rendimiento de una aplicación paralela.