

2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores

Tema 2

Programación paralela

Material elaborado por los profesores responsables de la asignatura:

Mancia Anguita – Julio Ortega

Licencia Creative Commons



ugr

Universidad
de Granada

ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



ATC

Departamento de Arquitectura
y Tecnología de Computadores
UNIVERSIDAD DE GRANADA



Lecciones

- Lección 4. Herramientas, estilos y estructuras en programación paralela
 - Problemas que plantea la programación paralela al programador. Punto de partida
 - Herramientas para obtener código paralelo
 - Estilos/paradigmas de programación paralela
 - Estructuras típicas de códigos paralelos
- Lección 5. Proceso de paralelización
- Lección 6. Evaluación de prestaciones en procesamiento paralelo

Objetivos Lección 4

- Distinguir entre los diferentes tipos de herramientas de programación paralela: compiladores paralelos, lenguajes paralelos, API Directivas y API de funciones.
- Distinguir entre los diferentes tipos de comunicaciones colectivas.
- Diferenciar el estilo/paradigma de programación de paso de mensajes del de variables compartidas.
- Diferenciar entre OpenMP y MPI en cuanto a su estilo de programación y tipo.
- Distinguir entre las estructuras de tareas/procesos/treads master-slave, cliente-servidor, descomposición de dominio, flujo de datos o segmentación, y divide y vencerás.

Bibliografía Lección 4

➤ Fundamental

- Capítulo 7. Sección 7.4. J. Ortega, M. Anguita, A. Prieto. “Arquitectura de Computadores”. ESII/C.1 ORT arq

➤ Complementaria

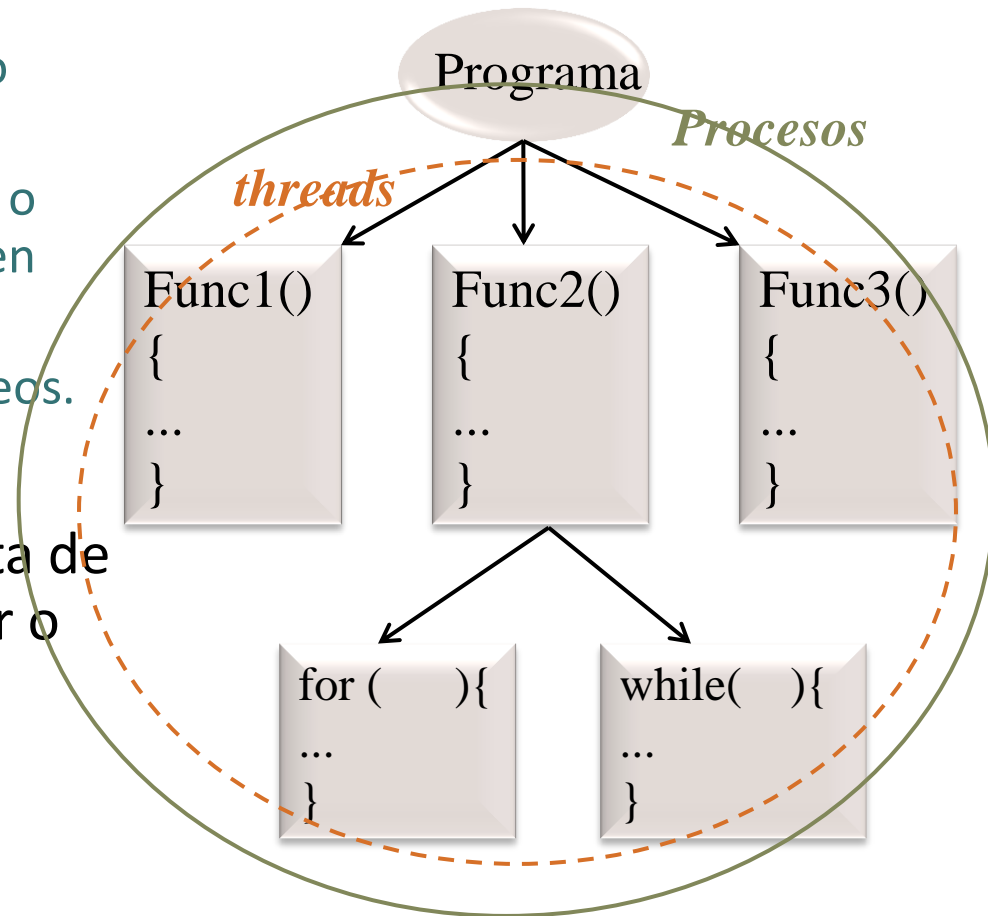
- Thomas Rauber, Gudula Rünger. “Parallel Programming: for Multicore and Cluster Systems.” Springer, 2010. Disponible en línea (biblioteca UGR): <http://dx.doi.org/10.1007/978-3-642-04818-0>
- Barry Wilkinson. “Parallel programming : techniques and applications using networked workstations and parallel computer”, 2005. ESIIT/D.1 WIL par

Contenido Lección 4

- Problemas que plantea la programación paralela al programador. Punto de partida
- Herramientas para obtener código paralelo
- Estilos/paradigmas de programación paralela
- Estructuras típicas de códigos paralelos

Problemas que plantea la programación paralela

- Nuevos problemas, respecto a programación secuencial:
 - División en unidades de cómputo independientes (tareas).
 - Agrupación/asignación de tareas o carga de trabajo (código, datos) en procesos/threads.
 - Asignación a procesadores/núcleos.
 - Sincronización y comunicación.
- Los debe abordar la herramienta de programación o el programador o ambos

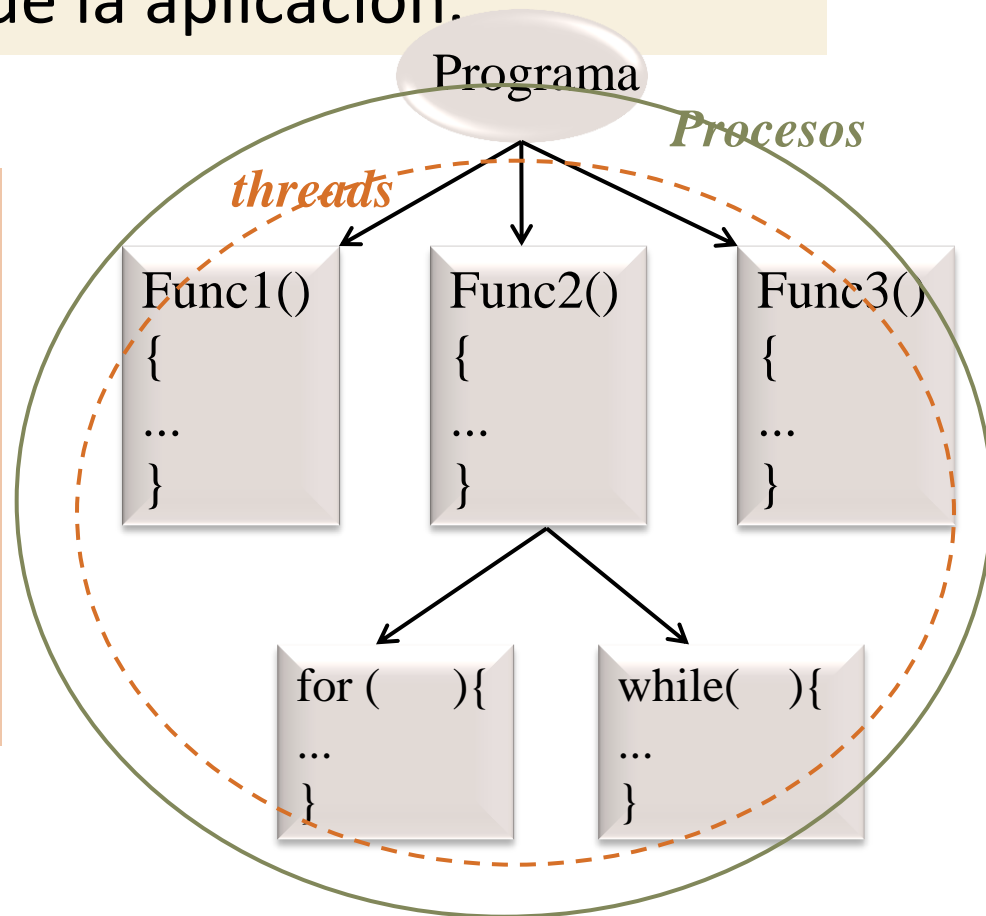


Punto de partida

- Partir de una versión secuencial.
- Descripción o definición de la aplicación.

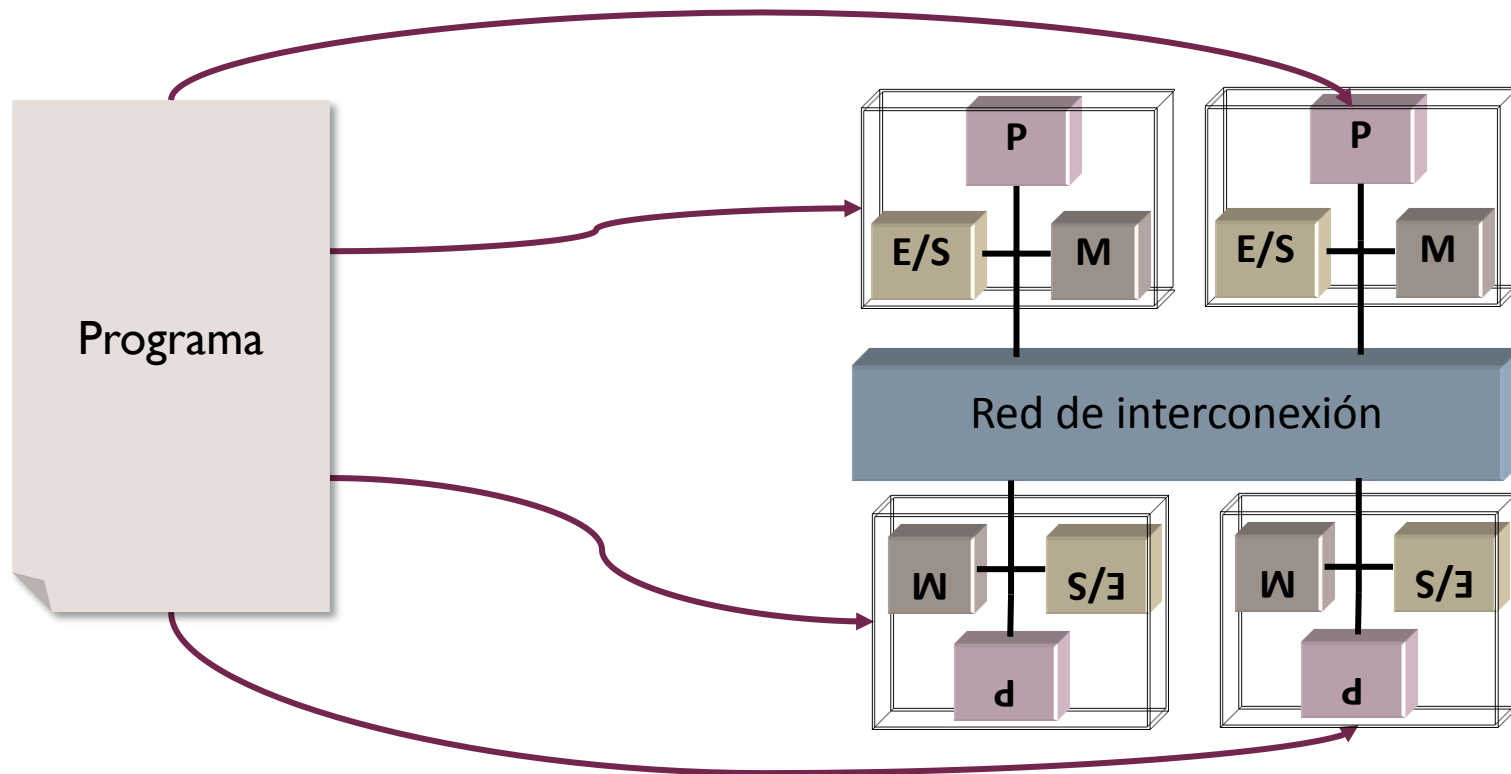
Apoyo:

- Programa paralelo que resuelva un problema parecido.
- Versiones paralelas u optimizadas de bibliotecas de funciones:
BLAS, LAPACK, ...



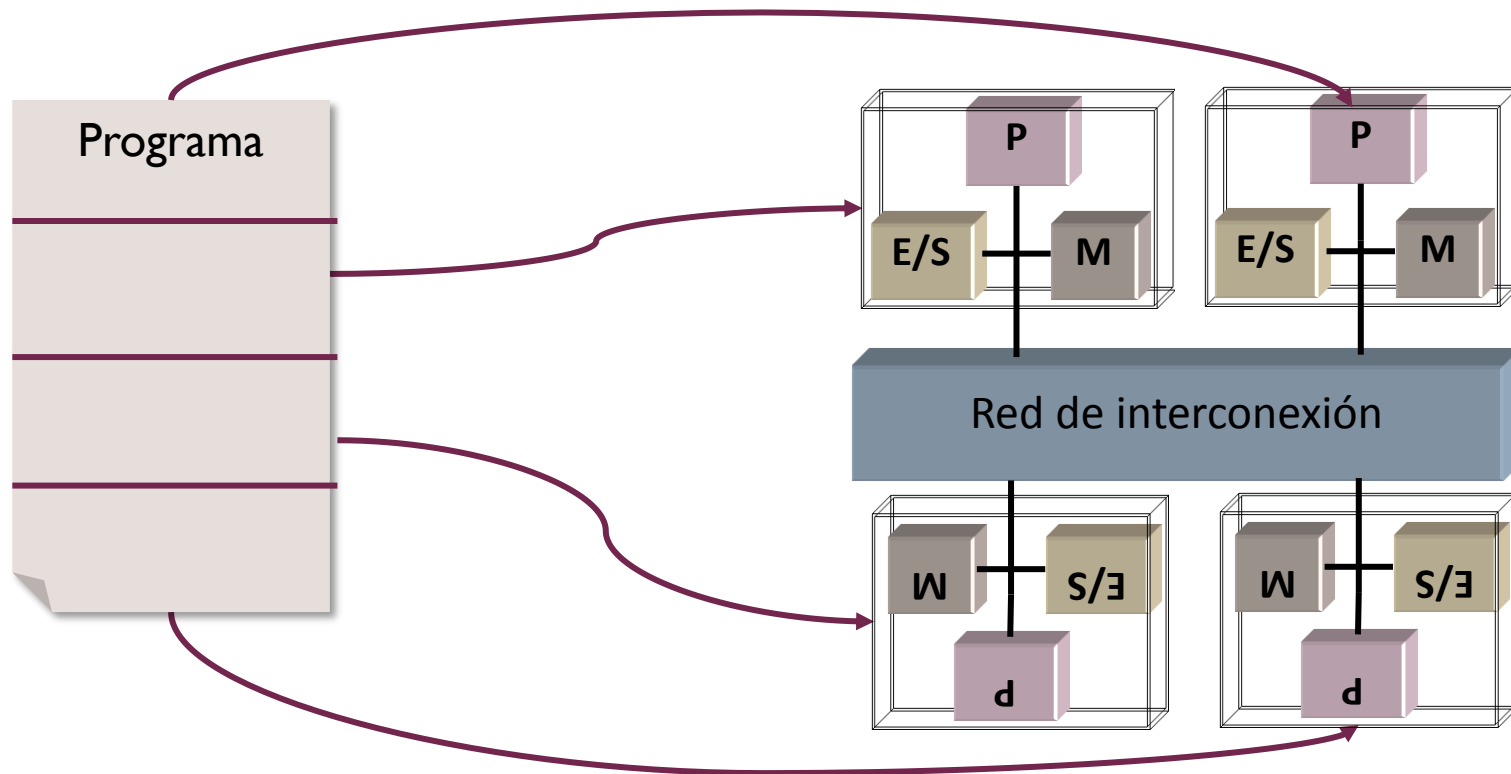
Modos de programación MIMD

- SPMD (*Single-Program Multiple Data*)



Modos de programación MIMD

- MPMD (*Multiple-Program Multiple Data*)



Contenido Lección 4

- Problemas que plantea la programación paralela al programador. Punto de partida
- Herramientas para obtener código paralelo
- Estilos/paradigmas de programación paralela
- Estructuras típicas de códigos paralelos

Herramientas de programación paralela

Compiladores paralelos (paralelización automática)

Extracción automática del paralelismo

Lenguajes paralelos (Occam, Ada, Java) y API funciones + Directivas (OpenMP)

Construcciones del lenguaje + funciones

Lenguaje secuencial + directivas +
funciones

API funciones (Pthreads, MPI)

Lenguaje secuencial + funciones

Herramientas para obtener programas paralelos

- Las herramientas permiten de forma implícita o explícita (lo hace el programador):
 - Localizar paralelismo o descomponer en tareas independientes (*descomposition*)
 - Asignar las tareas, es decir, la carga de trabajo (código + datos), a procesos/threads (*scheduling*)
 - Crear y terminar procesos/threads (o enrolar y desenrolar en un grupo)
 - Comunicar y sincronizar procesos/threads
- El programador, la herramienta o el SO se encarga de
 - Asignar procesos/threads a unidades de procesamiento (*mapping*)

Ejemplo: cálculo de PI con OpenMP/C

```
#include <omp.h>
#define NUM_THREADS 4
main(int argc, char **argv) {
    double ancho,x, sum=0;  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;
}
```

Localizar

Crear y Terminar

Comunicar y sincronizar

Agrupar/Asignar

Ejemplo: cálculo de PI en MPI/C

```
#include <mpi.h>

main(int argc, char **argv) {
    double ancho,x,sum,lsum; int intervalos,i,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(&lsum, &sum, 1, MPI_DOUBLE,
               MPI_SUM,0,MPI_COMM_WORLD);
    MPI_Finalize();
}
```

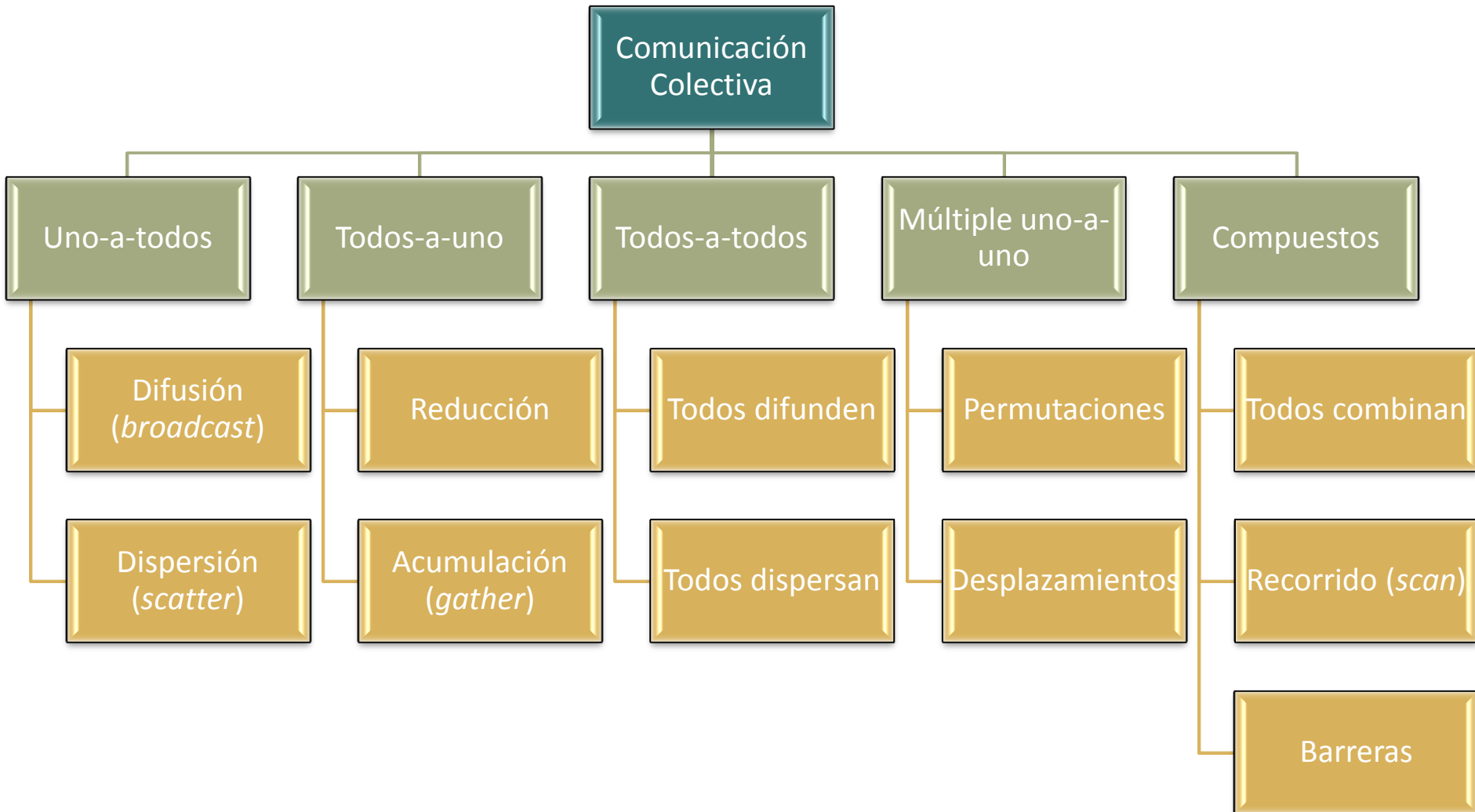
Enrolar

Localizar y Asignar

Comunicar/sincronizar

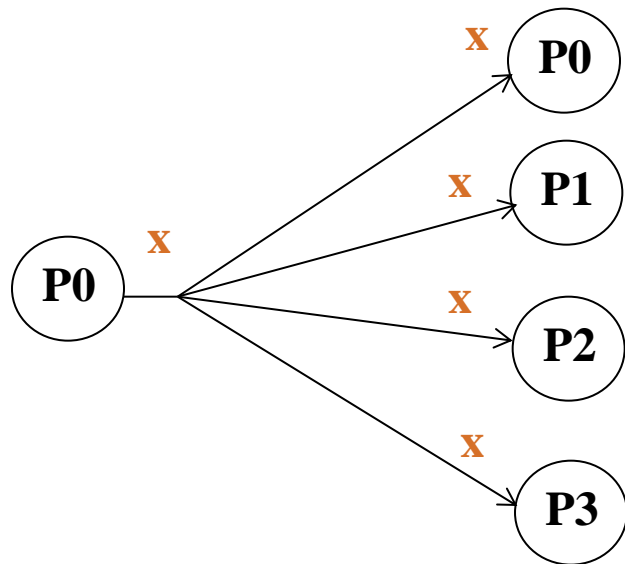
Desenrolar

Comunicaciones colectivas

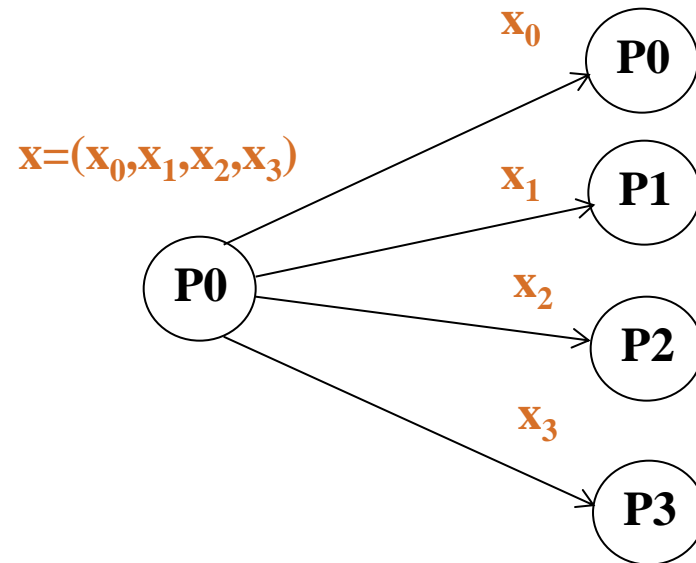


Comunicación uno-a-todos

Difusión (*broadcast*)

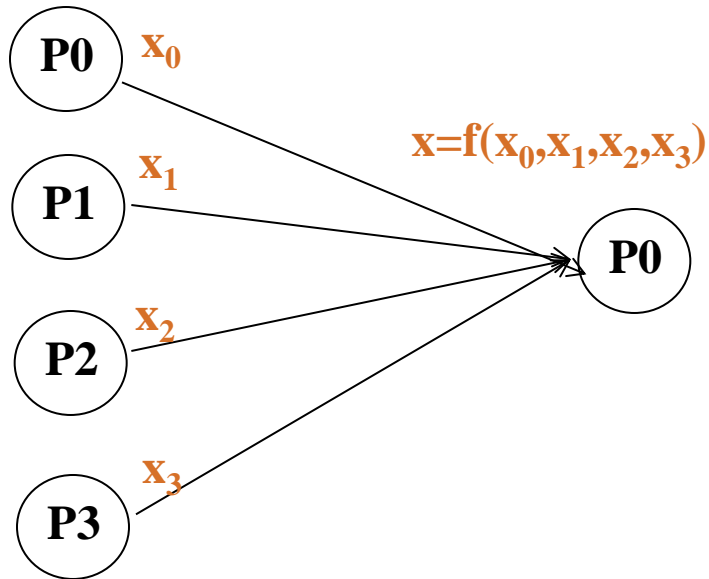


Dispersión (*scatter*)

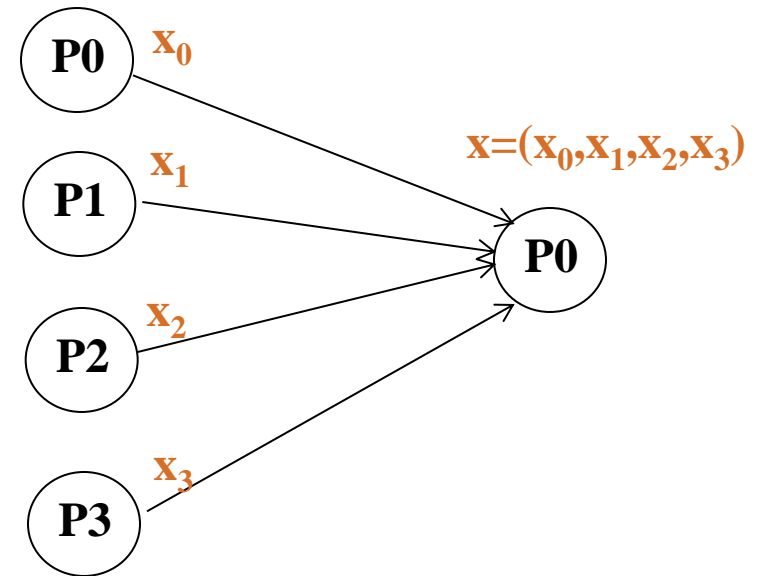


Comunicación todos-a-uno

Reducción

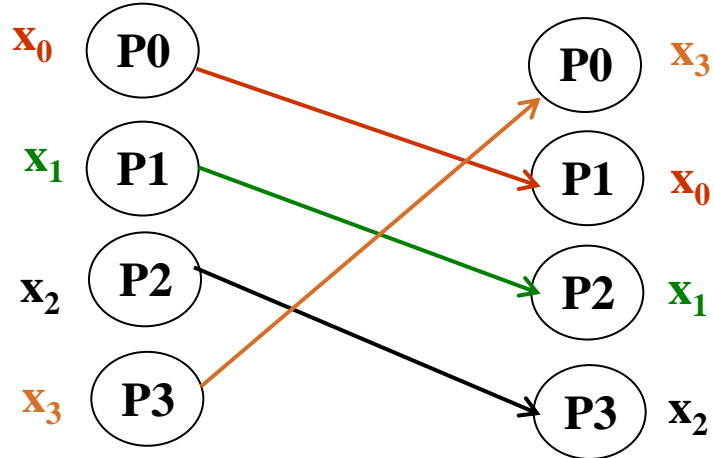


Acumulación (*gather*)

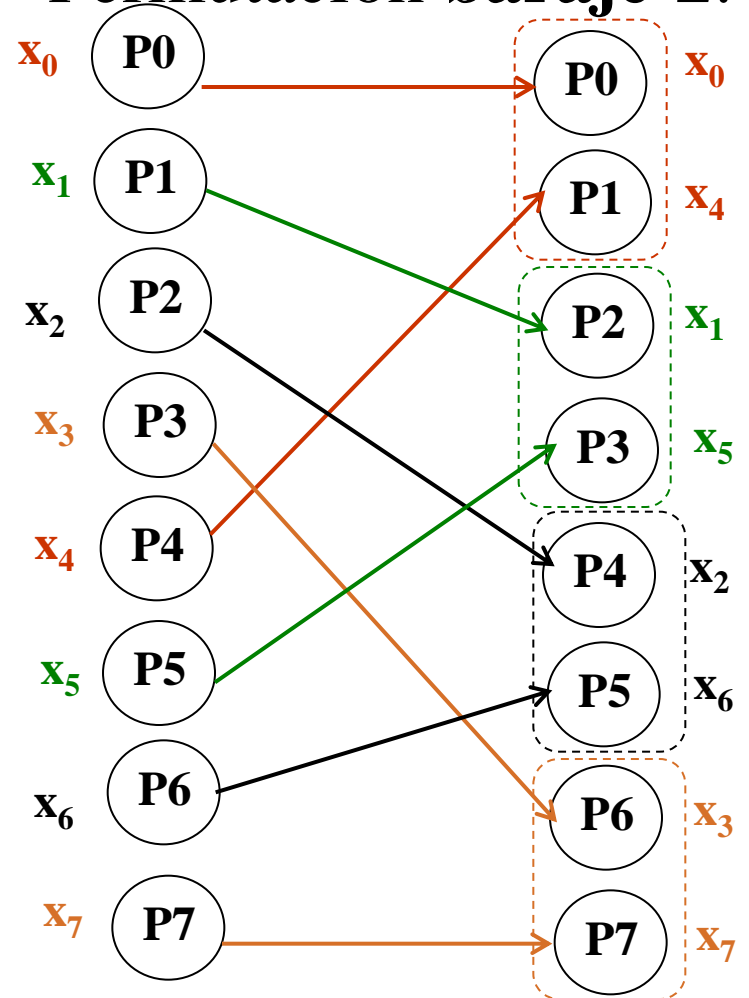


Comunicación múltiple uno-a-uno

Permutación rotación:

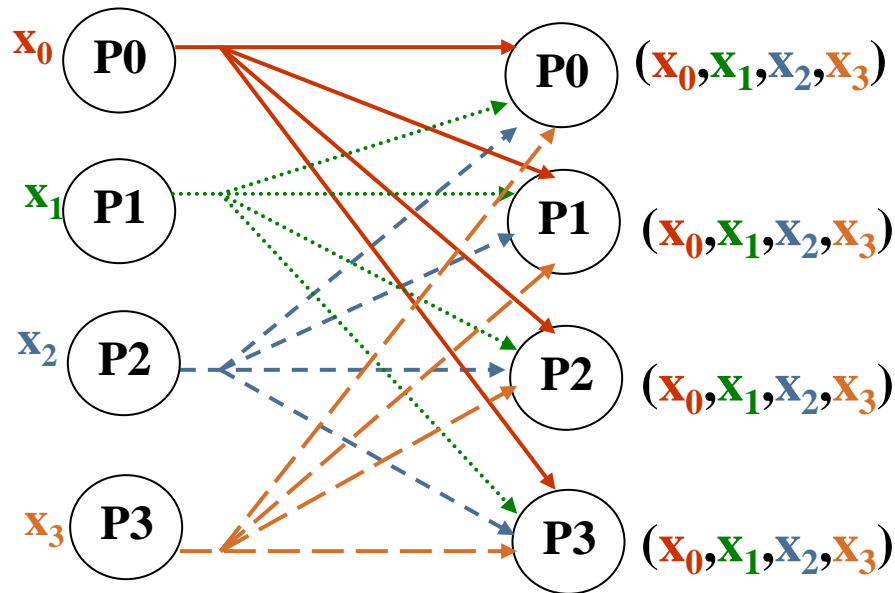


Permutación baraje-2:

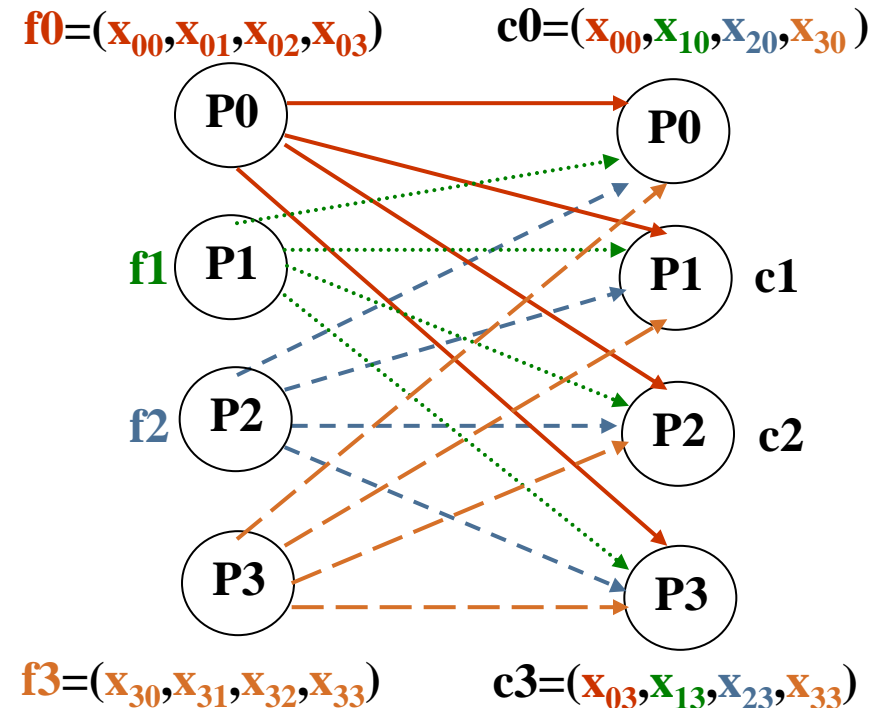


Comunicación todos-a-todos

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



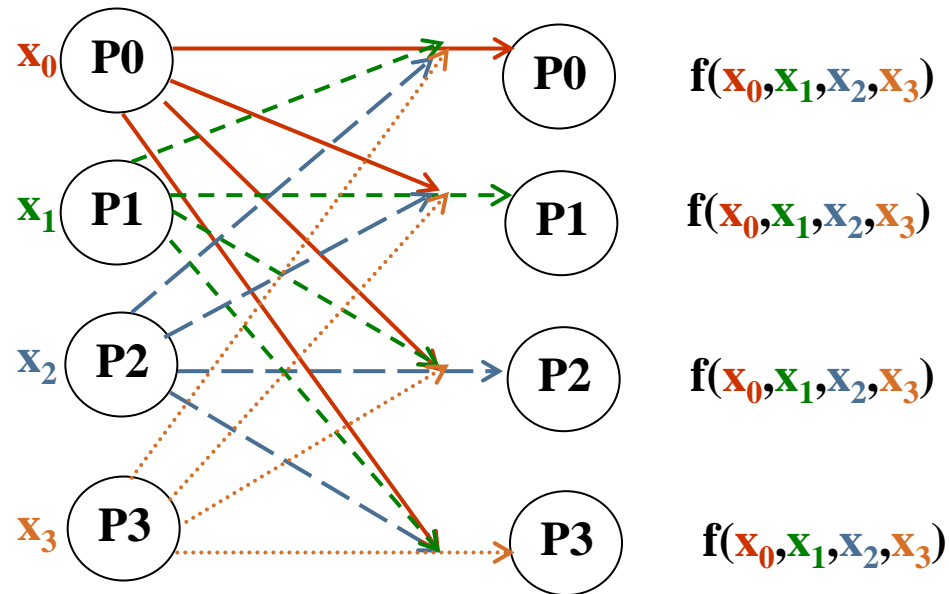
Todos Dispersan (*all-scatter*)



c= column matrix
f= fila matrix

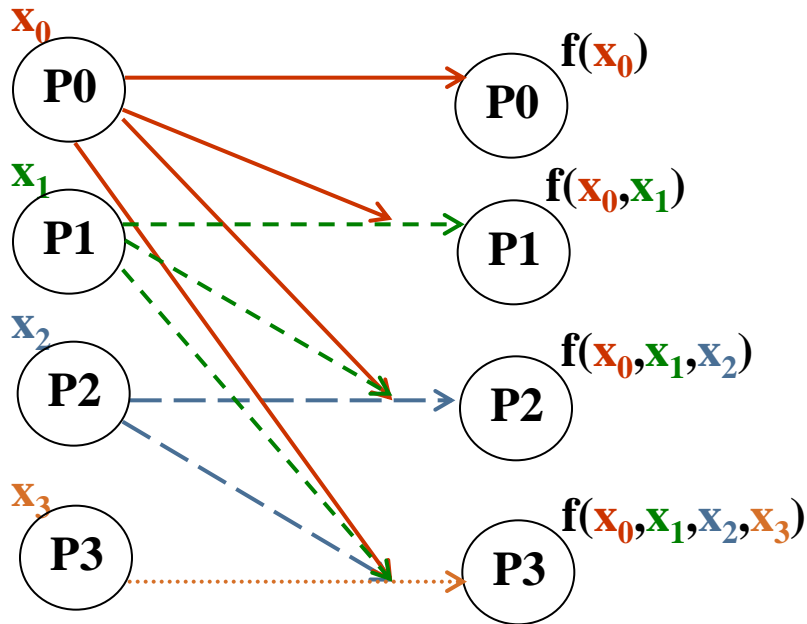
Servicios compuestos

Todos combinan

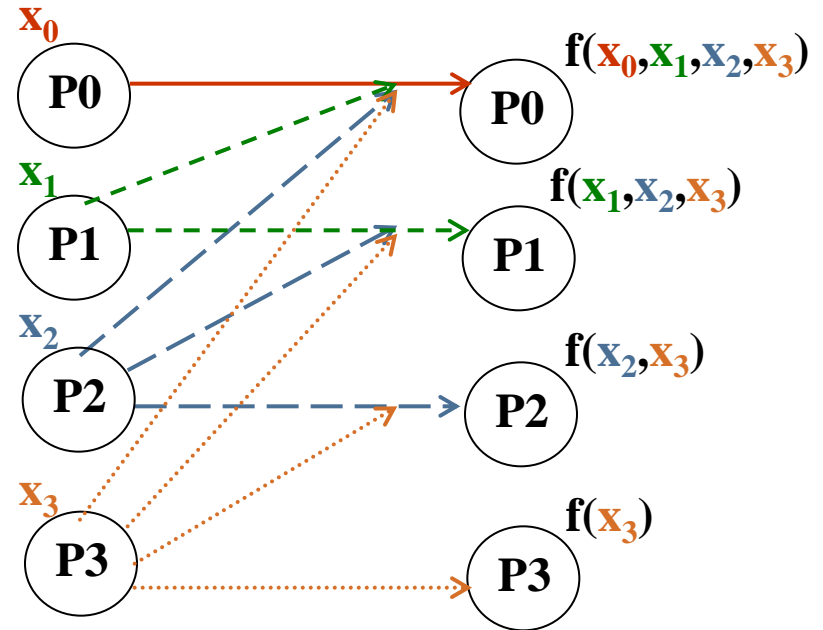


Servicios compuestos

Recorrido (scan) prefijo paralelo



Recorrido sufijo paralelo



Ejemplo: comunicación colectiva en OpenMP

Uno-a-todos	Difusión (Seminario pract. 2)	<ul style="list-style-type: none">✓ Cláusula <code>firstprivate</code> (desde thread 0)✓ Directiva <code>single</code> con cláusula <code>copyprivate</code>✓ Directiva <code>threadprivate</code> y uso de cláusula <code>copyin</code> en directiva <code>parallel</code> (desde thread 0)
Todos-a-uno	Reducción (Seminario pract. 2)	<ul style="list-style-type: none">✓ Cláusula <code>reduction</code>
Servicios compuestos	Barreras (Seminario pract. 1)	<ul style="list-style-type: none">✓ Directiva <code>barrier</code>

Ejemplo: comunicación en MPI

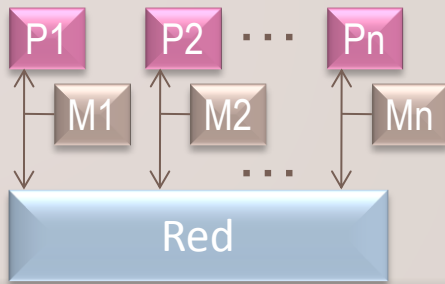
Uno-a-uno	Asíncrona	<code>MPI_Send()</code> / <code>MPI_Receive()</code>
Uno-a-todos	Difusión	<code>MPI_Bcast()</code>
	Dispersión	<code>MPI_Scatter()</code>
Todos-a-uno	Reducción	<code>MPI_Reduce()</code>
	Acumulación	<code>MPI_Gather()</code>
Todos-a-todos	Todos difunden	<code>MPI_Allgather()</code>
Servicios compuestos	Todos combinan	<code>MPI_Allreduce()</code>
	Barreras	<code>MPI_Barrier()</code>
	Scan	<code>MPI_Scan</code>

Detalles de la programación con MPI en la asignatura: **Arquitecturas y Computación de Altas Prestaciones**
(IC.SCAP.ACAP – Especialidad (IC), Materia (SCAP), Asignatura (ACAP))

Contenido Lección 4

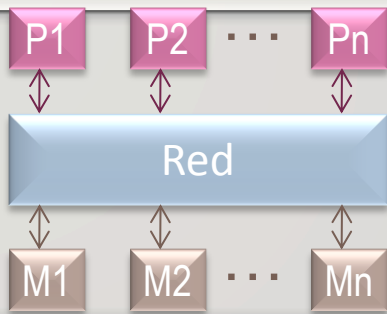
- Problemas que plantea la programación paralela al programador. Punto de partida
- Herramientas para obtener código paralelo
- Estilos/paradigmas de programación paralela
- Estructuras típicas de códigos paralelos

Estilos de programación y arquitecturas paralelas



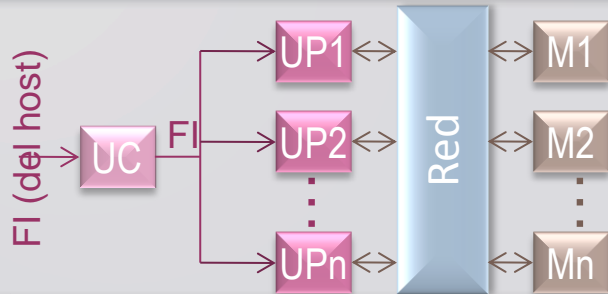
Paso de mensajes

- Multicomputadores



Variables compartidas

- Multiprocesadores



Paralelismo de datos

- Procesadores matriciales, GPU (*stream processing*)

Estilos de programación y herramientas de programación

- Paso de mensajes (*message passing*)
 - Lenguajes de programación: Ada, Occam
 - API (Bibliotecas de funciones): MPI, PVM
- Variables compartidas (*shared memory, shared variables*)
 - Lenguajes de programación: Ada, Java
 - API (directivas del compilador + funciones): **OpenMP**
 - API (Bibliotecas de funciones): POSIX Threads, shmem, Intel TBB (*Threading Building Blocks*)
- Paralelismo de datos (*data parallelism*)
 - Lenguajes de programación: HPF (*High Performance Fortran*), Fortran 95 (`forall`, operaciones con matrices/vectores)
 - API (funciones - *stream processing*): Nvidia CUDA

Contenido Lección 4

- Problemas que plantea la programación paralela al programador. Punto de partida
- Herramientas para obtener código paralelo
- Estilos/paradigmas de programación paralela
- Estructuras típicas de códigos paralelos

Estructuras típicas de procesos/threads/tareas

Estructuras típicas de procesos/threads/tareas en código paralelo

Descomposición de dominio o descomposición de datos

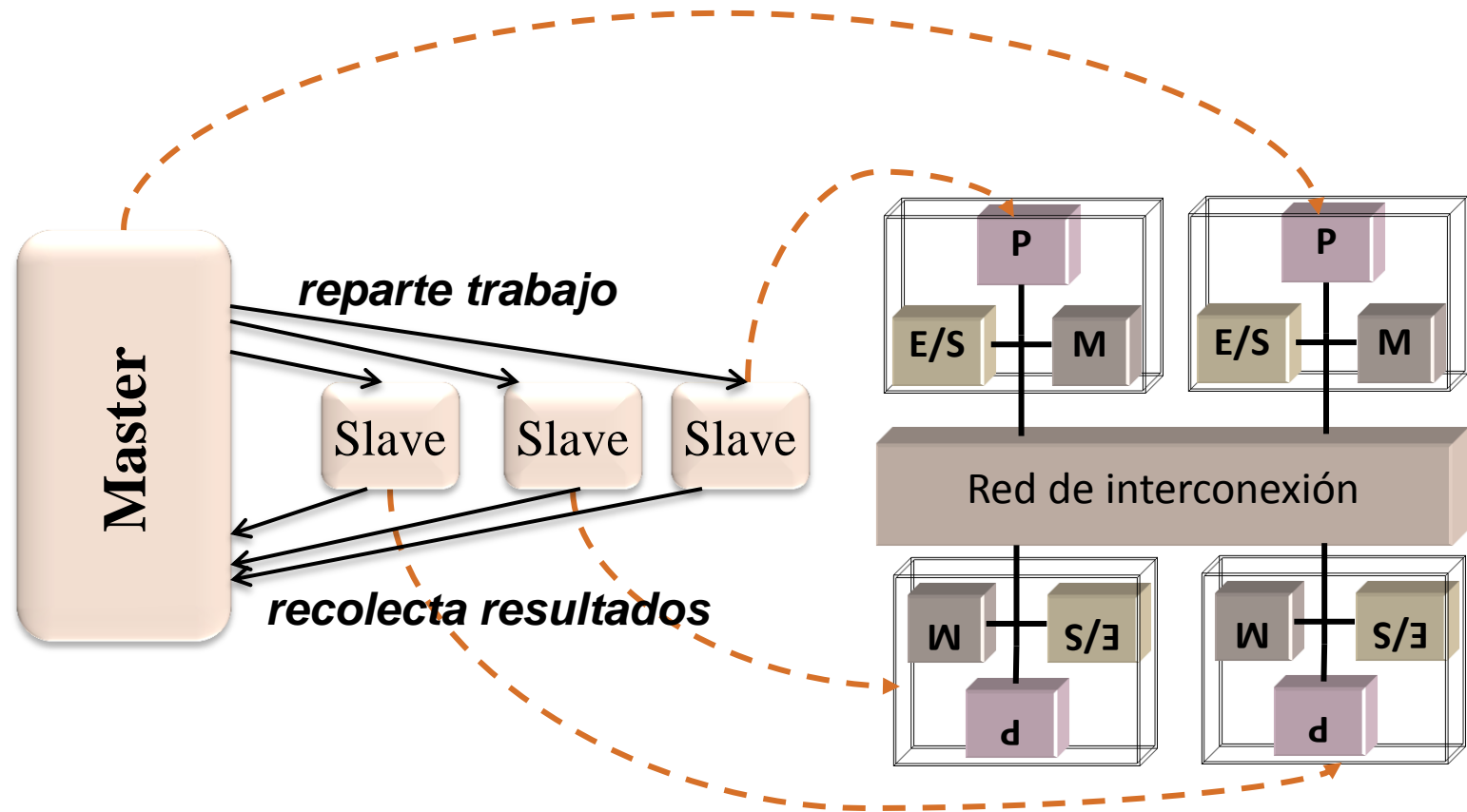
cliente/servidor

Divide y vencerás o descomposición recursiva

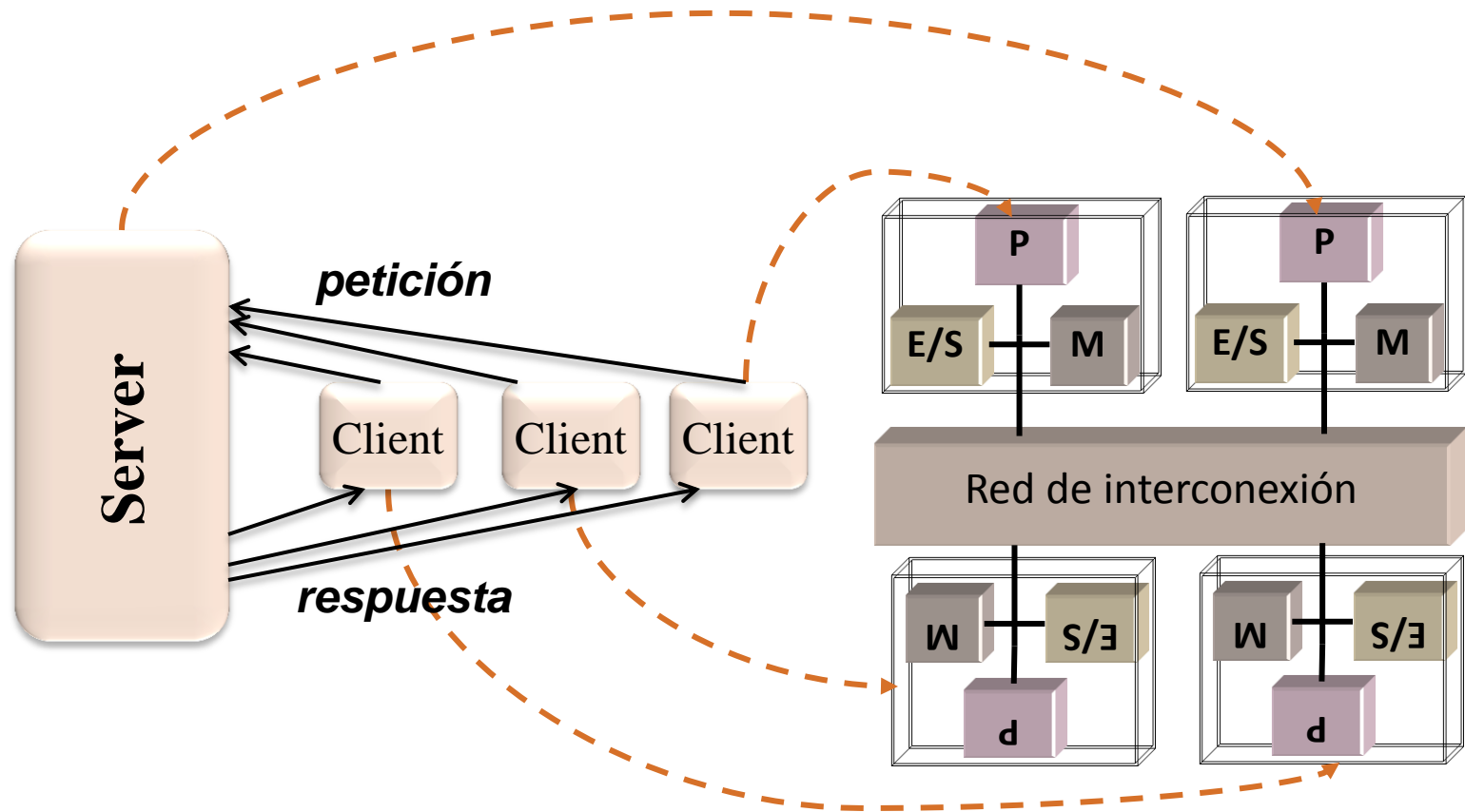
Segmentación o flujo de datos

Master-Slave, o granja de tareas

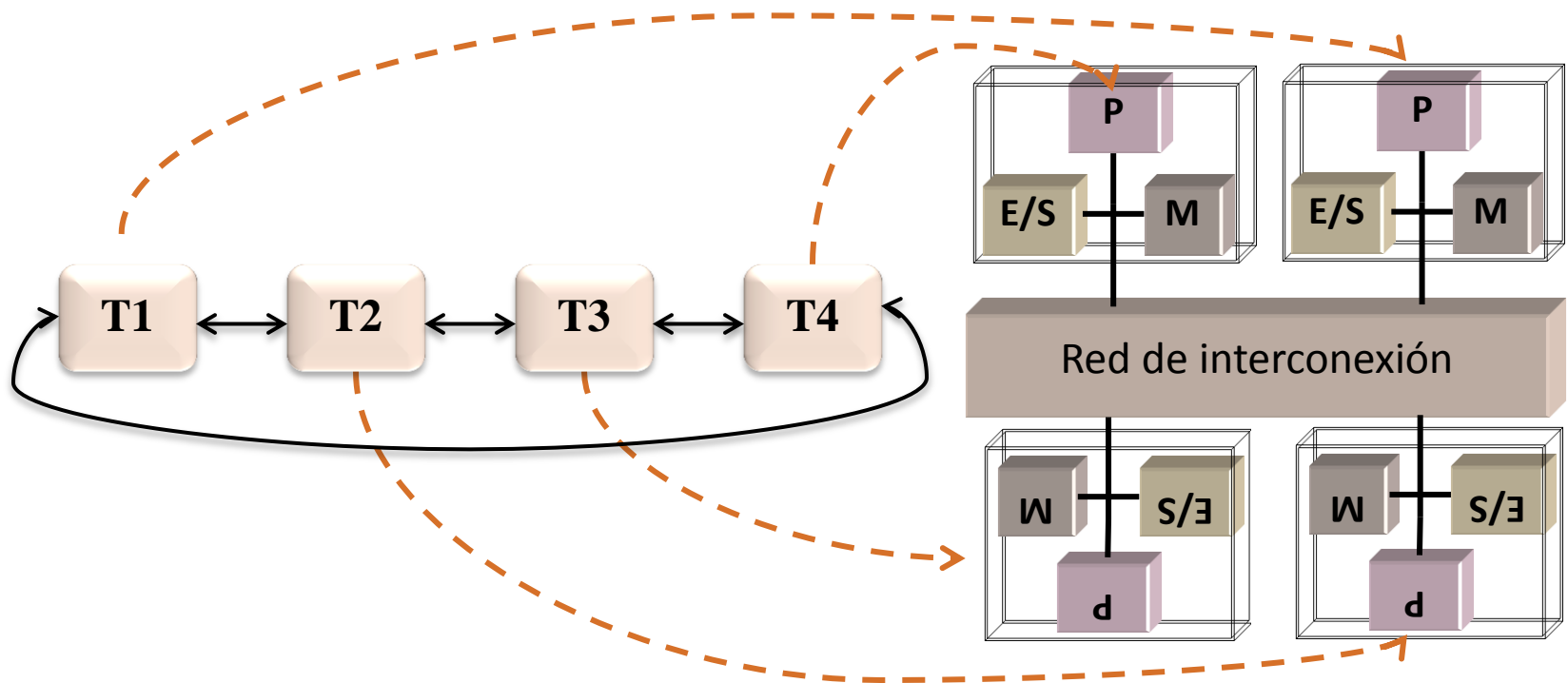
Master-Slave o granja de tareas



Cliente-Servidor

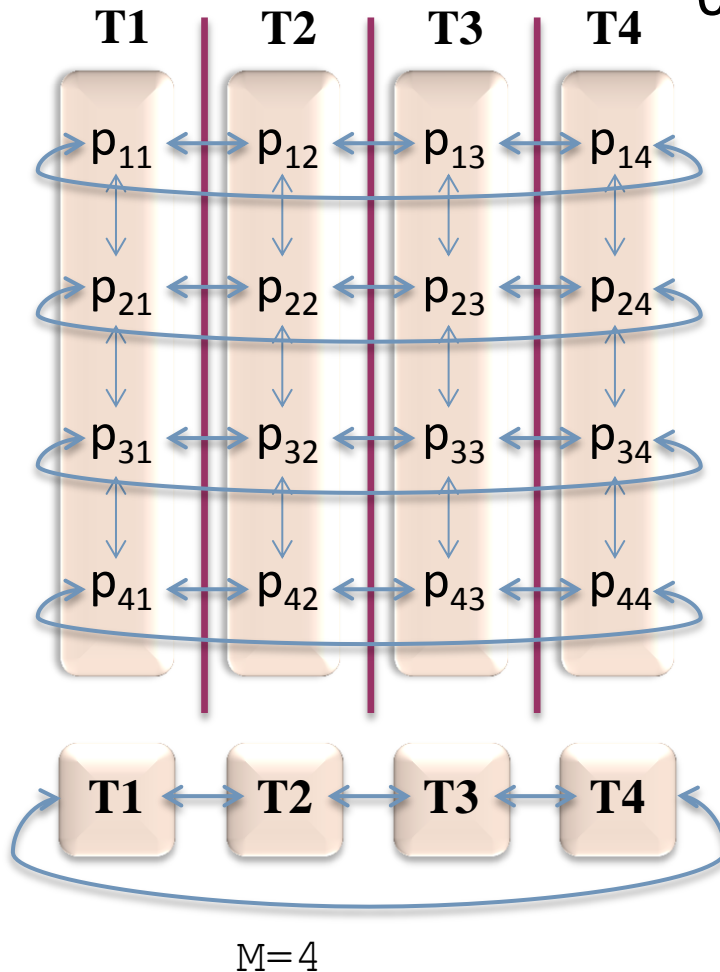


Descomposición de dominio o descomposición de datos I



Descomposición de dominio II. Ej: filtrado imagen

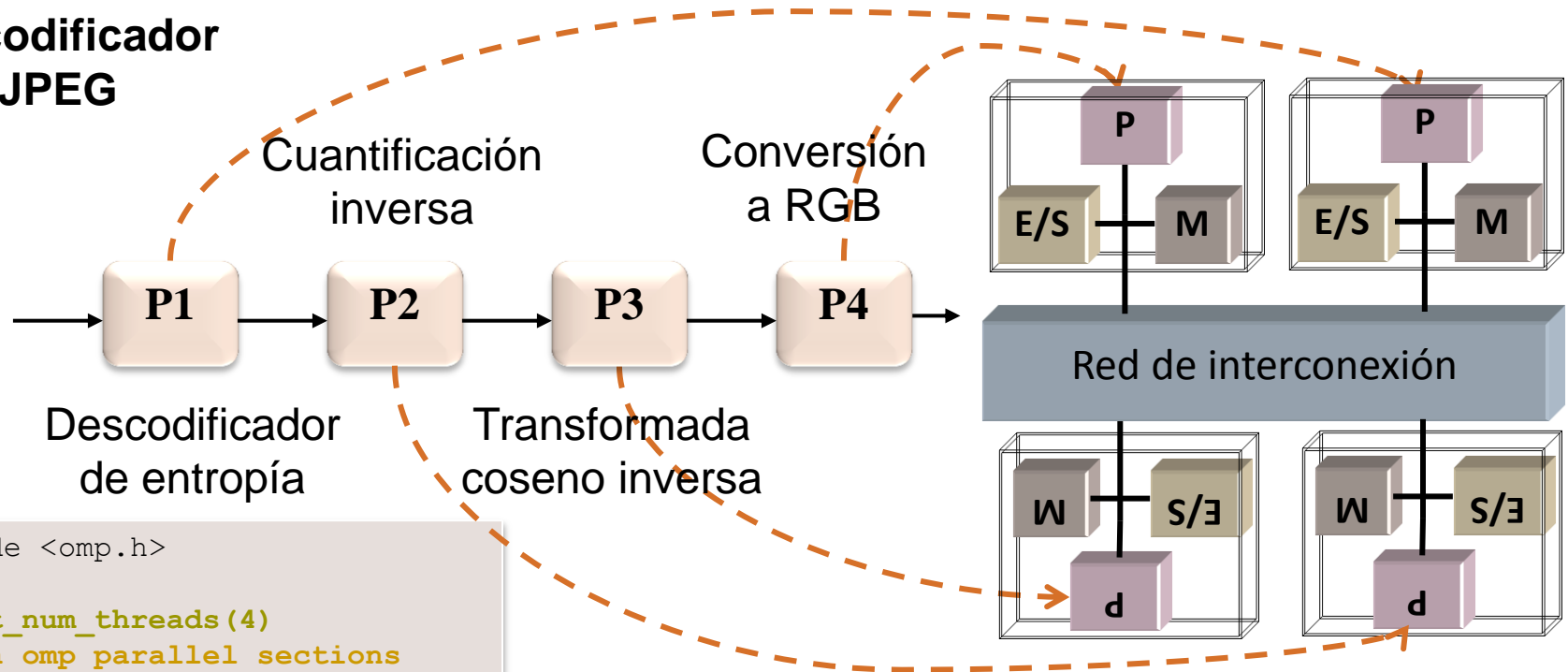
$$p_{ij}(t) = 0,75 \times p_{ij}(t-1) + 0,0625 \times p_{i-1j}(t-1) + 0,0625 \times p_{ij-1}(t-1) + 0,0625 \times p_{i+1j}(t-1) + 0,0625 \times p_{ij+1}(t-1)$$



```
#include <omp.h>
...
omp_set_num_threads(M)
#pragma omp parallel private(i)
{
    for (i=0; i<N; i++) //recorre filas
        #pragma omp for
        for (j=0; j<M; j++) //recorre columnas
            pS[i,j] = 0,75*p[i,j] +
                0,0625*(p[i-1,j]+p[i,j-1]+
                    p[i+1,j]+ p[i,j+1]);
}
...
```

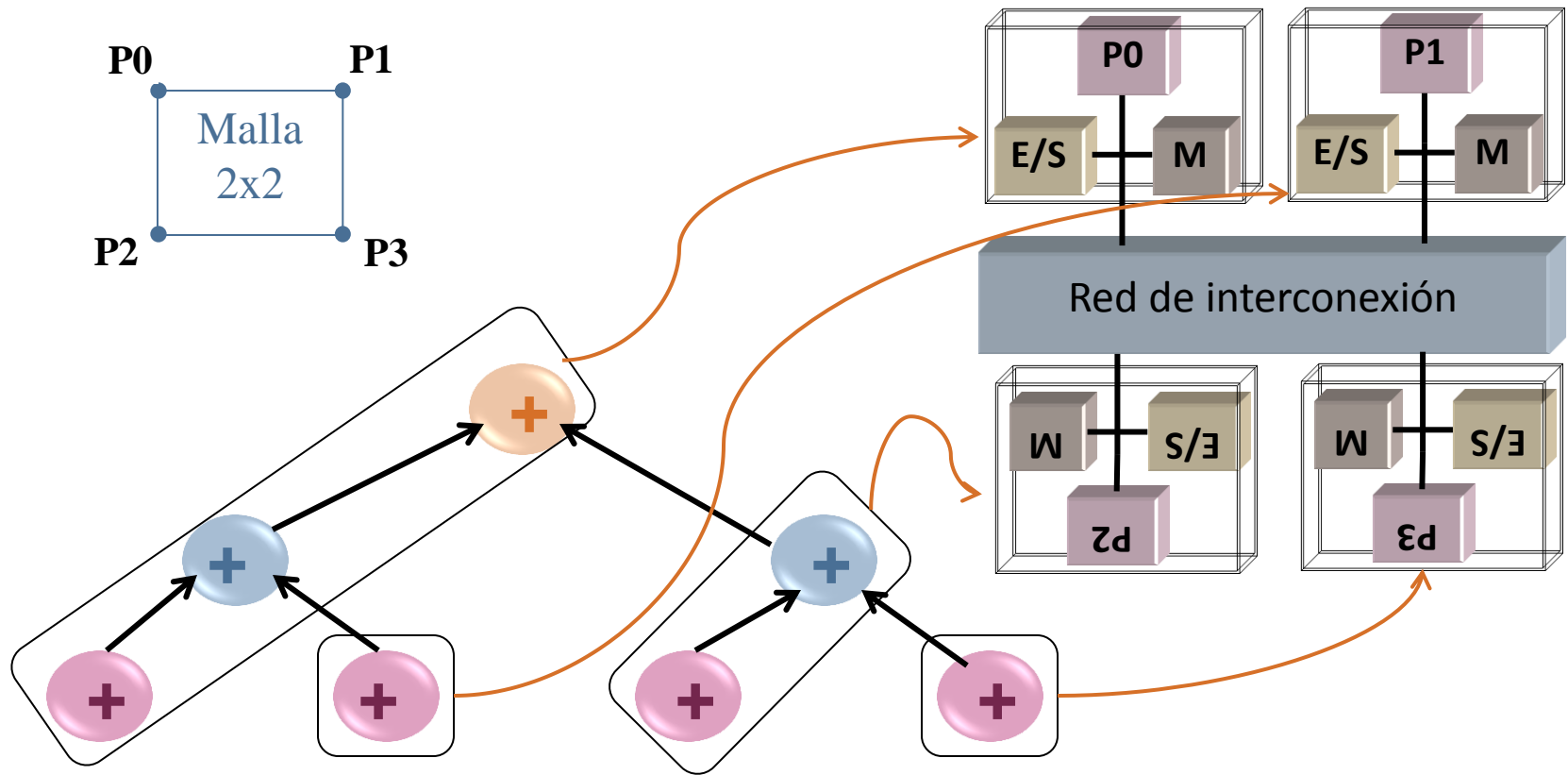

Estructura segmentada o de flujo de datos

Descodificador JPEG



```
#include <omp.h>
...
omp_set_num_threads(4)
#pragma omp parallel sections
{
    #pragma omp section
    P1();
    #pragma omp section
    P2();
    #pragma omp section
    P3();
    #pragma omp section
    P4();
}
```

Divide y vencerás o descomposición recursiva



$$(((a_0 + a_1) + (a_2 + a_3)) + ((a_4 + a_5) + (a_6 + a_7)))$$