

# Apuntes-Tema-2-Teoria-AC.pdf



Togoto2



Arquitectura de Computadores



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación  
Universidad de Granada



Estamos de  
**Aniversario**

De la universidad al  
mercado laboral:  
especialízate con los posgrados  
de EOI y marca la diferencia.



**EOI** Escuela de  
organización  
industrial



[saber más](#)

# Si estás en tu **spending era...**

mejor tener una app que te diga en qué tiendas se ha quedado registrada tu tarjeta.

¡Como la app de ING!

Saber más

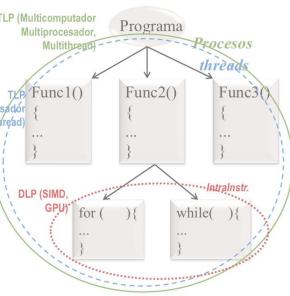


## TEMA 2. → PROGRAMACIÓN PARALELA

Los problemas que generan la programación paralela y NO la programación secuencial son los siguientes:

### 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/a flujos (procesos y/o threads).
  - Asignación de flujos a procesadores/núcleos.
  - Sincronización y comunicación entre flujos.
- Los debe abordar la herramienta de programación o el programador o SO



6 | Tema 2. Programación Paralela

1. División en unidades de cómputo independientes (tareas).

2. Agrupación / asignación de tareas o carga de trabajo (datos) en/a flujos (procesos y/o thread).

3. Asignación de flujos a procesadores / núcleos.

4. Sincronización y comunicación entre flujos

Los debe abordar la herramienta de programación o el programador o SO.

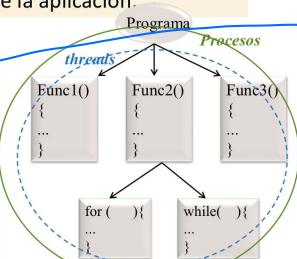
La situación en la que nos encontramos es la siguiente:

### Punto de partida

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

**Apoyo:**

- Programa paralelo que resuelve un problema parecido.
- Versiones paralelas u optimizadas de bibliotecas de funciones:  
BLAS (Basic Linear Algebra Subroutine), LAPACK (Linear Algebra PACKAGE), ...



7 | Tema 2. Programación Paralela

- Partir de una versión secuencial, esto resulta útil

- Descripción o definición de la aplicación.

→ A veces, ya están programas paralelizados que resuelven

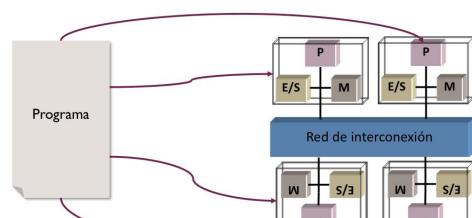
un problema parecido. Entonces hay versiones paralelas u

optimizadas de bibliotecas de funciones: - LAPACK. ↗ ejecución paralela.

Modos de programación, MIMD: varios flujos de instrucciones y varios flujos de datos: Multicomputadores o multiprocesadores

### Modos de programación MIMD

• SPMD (Single-Program Multiple Data)



SPMD → Hay un programa que se ejecuta en cada uno de

los procesadores / núcleos, el programa es el mismo, el flujo de

instrucciones puede ser totalmente diferente. Como los datos son

distintos, el flujo de instrucciones también es distinto.

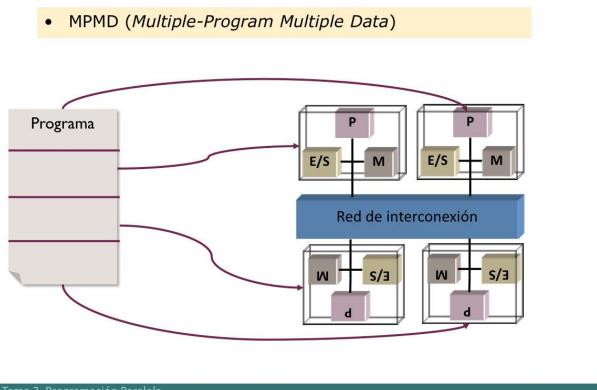
WUOLAH



Modos de programación, MIMD: varios flujos de instrucciones y varios flujos de datos: Multicomputadores o multiprocesadores

## Modos de programación MIMD

MIMD → El programa se divide en partes, que cada parte lo que



hacen es ejecutarse cada una, en distintos nodos / procesador / núcleo.

Los datos que usan los programas son diferentes y por consiguiente pienso que las instrucciones son diferentes.

9 | Tema 2. Programación Paralela

## HERRAMIENTAS DE PROGRAMACION PARALELA

### Herramientas de programación paralela

Compiladores paralelos (parallelización automática): Son capaces de generar un código de alto nivel → programa paralelo → se encarga

de la extracción automática del paralelismo implícito en la aplicación.

Lenguajes paralelos: Tienen una serie de construcciones que permiten el paralelismo

API funciones: Permiten realizar tipos de tareas como

comunicación entre procesos o hilos. Lenguajes para arquitecturas paralelas: Construcciones lenguaje + funciones

Nivel de abstracción

Compiladores paralelos (parallelización automática)

Extracción automática del paralelismo

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

Construcciones del lenguaje + funciones (Lenguaje secuencial + directivas) + funciones

API funciones (Pthreads, MPI)

Lenguaje secuencial + funciones

Lenguajes para arquitecturas paralelas de propósito específico (CUDA)

Construcciones del lenguaje + funciones

↳ Si somos conscientes

Las herramientas permiten de forma implícita (quiere decir que

↳ Si somos conscientes

lo hace el SO) o explícita (lo hace el programador):

Localizar paralelismo o descomponer en tareas independientes

Asignar las tareas, la carga de trabajo a procesos / hilos

Crear y terminar procesos/hilos

Comunicar y sincronizar procesos / hilos

### HERRAMIENTAS PARA OBTENER PROGRAMAS PARALELOS

#### Herramientas para obtener programas paralelos

- Las herramientas permiten de forma implícita (lo hace la herramienta) o explícita (lo hace el programador):
  - Localizar paralelismo o descomponer en tareas (trabajo independientes (*decomposition*))
  - Asignar las tareas, es decir, la carga de trabajo (código + datos), a procesos/threads (*scheduling*), o agrupar
  - Crear y terminar procesos/threads (o enrollar y desenrollar 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*)

12 | Tema 2. Programación Paralela

## EJEMPLO → Cálculo de PI en MPI/C

### 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); → Crear y Terminar
#pragma omp parallel → Comunicar y sincronizar
{ #pragma omp for reduction(+:sum) private(x) \
            schedule(dynamic) → Asignar
    for (i=0;i< intervalos; i++) {
        x = (i+0.5)*ancho; sum = sum + 4.0/(1.0+x*x);
    }
    sum*= ancho;
}
```

Localizar y Asignar

En la imagen se observa cuantas hebras se van a usar

luego esa cantidad de hebras se introducen en la variable

omp\_set\_num\_threads (NUM\_THREADS), esto es forma secuencial.

Hasta que llegamos a pragma parallel , que es todo de forma paralela

## EJEMPLO → Cálculo de PI en MPI/C

### Ejemplo: cálculo de PI en MPI/C

```
#include <mpi.h>
main(int argc, char **argv) {
    double ancho,x,lsum,sum; int intervalos,i,nproc,iproc;
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) exit(1);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc); → Enrolar
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc); → Localizar-Agrupar
    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; → Comunicar/sincronizar
    MPI_Reduce(&lsum, &sum, 1, MPI_DOUBLE,
               MPI_SUM,0,MPI_COMM_WORLD);
    MPI_Finalize(); → Desenrolar
}
```

En las dos imágenes anteriores, se puede observar como

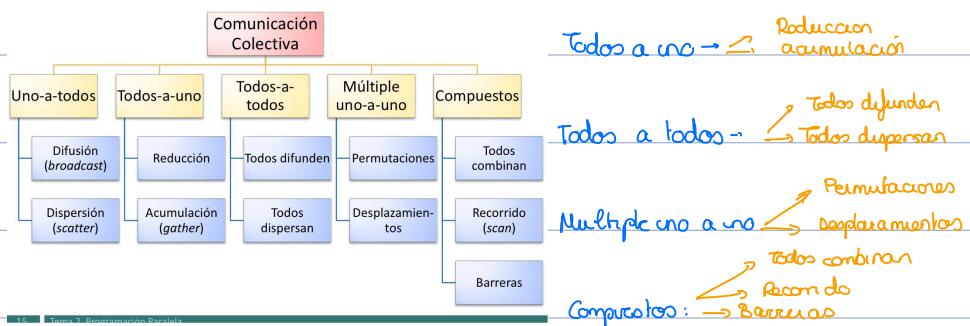
se cumplen las operaciones dentro las anteriores como

enrolar, localizar - agrupar comunicar / sincronizar,

desenrolar, crear y terminar, localizar y asignar.

## COMUNICACIONES COLECTIVAS

### Comunicaciones colectivas



Uno a todos → Difusión  
→ Dispersión

Todos a uno → Reducción  
acumulación

Todos a todos → Todos difunden  
Todos dispersan

Múltiple uno a uno → Permutaciones  
desplazamientos

Compuestos: → Todos combinan  
→ Recorrido  
→ Barreras

A continuación, voy a explicar cada uno de ellos, en las siguientes páginas.

# Si estás en tu **spending era...**

mejor tener una app que te diga en qué tiendas se ha quedado registrada tu tarjeta.

¡Como la app de ING!

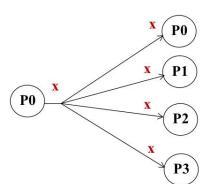
Saber más



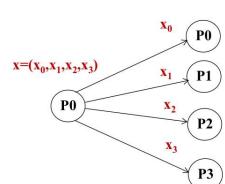
## COMUNICACIÓN UNO - A - TODOS

### Comunicación uno-a-todos

Difusión (broadcast)



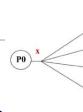
Dispersión (scatter)



DIFUSIÓN (BROADCAST): El proceso 0 envía el mismo

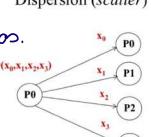
Difusión (broadcast)

dato a todos los demás, P0, P1, P2, P3.



DISPERSIÓN (SCATTER): Un proceso envía un

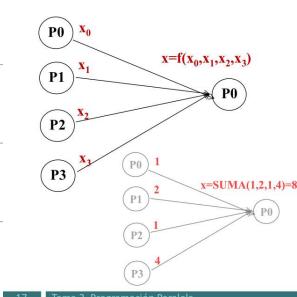
tipo de dato distinto a todos los demás procesos.



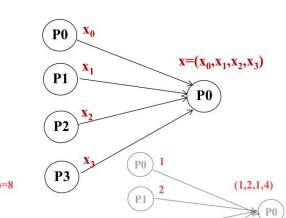
## COMUNICACIÓN TODOS - A - UNO

### Comunicación todos-a-uno

Reducción

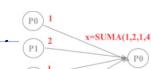


Acumulación (gather)



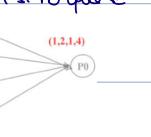
REDUCCIÓN: Envían todos los procesos con distintos tipo

de dato a un mismo proceso, de manera que se combinen  
con alguna función (sumar, restar, etc...).



ACUMULACIÓN (GATHER): Es lo mismo que

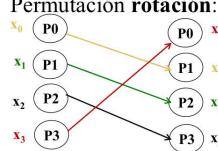
la reducción pero no se combinan con una función si no que se  
realiza una copia de todos datos.



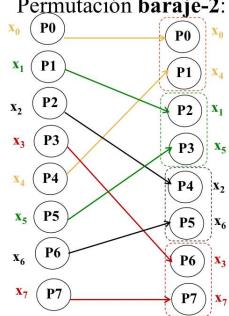
## COMUNICACIÓN MÚLTIPLE UNO - A - UNO

### Comunicación múltiple uno-a-uno

Permutación rotación:



Permutación baraje-2:



PERMUTACIÓN ROTACIÓN: El proceso por ejemplo P0, manda  
el dato al siguiente índice de ese mismo proceso. Es como  
si hubiera un ciclo, por ello, el nombre de rotación.

PERMUTACIÓN BARAJE-2: Los 8 procesos se dividen en

2 grupos y el primero de cada grupo hace P0-P0,

P1-P2, P2-P4, P3-P6 y con eso así y se

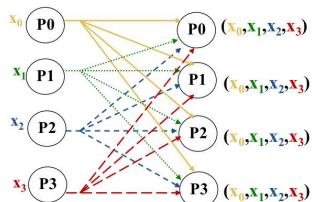
cumple lo que viene siendo el nombre baraje-2.

WUOLAH

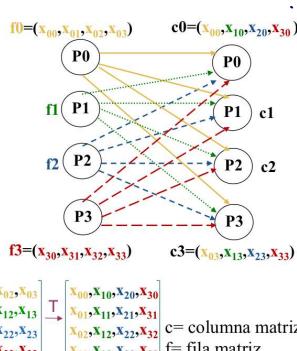
## COMUNICACION TODOS A TODOS

### Comunicación todos-a-todos

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



Todos Dispersan (all-scatter)



Todos difunden o all - broadcast : Cada proceso

realiza un broadcast. El P0 envia su dato a todos

los procesos al igual que P1, P2 y P3. Va a tener

los mismos datos tanto P0, P1, P2, P3.

Todos dispersan (all - scatter) : Es lo mismo pero

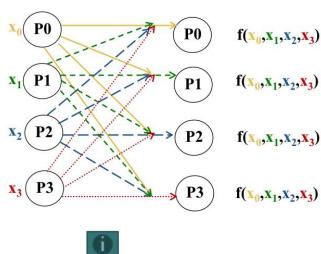
$$f0 = (x_{00}, x_{01}, x_{02}, x_{03})$$

donde viene hacia que proceso va

## SERVICIOS COMPLEJOS

### Servicios compuestos

Todos combinan o todos reducen



Todos combinan: Se hace una reducción en todos

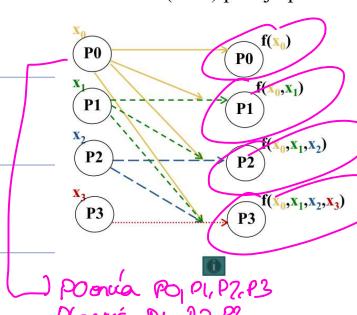
los procesos, y cada proceso va a obtener un mismo resultado

resultado. Tambien se le conoce como todos reducen.

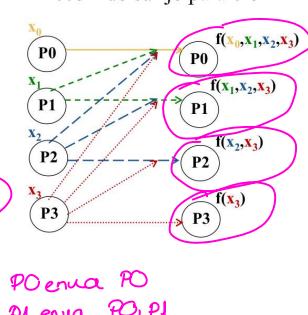
## SERVICIOS COMPLEJOS

### Servicios compuestos

Recorrido (scan) prefijo paralelo



Recorrido sufijo paralelo



Recorrido (scan) o prefijo paralelo: Cada proceso manda

a aquellos procesos que tienen un indice mayor o igual.

Recorrido sufijo paralelo: Cada proceso manda los datos a aquellos procesos que tienen un indice menor o igual que el mismo proceso.



**ÚNETE A McDONALD'S  
Y ENCUENTRA A TU GENTE**

**¿TE VIENES?**



**M**  
**My CREW**  
**Mi trabajo. Mi pasión. Mi gente.**

## COMUNICACION COLECTIVA EN OPEN MP

### Ejemplo: comunicación colectiva en OpenMP

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

Uno a todos → difusión

Todos a uno → Reducción

Servicios compuestos → Barreras

### 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 acumulan	<code>MPI_Allgather()</code>
	Todos combinan	<code>MPI_Allreduce()</code>
Servicios compuestos	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))

Uno a todos → difusión, dispersión

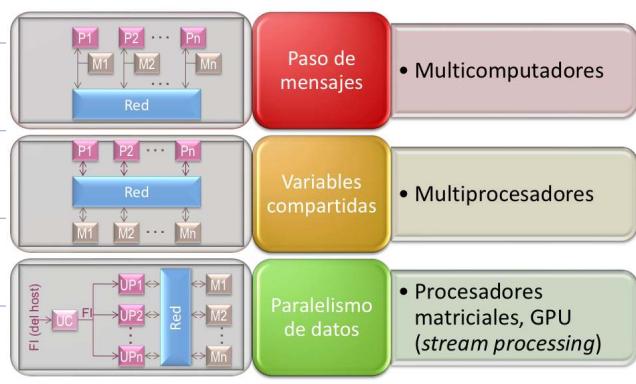
Uno a uno → asíncrona

Todos a uno → Reducción, acumulación

Todos a todos → Todos acumulan

Servicios compuestos → Todos combinan, barreras, scan.

### ESTILOS/ MODELOS DE PROGRAMACIÓN Y ARQUITECTURAS PARALELAS



MIMD → Pasos de mensajes y variables compartidas.

MULTICOMPUTADORES → Pasos de mensajes

MULIPROCESADORES → Variables compartidas

Memoria físicamente distribuida

Aprovechamos el paralelismo de datos con los

procesadores matriciales, GPU, (streaming processing).

Ya has abierto los apuntes,  
te mereces ese descanso.

También te mereces que no te cobren  
por tener una cuenta. **Cositas.**

Ven a la  
Cuenta NoCuenta



Saber más



zzz



## ESTILOS/MODELOS DE PROGRAMACIÓN Y ARQUITECTURAS PARALELAS

No muy importante  
que se pueda decir!

### Estilos de programación y herramientas de programación

- Paso de mensajes (*message passing*)
  - Lenguajes de programación: Ada, Occam
  - API (Bibliotecas de funciones): MPI ("de facto" industry standard for message passing), 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), C++ con clases thread, mutex, atomic ...
- Paralelismo de datos (*data parallelism*)
  - Lenguajes de programación + funciones: HPF (High Performance Fortran), Fortran 95 (`forall`, operaciones con matrices/vectores)
  - API (directivas del compilador + funciones - stream processing): OpenACC, OpenMP 4.0

26 | Tema 2. Programación Paralela

Maestro encargado ←  
pero en inglés → master-slave MASTER-SLAVE o granja de tareas

Maestro se encarga de distribuir el trabajo entre los

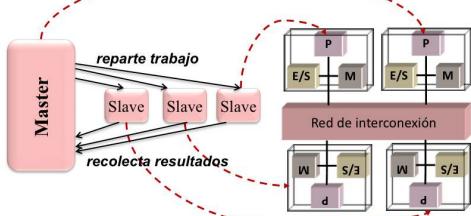
demás procesos, que son los trabajadores

Maestro envía info a trabajo a los procesos y eso

es una comunicación de UNO a TODOS.

Trabajadores recogen los resultados y se los envían

al maestro y esa es comunicación Todos a Uno.



30 | Tema 2. Programación Paralela

### CUENTE - SERVIDOR

Un proceso o hebra que actua como servidor, y luego

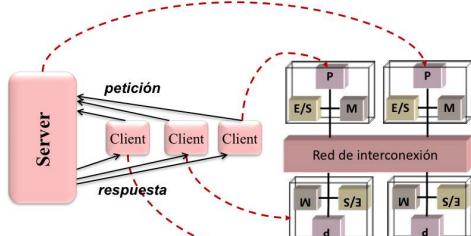
hay muchos mas procesos que son los clientes.

Clientes solicitan al servidor que haga operaciones,

ya que el servidor tiene mas capacidad de realizar

las operaciones mas eficientes que los propios clientes.

Clientes pide petición al servidor, el servidor



30 | Tema 2. Programación Paralela

ejecuta y lo que hace es enviarle la respuesta a los clientes.

**WUOLAH**

## DESCOMPOSICIÓN DE DOMINIO O DECOMPOSICIÓN DE DATOS

### Descomposición de dominio o decomposición de datos I

Varias hebras diferentes que hacen datos

diferentes, y se hace la distribución de datos.

Cada código y datos de las hebras se aplican a distintos nodos, según las comunicaciones se pasan de una forma u otra.

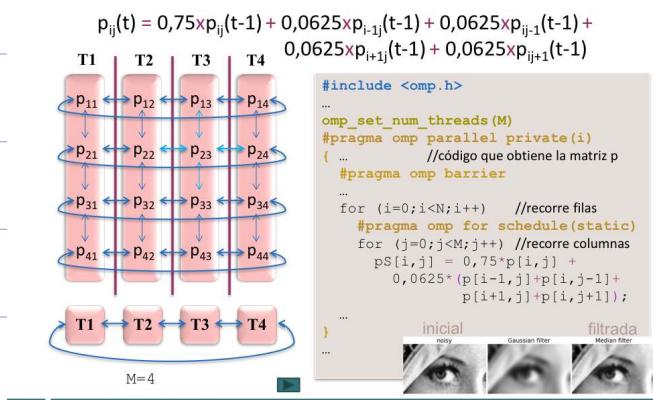
Figura 3. Descomposición de dominio

### EJEMPLO IMAGEN DECOMPOSICIÓN DE DOMINIO

### Descomposición de dominio II. Ej: filtrado de imagen

→ Esto es un ejemplo de la descomposición de dominio II.

↳ Ejemplo filtrado imagen!

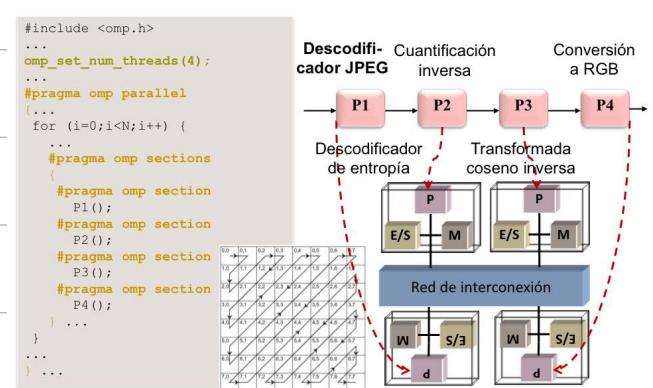


### Estructura segmentada o de flujo de datos

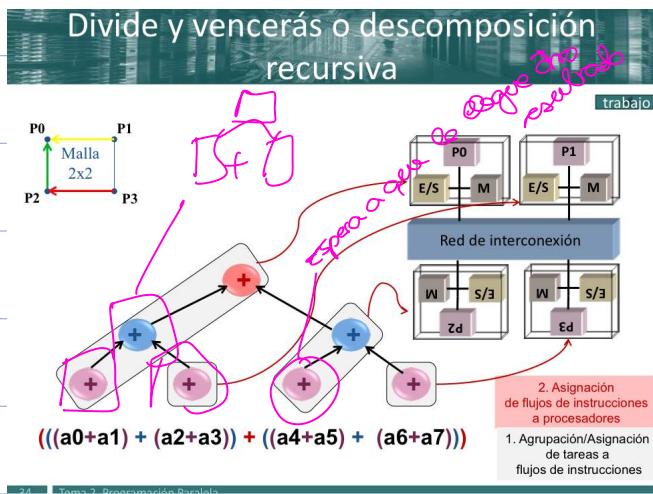
Numeros de procesos que ejecutan una tarea

distinta y se asignan a nodos distintos. Tienen sus conjuntos de datos propios de cada nodo

Ejemplo: 4 hebras → podemos hacer que cada una de las hebras hagan distinto trabajo.



## DIVIDE Y VENCERÁS O DESCOMPOSICIÓN RECURSIVA



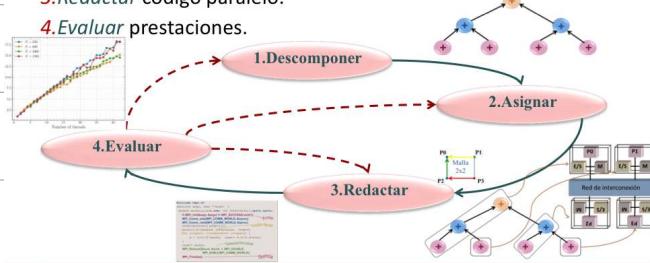
El problema grande se divide el problema

mas chiquitos, dan la solución se combinan  
todas las soluciones para dar lugar a la  
nueva solución → Divide y concurras

## [Lecón 2]

### Proceso de paralelización

1. Descomponer (decomposition) en tareas independientes o Localizar paralelismo
2. Asignar (planificar+mapear, schedule+map) tareas a procesos y/o threads.
3. Redactar código paralelo.
4. Evaluar prestaciones.



1. Descomponer: ¿d que tareas se pueden calcular

en forma independiente? → Paralelismo completo

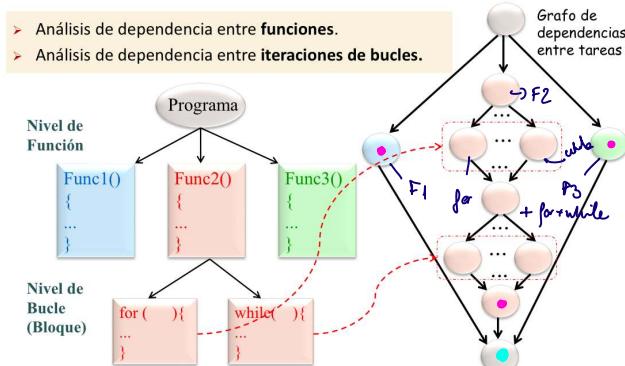
2. Asignar: Tenemos que asignar si las tareas

independientes las tenemos que agrupar → Mapado

3. Redactar: Que tareas independientes se realizan en los procesos. se redacta el código. 4. Evaluar prestaciones.

### DESCOMPOSICIÓN EN TAREAS INDEPENDIENTES

### Descomposición en tareas independientes



Para descomponer en tareas independientes:

- Hay que tener en cuenta las dependencias que hay entre las funciones.

- Luego las dependencias entre iteraciones de bucle.

Hasta que no terminan las tareas del punto rosa, no convierta la del punto azul.

# Si estás en tu **spending era...**

mejor tener una app que te diga en qué tiendas se ha quedado registrada tu tarjeta.

¡Como la app de ING!

Saber más



## ASIGNACIÓN DE TAREAS A PROCESOS/THREADS

### Asignación de tareas a procesos/threads I

- Incluimos: agrupación de tareas en procesos/threads (*scheduling*) y mapeo a procesadores/cores (*mapping*)
- La **granularidad** de la carga de trabajo (tareas) asignada a los procesos/threads depende de:
  - número de cores o procesadores o elementos de procesamiento
  - tiempo de comunicación/sincronización frente a tiempo de cálculo
- **Equilibrado de la carga** (tareas = código + datos) o *load balancing*:
  - Objetivo: unos procesos/threads no deben esperar a otros

46 Tomo 2 - Programación Paralela

La granularidad de la carga de trabajo (tareas)

asignadas a los procesos depende de:

nº de cores o procesadores o elementos de procesamiento

tiempo de comunicación / sincronización frente a tiempo de cálculo.

Equilibrio de carga → objetivo: unos procesos no deben hacer espera a otros.

¿De qué depende el equilibrado de carga?

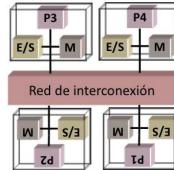
- La arquitectura y la aplicación / descomposición

Los homogéneos frente a no uniformes: es que dice que dependiendo del modulo de computo que se le asigne va a tardar más o menos.

uniforme frente a no uniforme: Lo de NUMA y UMA → Tema anterior.

Explícito → lo hace el programador

Implícito → lo hace la máquina, el so



### Tipos de asignación:

- Estática
  - Está determinado qué tarea va a realizar cada núcleo/procesador
  - Explícita: programador
  - Implícita: herramienta de programación al generar el código ejecutable
- Dinámica (en tiempo de ejecución)
  - Distintas ejecuciones pueden asignar distintas tareas a un núcleo/procesador
  - Explícita: el programador
  - Implícita: herramienta de programación al generar el código ejecutable

### Tipos de asignación:

- Estática → En un código indica que trabajo va a realizar cada hilo o cada proceso

La distribución de carga no cambia en tiempo de ejecución.

- Explícita: Lo hace el programador

- IMPUCITA: Herramienta de programación al generar código ejecutable.

Ventajas: Baja sobrecarga, más predecible, buena eficiencia en cargas equilibradas

Inconvenientes: Falta de adaptabilidad - infligente en cargas desequilibradas.

WUOLAH

• Dinámica → Las tareas se asignan en tiempo de ejecución, lo que permite una mejor adaptación a las variaciones en la carga de trabajo.

- Explícita: El programador define un esquema de balanceo de carga.

- Implicita: Haciéndolo de programación al generar el código ejecutable.

Ventajas: Mejor adaptabilidad, eficiencia en cargas heterogéneas (si algunas requieren más tiempo, otros núcleos pueden asumir trabajo adicional) uso óptimo de los recursos.

Inconvenientes: Mayor sobre carga, menos predecible, posibles problemas de sincronización.

### Mapeo de procesos/threads a unidades de procesamiento

#### Mapeo de procesos/threads a unidades de procesamiento

Normalmente se deja al SO el mapeo de threads

- Normalmente se deja al SO el mapeo de threads (*light process*)
- Lo puede hacer el entorno o sistema en tiempo de ejecución (*runtime system* de la herramienta de programación)
- El programador puede influir

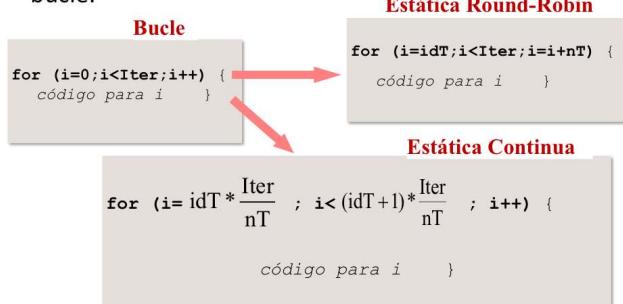
Lo puede hacer el entorno o sistema en tiempo de ejecución

El programador puede influir.

## EJEMPLO DE ASIGNACIÓN ESTÁTICA

### Asignación estática

- Asignación **estática** y explícita de las iteraciones de un bucle:



Iteraciones = 9      0 1 2 3 4 5 6 7 8

$$nT = 3$$

Identificador de hebras

$idT = 0$	0	3	6	← Asignación de hebras
1	1	4	7	↳ Round Robin
2	2	5	8	Identificador 0: 0, 3, 6 Identificador 1: 1, 4, 7 Identificador 2: 2, 5, 8

Como tenemos 9 iteraciones  
y 3 n<sup>o</sup> hebras pues se hace:

$$\frac{9}{3} = 3 \text{ por lo que se van a asignar 3 iteraciones a cada hebra.}$$

Todas las hebras ejecutan este código

Indice para saber que es lo que tiene que hacer  
Una hebra toma mucho más tiempo que otras  
sección crítica

en trabajo lo van realizando, a medida vayan

entrando a la sección crítica

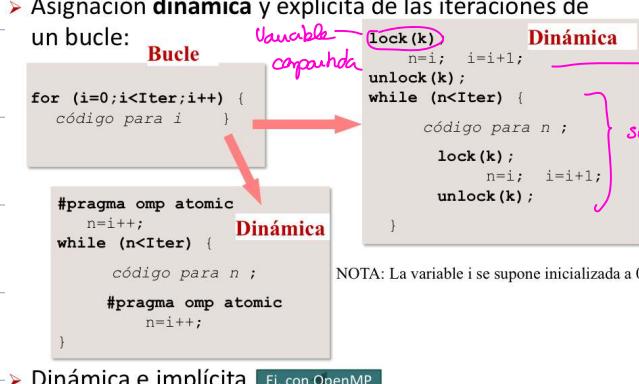
Desdoblamiento  
✓ de la carga

Hay una pausa, que la hebra dura en n<sup>o</sup> de iteración

52 | Tema 2: Programación Paralela

### Asignación dinámica

- Asignación **dinámica** y explícita de las iteraciones de un bucle:

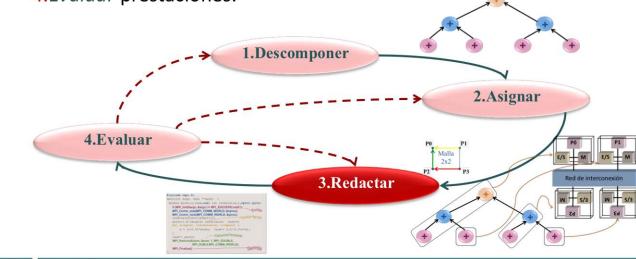


► Dinámica e implícita [El con OpenMP]

## REACTAR CÓDIGO PARALELO

### Proceso de paralelización

- Descomponer en tareas independientes.
- Asignar (planificar+mapear) tareas a procesos y/o threads.
- Redactar código paralelo.
- Evaluar prestaciones.



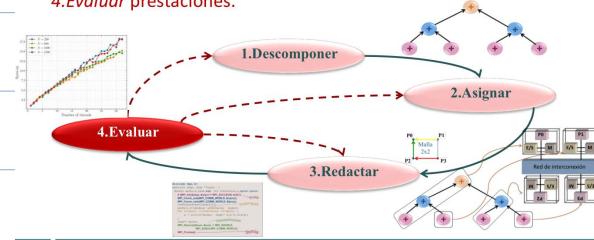
Ahora vamos a explicar

↳ Redactar código paralelo.



## Proceso de parallelización

1. Descomponer en tareas independientes.
2. Asignar (agrupar+mapear) tareas a procesos y/o threads.
3. Redactar código paralelo.
4. Evaluar prestaciones.



Aquí vamos a explicar la evaluación de prestaciones.

## Evaluación de prestaciones

### Medidas usuales

- Tiempo de respuesta
  - Real (wall-clock time, elapsed time) (/usr/bin/time)
  - Usuario, sistema, CPU time = user + sys
- Productividad
- Escalabilidad
- Eficiencia
  - Relación prestaciones/prestaciones máximas
  - Otras: Prestaciones/consumo\_potencia, prestaciones/área\_ocupada

prestaciones=desempeño

Medidas usuales

Tiempo de respuesta → cuanto tiempo tarda en ejecutarse el código  
 Productividad → cuantas operaciones se han realizado.

Medida de escalabilidad → Aumento en las prestaciones  
 La escalabilidad crece cuando se vaía el nº de procesadores, si crecen los procesadores.

Relación prestaciones máximas

Eficiencia →

$\frac{\text{prestaciones}}{\text{consumo - potencia}}$

$\frac{\text{prestaciones}}{\text{área - ocupada}}$

## GANANCIA EN PRESTACIONES : ESCALABILIDAD

### Ganancia en prestaciones. Escalabilidad

$$\text{Ganancia en prestaciones: } S(p) = \frac{\text{Prestaciones}(p)}{\text{Prestaciones}(1)} = \frac{T_s}{T_p(p)}$$

Ganancia en velocidad (Speedup)

$$T_p(p) = T_C(p) + T_O(p)$$

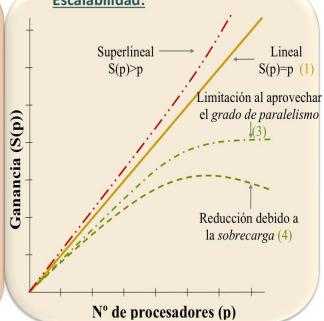
Sobrecarga (Overhead):

- Comunicación/sincronización.
- Crear/terminar procesos/threads.
- Cálculos o funciones no presentes en versión secuencial.
- Falta de equilibrio.

$$\text{Eficiencia}(p) = \frac{\text{Prestaciones}(p)}{p \times \text{Prestaciones}(1)}$$

$$E = \frac{S(p)}{p}$$

Escalabilidad:



$$S(p) = \frac{\text{Prestaciones}(p)}{\text{Prestaciones}(1)} = \frac{T_s}{T_p(p)} \rightarrow \text{Tiempo secuencial}$$

$$T_p(p) \rightarrow \text{Tiempo paralelo}$$

Tiempo de computación: Distribución de las distintas tareas y se está procesando en cada uno de los procesadores.

A esto hay que sumarle sobrecarga o overhead

$T_C(p)$   
Tiempo computo

$T_O(p)$   
Tiempo overhead.

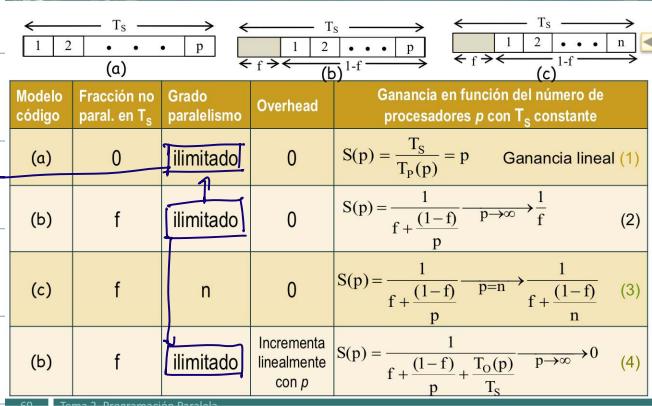
Lo se deba: comunicación / sincronización de los threads o funciones no presentes.



Júralo ahora!

## GANANCIA EN PRESTACIONES. GANANCIA MÁXIMA

### Ganancia en prestaciones. Ganancia máxima



Tengas los procesadores que tengas puedes distribuir el trabajo entre esos procesadores.

## LEY DE AMDAHL (1967)

- Ley de Amdahl: la ganancia en prestaciones utilizando  $p$  procesadores está limitada por la fracción de código que no se puede paralelizar (2):

$$S(p) = \frac{T_s}{T_p(p)} \leq \frac{T_s}{f \cdot T_s + (1-f) \cdot T_s} = \frac{p}{1 + f(p-1)} \rightarrow \frac{1}{f} (p \rightarrow \infty)$$

- $S$ : Incremento en velocidad que se consigue al aplicar una mejora. (parallelismo)
- $p$ : Incremento en velocidad máximo que se puede conseguir si se aplica la mejora todo el tiempo. (número de procesadores)
- $f$ : fracción de tiempo en el que no se puede aplicar la mejora. (fracción de t. no paralelizable)

72 Tema 2. Programación Paralela

- a) No hay parte secuencial. No hace overhead.

$$S(p) = \frac{T_s}{T_p(p)} = p \rightarrow \text{Gana si aumenta el n.º de procesadores.}$$

- b) A medida que se aumentan los procesadores las prestaciones tienden a  $\frac{1}{f}$

Prestación no paralelizable

- c)  $\rightarrow$  tamaño del problema no puedo meter mas procesadores como tal.

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

$$d) S(p) = \frac{1}{f + \frac{1-f}{p} + \frac{T_o(p)}{T_s}} \rightarrow p \rightarrow \infty$$

$$S(p) = 0 \rightarrow \text{ya que es constante entonces } \frac{1}{f+1} \text{ mas grande} \approx 0$$

## La ganancia en prestaciones utilizando $p$ procesadores

esta limitada por la fracción de código que no se puede paralelizar:

en la ley de Amdahl, el trabajo s+ se puede distribuir entre los distintos procesadores

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

incremento en velocidad  
fracción de tiempo en el que no se puede aplicar la mejora

incremento en velocidad maximo que se puede conseguir que no se puede aplicar la mejora.

## LEY DE GUSTAFSON

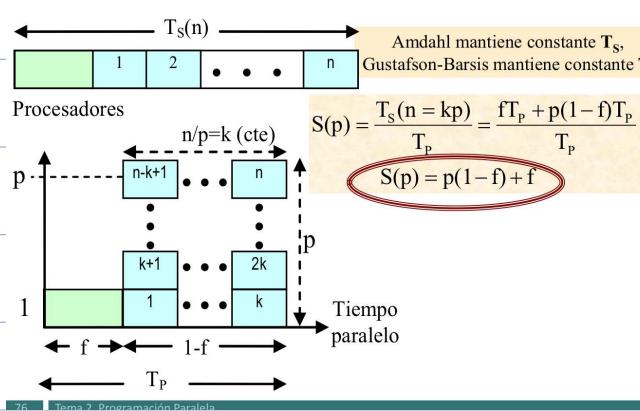
Esta ley dice que si aumenta el

problema, también van a aumentar los procesadores.

$$S(p) = p(1-f) + f$$

Podemos escalar mucho mejor y más eficiente si crece el problema

### Ganancia escalable o ley de Gustafson-Barsis



# Estudios de escalabilidad

Escala-bilidad	Qué se incrementa	Qué se mantiene constante	Qué se analiza	
Fuerte	Número de procesadores/núcleos	El tamaño total del problema a resolver <i>(Ej. Pl: nº total de intervalos de integración a repartir entre los procesadores/núcleos)</i>	Cómo evoluciona la S (se puede representar el Tp) en función de p	Mejor cuanto más se aproxime S a la ganancia lineal
Débil	Número de procesadores/núcleos	El tamaño total del problema asignado a cada procesador/núcleo <i>(Ej. Pl: nº total de intervalos de integración asignados a cada procesador/núcleo)</i>	Cómo evoluciona Tp (se podría usar S) conforme se incrementa p	Ideal: observar que Tp es cte conforme se incrementa p. Mejor cuanto menor sea la pendiente de Tp.

77 | Tema 2. Programación Paralela

Esta tabla explica la diferente escalabilidad que puede llegar a tener un problema.