

# Metodología de la Programación Paralela

Departamento de Ingeniería de Computadores

Primavera 2021



## 1 Introducción

## 2 Descomposición en tareas

- Descomposición de dominio
- Descomposición funcional
- Descomposición recursiva
- Descomposición especulativa
- Enfoques mixtos
- Factores a considerar en la descomposición

## 3 Asignación de tareas

- Esquemas de asignación estática
- Esquemas de asignación dinámica

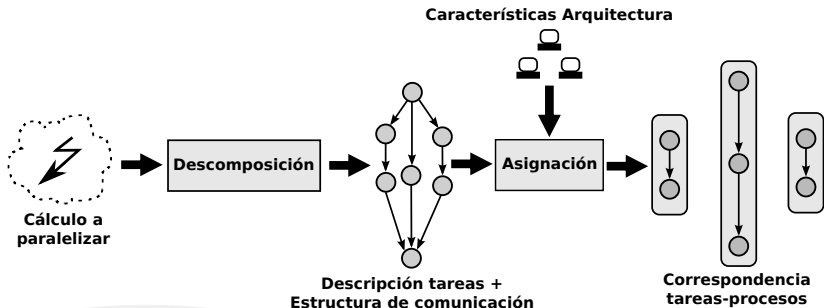
# Introducción

## Visión global del proceso de diseño paralelo

- Este tema abordará el diseño de algoritmos aprovechando los recursos paralelos disponibles.
- Un algoritmo secuencial describe una **secuencia** de operaciones de cómputo.
- El diseño de algoritmos multiprocesador debe tener en cuenta:
  - Concurrencia: coordinación de las acciones en los diferentes procesadores.
  - Asignación de datos y código a los procesadores.
  - Acceso simultáneo a datos compartidos.
  - Escalabilidad.

# Introducción

## Visión global del proceso de diseño paralelo

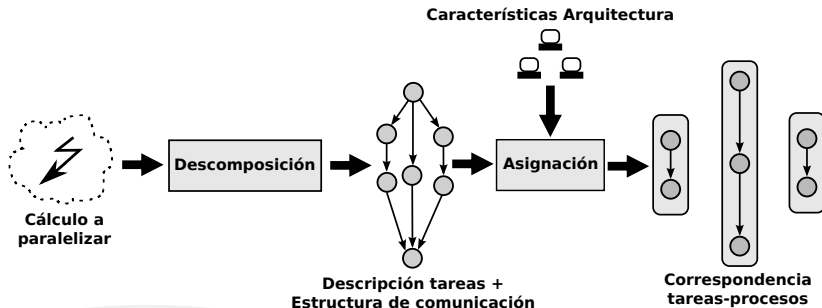


### 1 Descomposición:

- Descomponer cálculos en tareas de grano fino.
- Detectar dependencias entre tareas (comunicaciones).
- Se centra en describir el paralelismo del cálculo.

# Introducción

## Visión global del proceso de diseño paralelo



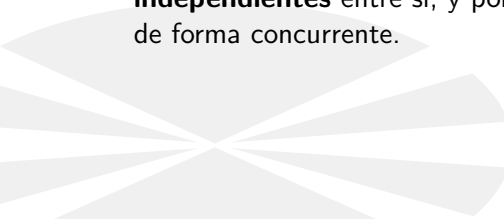
### 2 Asignación:

- Establecer qué tareas serán llevadas a cabo por cada proceso.
- Distinción: **proceso**  $\neq$  **procesador**.
- Dependiente de la arquitectura del sistema.

# Descomposición en tareas

## Definición

- Los algoritmos paralelos se basan, generalmente, en trocear un cálculo complejo en partes de menor tamaño.
- Cada uno de los trozos resultantes será una unidad de cómputo, de tamaño arbitrario, que denominaremos “tarea”.
- El objetivo es que muchas de estas tareas sean **independientes** entre sí, y por tanto puedan llevarse a cabo de forma concurrente.



# Descomposición en tareas

## Ejemplo 1: Evaluación polinomial

Supongamos que disponemos de  $m$  funciones polinomiales:

$$f^i(x) = a_{i,0} + a_{i,1} \cdot x + \dots + a_{i,n-1} \cdot x^{n-1} + a_{i,n} \cdot x^n, \quad i = 0, 1, \dots, m-1$$

Se desea evaluar todas ellas sobre un valor  $x = b$  y obtener el mínimo  $\nu(b)$  tal que:

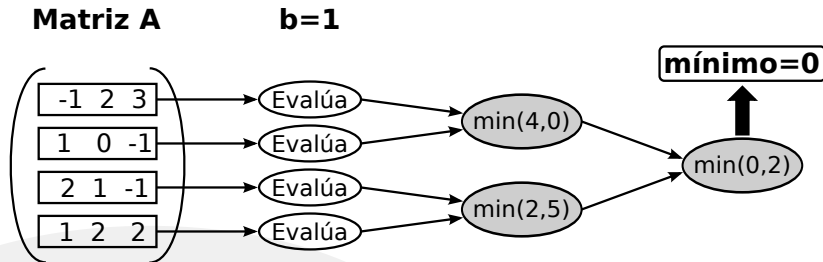
$$\nu(b) = \min_i \{f^i(b)\}, \quad i = 0, \dots, m-1$$

Para ello, se disponen los coeficientes de cada  $f^i$  en una matriz  $A$  de tamaño  $m \times (n+1)$ :

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n} \\ a_{1,0} & a_{1,1} & \dots & a_{1,n} \\ \dots & \dots & \dots & \dots \\ a_{m-1,0} & a_{m-1,1} & \dots & a_{m-1,n} \end{pmatrix}$$

# Descomposición en tareas

Evaluación polinomial: Descomposición





# Descomposición en tareas

## Ejemplo 2: Suma de vectores

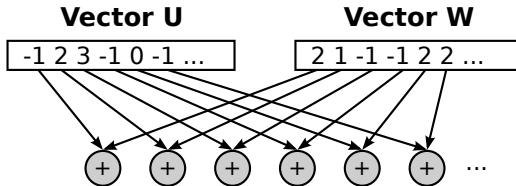
Dados dos vectores  $U, V \in \mathbb{R}^n$ :

$$U = (u_0, u_1, \dots, u_{n-1})$$

$$V = (v_0, v_1, \dots, v_{n-1})$$

Se desea calcular  $W = U + V$ :

$$W = (u_0 + v_0, u_1 + v_1, \dots, u_{n-1} + v_{n-1})$$



# Descomposición en tareas

## Granularidad de la descomposición

- Las tareas resultantes de una descomposición pueden tener diferentes costes computacionales.
- Granularidad fina: elevado número de pequeñas tareas.
- Granularidad gruesa: pocas tareas de gran tamaño.

### Ej. Evaluación polinomial con $m=2000$

- Granularidad gruesa: cuatro tareas de evaluación, 500 polinomios cada una; tres tareas de cálculo de mínimo.
- Granularidad fina: 2000 tareas de evaluación, una por polinomio; 1999 tareas de cálculo de mínimo.

# Descomposición en tareas

## Granularidad de la descomposición

### Límite inferior en la granularidad

- Existe un límite sobre lo fina que puede ser la granularidad de la descomposición de tareas del cálculo.
- Ej. Suma de vectores: una tarea por cada  $w_i = u_i + v_i$  ( $n$  tareas).
- Ej. Evaluación polinomial: una tarea por cada término polinómico ( $m \times (n + 1)$  tareas);  $n$  tareas de suma por polinomio ( $m \times n$  tareas);  $(m - 1)$  tareas de comparación.

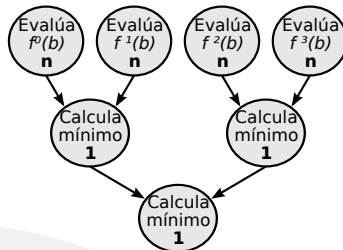
# Descomposición en tareas

## Grafo de dependencias

- El objetivo de una descomposición es que las tareas sean independientes, proporcionando oportunidades de paralelización.
- No obstante, existirán dependencias entre determinadas tareas (ej. evaluación polinomial y cálculo del mínimo global).
- Como consecuencia, ciertas tareas no pueden comenzar mientras las tareas de las que dependen no hayan terminado.
- **Def.** *Grafo de dependencias:*
  - Grafo  $G = \{N, A\}$ , **acíclico** y **dirigido**.
  - Cada nodo  $n \in N$  representa una tarea en la descomposición.
    - Opcionalmente puede etiquetarse cada nodo  $n$  del grafo con una estimación de su coste,  $w(n)$ .
  - Cada arista  $a \in A$  es un par ordenado de nodos  $n_o, n_d \in N$  representando la existencia de una dependencia entre dos tareas: la tarea origen  $n_o$  debe ejecutarse previamente a la tarea destino  $n_d$ .

# Descomposición en tareas

Ejemplo: Evaluación polinomial



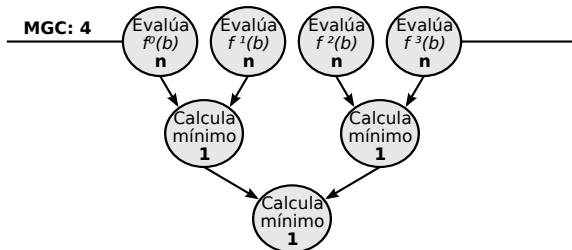
# Descomposición en tareas

Grado de concurrencia: máximo

- Grado de concurrencia: medida de cómo de paralelizable es un algoritmo, atendiendo a las dependencias entre las tareas constituyentes.
- **Def.** *Máximo grado de concurrencia*: mayor número de tareas de una descomposición que pueden ser ejecutadas al mismo tiempo.
- En muchos casos puede determinarse de forma sencilla observando el grafo de dependencias.

# Descomposición en tareas

Grado de concurrencia: máximo



- El máximo grado de concurrencia es un indicador de lo paralelizable que es una determinada descomposición, pero sólo de forma puntual. El grado **medio** de concurrencia será un mejor indicador global.



# Descomposición en tareas

## Grado medio de concurrencia

- **Def.** *Camino crítico*: el más largo de los caminos del grafo de dependencias que van desde un nodo de comienzo hasta un nodo de finalización del cómputo.
- Sea  $C \subset N$  el conjunto de nodos del camino crítico, su longitud  $L$  puede expresarse matemáticamente como la suma de los costes de cada nodo individual de  $C$ :

$$L = \sum_{n \in C} w(n)$$

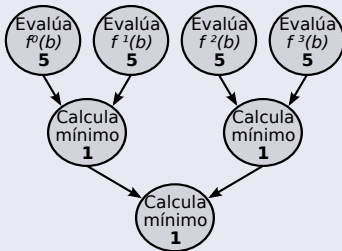
- **Def.** *Grado medio de concurrencia* ( $M$ ):

$$M = \sum_{n \in N} \frac{w(n)}{L}$$

# Descomposición en tareas

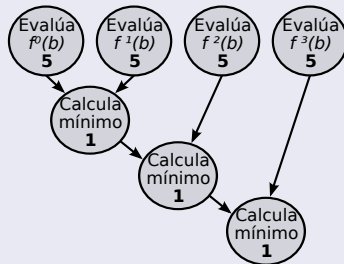
Grado medio de concurrencia: ejemplos

Grafo A



$$M = \frac{23}{7} = 3.28$$

Grafo B

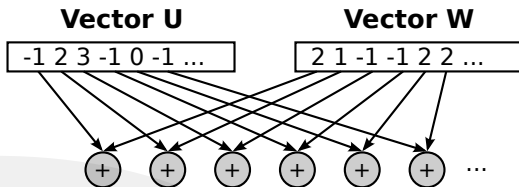


$$M = \frac{23}{8} = 2.875$$

# Descomposición en tareas

## Granularidad y grado medio de concurrencia

- Habitualmente, cuanto menor sea la granularidad de una descomposición mayor será el grado medio de concurrencia. Por ello es deseable buscar descomposiciones del grano más fino posible.
- Ej. Suma de vectores.

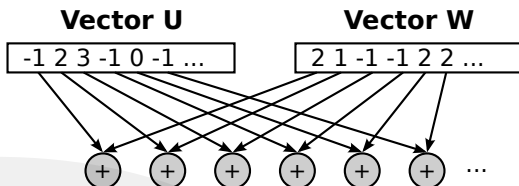


$$M = \sum_{i=1}^n \frac{w(n_i)}{L} = \sum_{i=1}^n \frac{1}{1} = n$$

# Descomposición en tareas

## Granularidad y grado medio de concurrencia

- Habitualmente, cuanto menor sea la granularidad de una descomposición mayor será el grado medio de concurrencia. Por ello es deseable buscar descomposiciones del grano más fino posible.
- Ej. Suma de vectores.



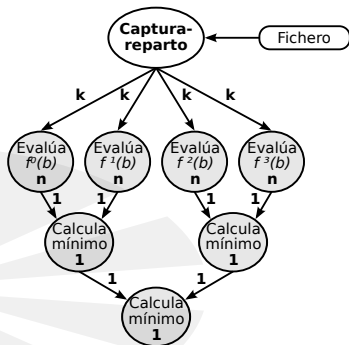
$$M = \sum_{i=1}^n \frac{w(n_i)}{L} = \sum_{i=1}^n \frac{1}{1} = n$$

¿Y creando una tarea por cada dos elementos?

# Descomposición en tareas

## Representación de las comunicaciones

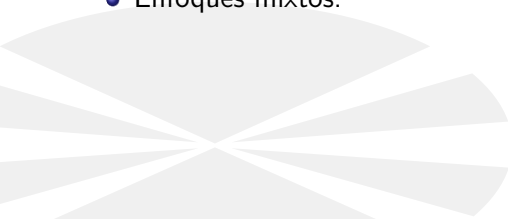
- Es posible etiquetar las aristas del grafo de dependencias para incluir información del coste de las comunicaciones.
- A la hora de implementar un algoritmo, el grafo de dependencias puede sufrir modificaciones debido a la inclusión de comunicaciones adicionales. P. ej. evaluación polinomial:



# Descomposición en tareas

## Técnicas de descomposición

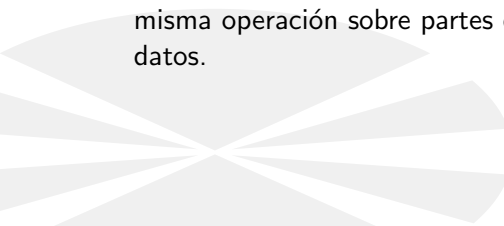
- Descomposición de dominio.
- Descomposición funcional dirigida por el flujo de datos.
- Descomposición recursiva.
- Descomposición especulativa.
- Enfoques mixtos.



# Descomposición de dominio

## Definición

- Técnica más usada sobre grandes estructuras de datos.
- Dos fases:
  - Troceo de datos, centrándose en la estructura más grande, o en la más usada.
  - Asociación de la computación a cada subdominio de datos.
- Se utiliza cuando es posible resolver un problema aplicando la misma operación sobre partes diferentes de su dominio de datos.



# Descomposición de dominio

Ejemplo: multiplicación matriz-vector

Sean:

$$A \in \mathbb{R}^{n \times n}, A = (a_{ij})$$

$$b \in \mathbb{R}^n, b = [b_0, \dots, b_{n-1}]^T$$

El problema consiste en calcular el vector  $x = A.b$ , donde:

$$x \in \mathbb{R}^n, x = [x_0, \dots, x_{n-1}]^T$$

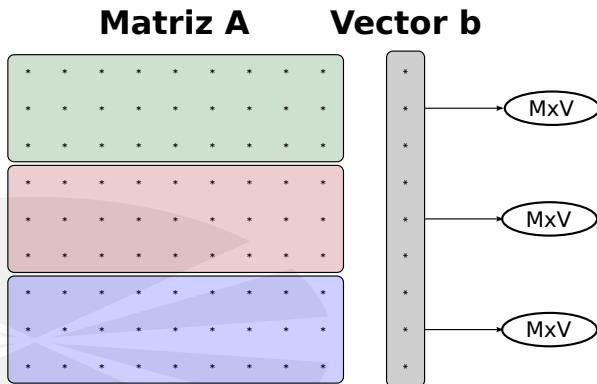
$$x_i = \sum_{j=0}^{n-1} a_{ij} b_j, \quad i = 0, \dots, n-1$$



# Descomposición de dominio

Ejemplo: multiplicación matriz-vector

- Dividir la matriz en  $m$  bloques de  $n/p$  filas (por simplicidad, suponemos que  $n \bmod p = 0$ ).
- Cada tarea se encargará de calcular  $n/p$  filas del vector  $x$ .



# Descomposición de dominio

## Regla del propietario

### Regla del propietario (owner-computes rule)

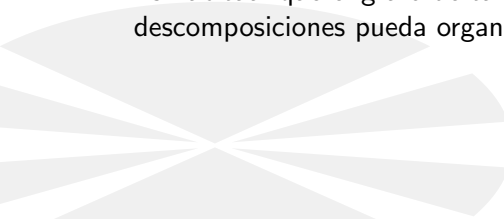
*La tarea a la que se le asigna un determinado dato es responsable de realizar todos los cálculos asociados al mismo*

- Descomposición centrada en la entrada: todas las computaciones que usen los datos de entrada serán realizadas por la tarea.
- D. c. e. l. salida: la salida es computada por el proceso al que se le asigna la misma.
- Regla general: no siempre es posible.

# Descomposición funcional

## Definición

- Consiste en descomponer el cálculo en tareas atendiendo a las partes diferenciadas del mismo.
- Pasos:
  - 1 Identificar las fases funcionales del cálculo a paralelizar.
  - 2 Asignar una tarea para la realización de cada fase.
- Es habitual que el grafo de tareas resultante de este tipo de descomposiciones pueda organizarse como un *pipeline*.



# Descomposición funcional

Ejemplo: cálculo vectorial

Sean:

$$A \in \mathbb{R}^{n \times n}, A = (a_{ij})$$

$$\nu \in \mathbb{R}^n, \nu = [\nu_0, \dots, \nu_n]^T$$

Se desea calcular:

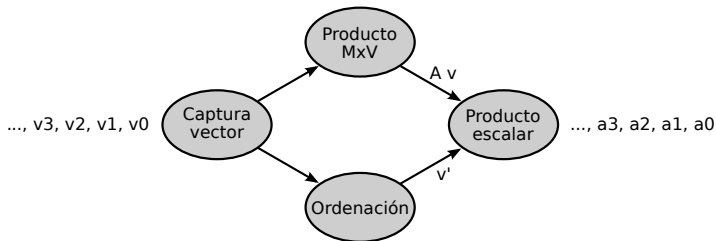
$$a = (A\nu) \cdot \hat{\nu}$$

Donde:

$$\hat{\nu} \in \mathbb{R}^n, \{\nu\} = \{\hat{\nu}\}, \hat{\nu} = [\hat{\nu}_0, \dots, \hat{\nu}_n]^T, \hat{\nu}_0 \leq \hat{\nu}_1 \leq \dots \hat{\nu}_n$$

# Descomposición funcional

Ejemplo: cálculo vectorial



- Cada tarea puede implementarse como algoritmo secuencial o descomponerse en subtareas de grano más fino.
- Si no tenemos un único vector  $v$  de entrada, sino una secuencia de ellos, el grafo se convierte en un cauce segmentado donde en cada momento se realizan cuatro tareas de forma simultánea.

# Descomposición funcional

Ejemplo: cálculo vectorial

	Captura	MxV	Ord.	P. Esc.
$t_0$	$v_0$	—	—	—
$t_1$	$v_1$	$Av_0$	$v'_0$	—
$t_2$	$v_2$	$Av_1$	$v'_1$	$a_0$
$t_3$	$v_3$	$Av_2$	$v'_2$	$a_1$
$t_4$	$v_4$	$Av_3$	$v'_3$	$a_2$
$t_5$	$v_5$	$Av_4$	$v'_4$	$a_3$
$t_6$	$v_6$	$Av_5$	$v'_5$	$a_4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

# Descomposición recursiva

## Definición

- También conocida como *divide-y-vencerás*.
- Pasos:
  - 1 Si un subproblema ha alcanzado el tamaño crítico, obtener su solución parcial.
  - 2 De lo contrario, descomponerlo en subproblemas.
  - 3 Combinar los resultados de los subproblemas para obtener la solución al problema original.
- El paralelismo surge de la independencia entre las tareas de resolución de los subproblemas.

# Descomposición recursiva

Ejemplo: cálculo de área

Supongamos que deseamos aproximar numéricamente el valor de:

$$I = \int_a^b f(x) dx$$

Una aproximación al valor de la integral en el intervalo  $[p, q]$  viene dada por:

$$A(p, q) = \frac{(q - p)(f(q) + f(p))}{2}$$

Se puede estimar el error troceando el intervalo en dos,  $[p, r]$ ,  $[r, q]$  y estimando:

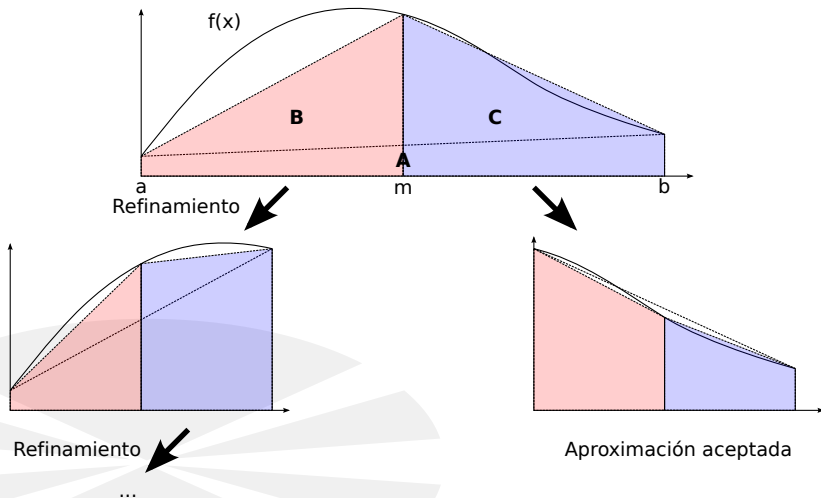
$$Err = A(p, q) - (A(p, r) + A(r, q))$$

Si el error es suficientemente pequeño, se acepta  $A(p, q)$  como aproximación. En caso contrario, se divide el intervalo en dos y se repiten los cálculos.



# Descomposición recursiva

Ejemplo: cálculo de área



# Descomposición especulativa

## Definición

- En ocasiones un problema no puede dividirse en tareas independientes. No obstante, sí puede dividirse funcionalmente en fases que se ejecutan condicionalmente dependiendo de los resultados de fases anteriores.
- La descomposición especulativa consiste en comenzar la ejecución de tareas condicionales excluyentes entre sí sin esperar a la finalización de las tareas de las que depende la selección de una de ellas.
- Una vez la tarea de selección haya finalizado, es posible seleccionar el resultado correcto de entre los obtenidos por las tareas ejecutadas de forma especulativa.

# Descomposición especulativa

## Definición

- Si no es posible comenzar todas las tareas alternativas se seleccionarán aquéllas que se ejecutarán más probablemente. Si la tarea correcta no se ejecuta especulativamente, será necesario cancelar el trabajo realizado e iniciarla.
- El coste computacional total es superior, pero puede proporcionar mejoras en tiempo de ejecución.
- Aplicación alternativa: lanzar en paralelo diferentes algoritmos de resolución de un mismo problema, quedándose con la solución que se obtenga más rápidamente y cancelando el resto de tareas en ejecución.

# Descomposición especulativa

## Ejemplo

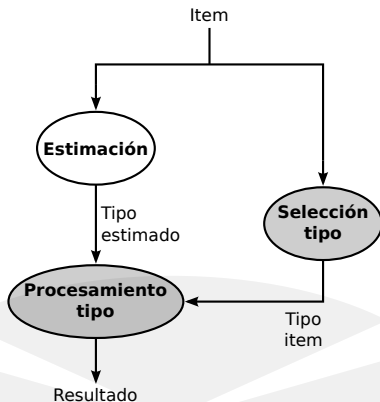
- Dependiendo del tipo de ítem de entrada se realiza un procesamiento diferente.
- El tiempo de clasificación es aproximadamente igual al de procesamiento,  $\sim T$ .
- El tipo de un elemento puede estimarse con coste prácticamente nulo y tasa de acierto del 80%.



$$T_{sec} \simeq T + T = 2T$$

# Descomposición especulativa

## Ejemplo



- Aprovechamos la estimación eficiente del elemento.
- Se acierta en un 80% de los casos. En el resto, debe cancelarse la ejecución especulativa y comenzar el procesamiento correcto.
- El speed-up es cercano a 2, siempre que dispongamos de dos unidades de procesamiento.

$$T_{esp} \simeq 0.8T + 0.2(2T) = 1.2T$$

# Descomposición en tareas

## Enfoques mixtos

- Es común aplicar diferentes esquemas de descomposición paralela a diferentes partes de un problema. P. ej:
  - En el ejemplo de evaluación polinomial, la evaluación se ha analizado a través de descomposición de dominio, mientras que para el cálculo del mínimo se ha usado descomposición recursiva.
  - En el ejemplo de descomposición funcional, cada una de las tareas obtenidas es de grano grueso. Podrían ser descompuestas adicionalmente (ej. producto matriz-vector: descomposición de dominio, ordenación: descomposición recursiva; producto escalar: descomposición de dominio).
- En general, diferentes fases de cómputo pueden operar sobre diferentes estructuras de datos siguiendo diferentes descomposiciones.

# Descomposición en tareas

## Factores a considerar

### Creación de tareas

- **Creación estática:** todas las tareas que participan en el algoritmo se crean al inicio de su ejecución.
- **Creación dinámica:** las tareas se crean durante la ejecución del algoritmo. Útil cuando no se conoce la estructura del grafo de dependencias de forma estática. Sí se define la política de creación de nuevas tareas.

# Descomposición en tareas

## Factores a considerar

### Número de tareas obtenidas

- Tras la descomposición en tareas, se procede a asignarlas a procesadores.
- Interesa disponer de un número de tareas que permita tener flexibilidad en la asignación. En general, es aconsejable que

$$\#tareas \gg \#procesadores$$

- Es deseable que el número de tareas escale con el tamaño del problema.



# Descomposición en tareas

## Factores a considerar

### Balanceo de carga

- Es deseable que las tareas obtenidas tengan un coste similar, para facilitar la etapa de asignación de tareas.
- Es necesario evitar que se produzcan desequilibrios entre la carga computacional de los diferentes procesadores.
- La descomposición de dominio habitualmente lleva a algoritmos equilibrados. En la descomposición funcional, recursiva y especulativa dependerá del análisis realizado.

# Asignación de tareas

## Definición

- Tras descomponer un algoritmo paralelo en tareas, debe definirse una correspondencia entre éstas y los procesos que las ejecutarán. Denominamos a esta correspondencia “asignación de tareas”.
- Es importante distinguir en este nivel entre procesos y procesadores. No usamos el término “procesos” en el sentido técnico comúnmente usado en los sistemas operativos, sino que en nuestro contexto denomina a una unidad lógica de cómputo capaz de ejecutar tareas computacionales (hilos, procesos UNIX).

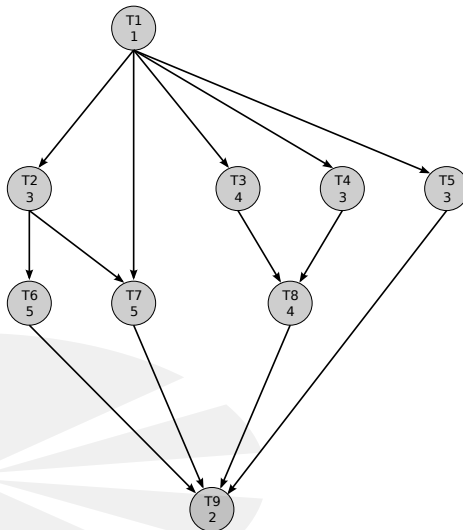
# Asignación de tareas

## Definición

- Este paso supone un paso descendente en el nivel de abstracción. La asignación de tareas se hará teniendo en cuenta algunas de las características de la arquitectura paralela concreta a la que adaptamos el algoritmo (p. ej. crear más procesos que procesadores en una arquitectura que no permite la ejecución eficiente de pequeñas tareas será contraproducente).
- La asignación de tareas incluye también la decisión sobre el orden de ejecución de las mismas.

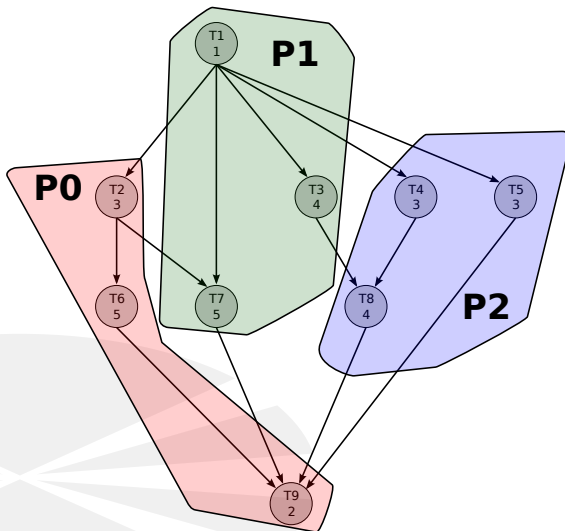
# Asignación de tareas

## Objetivos



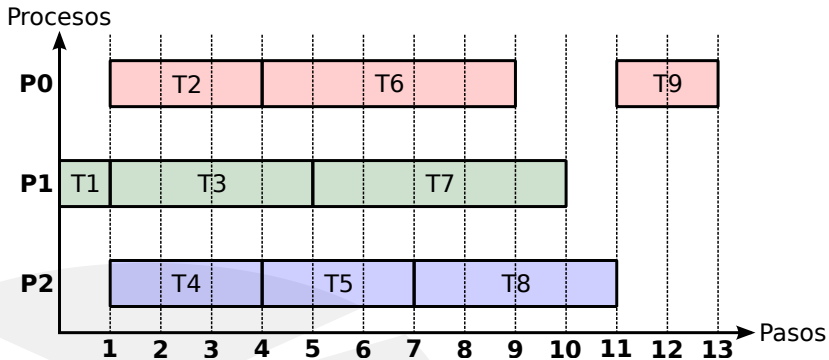
# Asignación de tareas

## Objetivos



# Asignación de tareas

## Objetivos



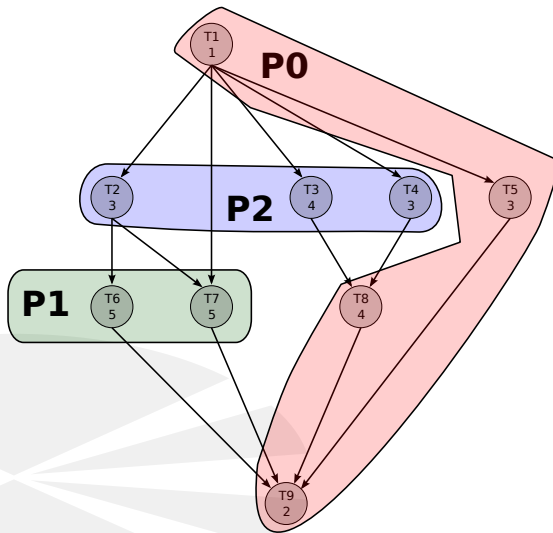
# Asignación de tareas

## Objetivos

- Minimizar el tiempo de computación, a través de la ejecución concurrente de tareas.
- Minimizar el tiempo de comunicaciones, asignando tareas que se comuniquen mucho al mismo proceso.
- Minimizar el tiempo de inactividad de los procesos. Fuentes de inactividad:
  - Desequilibrios de carga.
  - Dependencias entre tareas.
- Estos tres objetivos suelen estar muy interrelacionados, por lo que será necesario alcanzar un punto de compromiso entre los 3 para lograr una buena asignación.

# Asignación de tareas

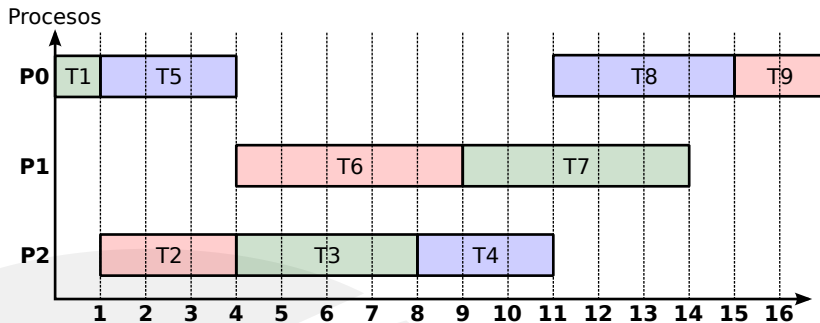
## Objetivos





# Asignación de tareas

## Objetivos



# Esquemas de asignación estática

- Cuando el grafo de tareas se conoce con detalle de forma estática (número de tareas, coste, relaciones, comunicaciones), las decisiones de asignación pueden tomarse antes de la ejecución.
- Su mayor ventaja es que no añaden sobrecarga en tiempo de ejecución.
- El diseño y la implementación es sencillo.
- Dos tipos fundamentales:
  - Esquemas para descomposición de dominio: distribución en bloques.
  - Esquemas para descomposición funcional o recursiva: grafos de dependencias estáticos.

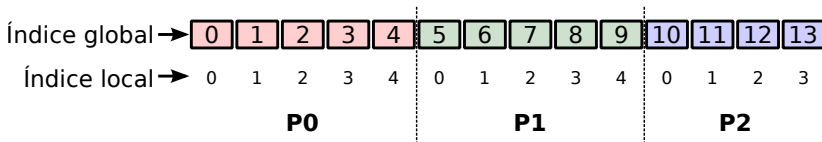
# Distribución de matrices por bloques

- En numerosos algoritmos sobre matrices el cálculo de una entrada de la matriz depende de entradas contiguas de la misma (localidad espacial).
- La descomposición de dominio de estos problemas dará lugar a tareas que se comunican con las tareas que representan a celdas vecinas.
- La asignación de estas tareas contiguas a un mismo proceso permite reducir en gran medida las necesidades de comunicación del algoritmo.
- Esto provoca la reducción del número de comunicaciones, así como un mejor aprovechamiento de la jerarquía de memoria (registros CPU, caché, memoria principal de cada procesador).

# Distribución de matrices por bloques

## Matriz unidimensional (vector)

Si tenemos  $p$  procesos, se asigna a cada proceso un bloque de tamaño  $m_b = \left\lceil \frac{n}{p} \right\rceil$ :

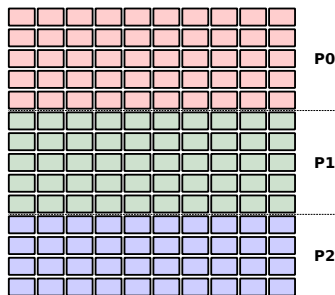


- El proceso  $p - 1$  recibirá entre 1 y  $m_b$  elementos.  
Concretamente  $\hat{m}_b = (n - m_b(p - 1))$ .
- A pesar de este pequeño desequilibrio, esta asignación será óptima cuando todas las celdas de la matriz llevan asociada aproximadamente la misma carga computacional.
- El elemento  $i$  del vector se asigna al proceso  $\left\lfloor \frac{i}{m_b} \right\rfloor$ , ocupando la posición  $(i \bmod m_b)$  en su vector local.

# Distribución de matrices por bloques

Distribución de una matriz bidimensional: distribución por bloques de filas

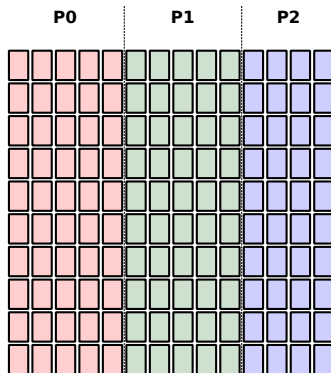
- Cada proceso recibe un bloque de  $m_b = \left\lceil \frac{n}{p} \right\rceil$  filas de la matriz, a excepción del último, que recibirá  $\hat{m}_b = (n - m_b(p - 1))$ .
- La fila  $i$  se encontrará en el proceso  $\left\lfloor \frac{i}{m_b} \right\rfloor$ , ocupando la posición  $(i \bmod m_b)$  en la submatriz local.
- El índice de columna  $j$  no se ve alterado al transponerse a la submatriz local, pues las filas se reparten enteras.



# Distribución de matrices por bloques

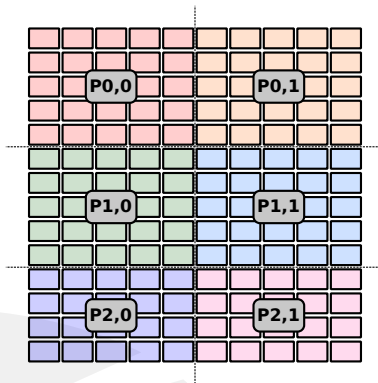
Distribución de una matriz bidimensional: distribución por bloques de columnas

- Cada proceso recibe un bloque de  $m_b = \left\lceil \frac{m}{p} \right\rceil$  columnas de la matriz, a excepción del último, que recibirá  $\hat{m}_b = (n - m_b(p - 1))$ .
- La columna  $j$  se encontrará en el proceso  $\left\lfloor \frac{j}{m_b} \right\rfloor$ , ocupando la posición  $(j \bmod m_b)$  en la submatriz local.
- El índice de fila  $i$  no se ve alterado al transponerse a la submatriz local, pues las columnas se reparten enteras.



# Distribución de matrices por bloques

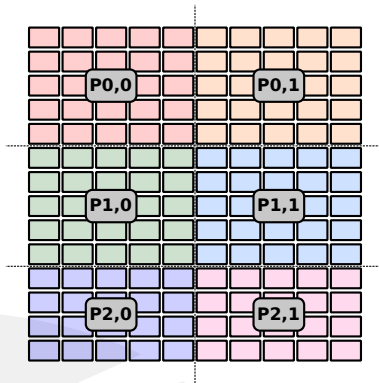
Distribución bidimensional de una matriz bidimensional



- Los procesos se organizan lógicamente como una malla bidimensional.

# Distribución de matrices por bloques

Distribución bidimensional de una matriz bidimensional

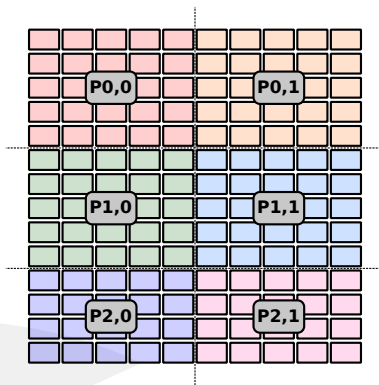


- $p = p_x p_y$ , siendo  $p_{ij}$  ( $i = 0, \dots, p_x - 1$ ,  $j = 0, \dots, p_y - 1$ ) el proceso que ocupa la posición  $(i, j)$  en dicha malla.



# Distribución de matrices por bloques

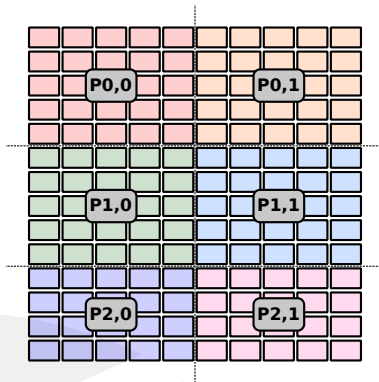
Distribución bidimensional de una matriz bidimensional



- Cada proceso recibe una submatriz con un máximo de  $m_{b_x} = \left\lceil \frac{n}{p_x} \right\rceil$  filas y  $m_{b_y} = \left\lceil \frac{m}{p_y} \right\rceil$  columnas.

# Distribución de matrices por bloques

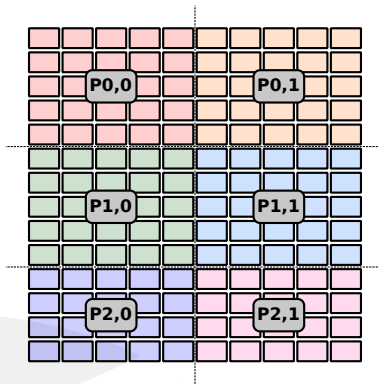
Distribución bidimensional de una matriz bidimensional



- Los procesos en la fila  $p_{x-1}$  de la malla reciben  $\hat{m}_{b_x} = n - m_{b_x}(p_x - 1)$  subfilas.

# Distribución de matrices por bloques

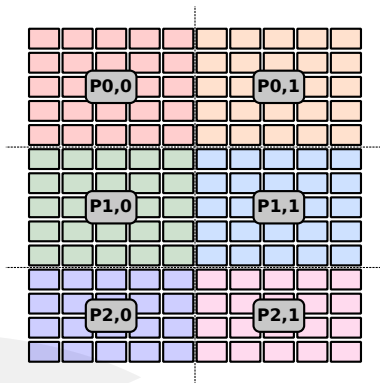
Distribución bidimensional de una matriz bidimensional



- Los procesos en la columna  $p_y - 1$  de la malla reciben  $\hat{m}_{b_y} = m - m_{b_y}(p_y - 1)$  subcolumnas.

# Distribución de matrices por bloques

Distribución bidimensional de una matriz bidimensional



- El elemento  $(i, j)$  de la matriz global se encontrará en el procesador  $\left( \left\lfloor \frac{i}{m_{bx}} \right\rfloor, \left\lfloor \frac{j}{m_{by}} \right\rfloor \right)$ , donde ocupará la posición  $(i \bmod m_{bx}, j \bmod m_{by})$  en la submatriz local.

# Distribución de matrices por bloques

Ejemplo: diferencias finitas

Sea  $A$  una matriz cuadrada que representa el estado de un determinado sistema,  $A \in \mathbb{R}^{n \times n}$ ,  $A = (a_{i,j})$ .

En el instante ( $t = 0$ ) partimos de una matriz  $A^{(0)} \in \mathbb{R}^{n \times n}$ . Se realiza un cálculo iterativo en el que el estado del sistema tras un tiempo ( $k\Delta t$ ),  $A^{(k)}$  se calcula como:

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \Delta t \left( \frac{a_{i+1,j}^{(k)} - a_{i-1,j}^{(k)}}{0.1} + \frac{a_{i,j+1}^{(k)} - a_{i,j-1}^{(k)}}{0.02} \right)$$

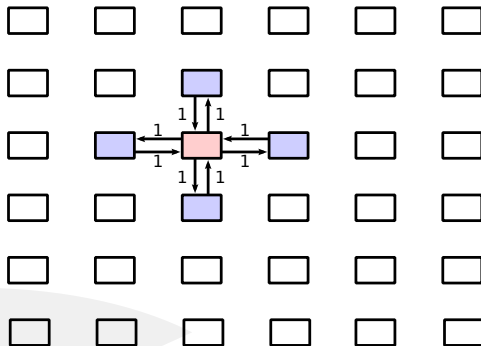
Los valores que quedan fuera del rango de definición de la fórmula se calculan como:

$$a_{n,j}^{(k)} = a_{0,j}^{(k)}, \quad a_{i,n}^{(k)} = a_{i,0}^{(k)}, \quad a_{-1,j}^{(k)} = a_{n-1,j}^{(k)}, \quad a_{i,-1}^{(k)} = a_{i,n-1}^{(k)}$$

# Distribución de matrices por bloques

Ejemplo: diferencias finitas

Si asignamos una tarea a cada posición en la matriz:



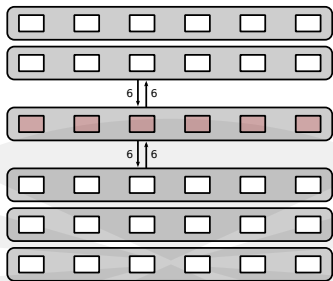
A cada iteración:

- 4 envíos por tarea, 1 elemento por envío.
- $(4 \times n \times n)$  envíos y elementos transferidos por iteración.

# Distribución de matrices por bloques

## Ejemplo: diferencias finitas

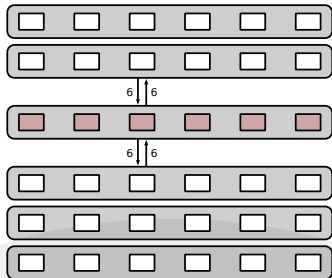
Asignando la matriz por bloques de filas a  $p$  procesadores, los envíos sólo se producen en las fronteras entre los bloques asignados a cada procesador. Asumiendo que  $n \bmod p = 0$ , por simplicidad, tenemos:



- $p - 1$  bloques de  $m_b = \left\lceil \frac{n}{p} \right\rceil$  y 1 bloque de  $\hat{m}_b = n - m_b(p - 1)$  filas.
- Cada bloque debe comunicar  $n$  datos (una fila completa) a los procesos inmediatamente superior e inferior en la malla unidimensional:
  - 2 envíos por tarea,  $n$  elementos por envío.
  - $(p \times 2)$  envíos por iteración.
  - $(p \times 2 \times n)$  elementos por iteración.

# Distribución de matrices por bloques

Ejemplo: diferencias finitas



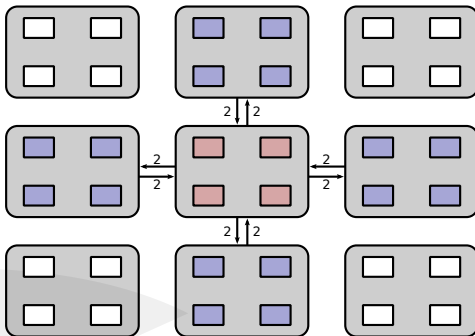
- El número de mensajes se ha reducido en proporción a  $\frac{n^2}{p}$ , y la cantidad de datos enviados en proporción a  $\frac{n}{p}$ .



# Distribución de matrices por bloques

Ejemplo: diferencias finitas

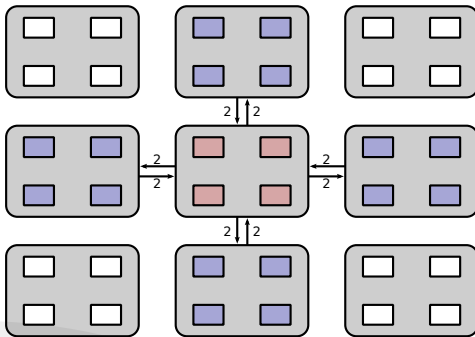
Asignando la matriz a una malla de  $p = p_x p_y$  procesadores:



- Cada proceso debe comunicarse con los 4 procesos vecinos.
- Por simplicidad, asumimos  $n \bmod p_x = n \bmod p_y = 0$ .

# Distribución de matrices por bloques

Ejemplo: diferencias finitas

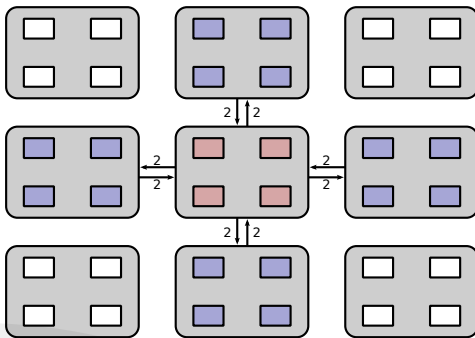


- Por filas, se envían:

- 2 envíos por tarea,  $m_{b_x} = \left\lceil \frac{n}{p_x} \right\rceil$  elementos por envío.
- $(p \times 2)$  envíos por iteración.
- $(p \times 2 \times m_{b_x})$  elementos por iteración.

# Distribución de matrices por bloques

Ejemplo: diferencias finitas

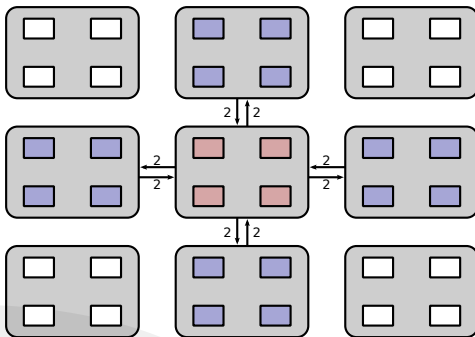


- Por columnas, se envían:

- 2 envíos por tarea,  $m_{b_y} = \left\lceil \frac{n}{p_y} \right\rceil$  elementos por envío.
- $(p \times 2)$  envíos por iteración.
- $(p \times 2 \times m_{b_y})$  elementos por iteración.

# Distribución de matrices por bloques

Ejemplo: diferencias finitas



- En total, se envían:
  - $(4 \times p)$  envíos por iteración.
  - $(2 \times p \times (m_{b_x} + m_{b_y}))$  elementos por iteración.

# Distribución de matrices por bloques

## Ejemplo: diferencias finitas

Las tareas por bloques de filas realizan:

- $m_b n$  veces más computación que las tareas iniciales.
- $\frac{n}{2}$  veces más elementos enviados que las tareas iniciales.

Las tareas por bloques bidimensionales realizan:

- $m_{b_x} m_{b_y}$  veces más computación que las tareas iniciales.
- $\frac{m_{b_x} + m_{b_y}}{2}$  veces más elementos enviados que las tareas iniciales.

Es decir, la computación realizada crece más rápido que el número de envíos, mejorando la localidad. En general:

### Efecto volumen-superficie

La carga computacional de la tarea agrupada por bloques aumenta de forma proporcional a la superficie del subdominio, mientras que el volumen de comunicaciones aumenta de forma proporcional a su perímetro.

# Distribución de matrices por bloques

Ejemplo: diferencias finitas

Ejemplo: matriz  $8 \times 8$ . Comunicaciones por iteración:

- Tareas monoelemento:
  - $4 \times 8 \times 8 = 256$  envíos de 1 elemento.
  - $4 \times 8 \times 8 = 256$  elementos comunicados.
- Distribución en bloques de filas,  $p = 8$  (1 fila por proceso):
  - $2 \times 8 = 16$  envíos de 8 elementos.
  - $2 \times 8 \times 8 = 128$  elementos comunicados.
- Distribución bidimensional en malla de  $4 \times 4$  procesadores:
  - $4 \times 16 = 64$  envíos de 2 elementos.
  - $2 \times 16 \times (2 + 2) = 128$  elementos comunicados.

# Distribución de matrices por bloques

## Ventajas del incremento de dimensionalidad

- A mayor dimensionalidad mayor flexibilidad para asignar los elementos de la matriz. En una distribución bloque unidimensional el número de procesos utilizables es igual al tamaño de la dimensión distribuida (en el ejemplo anterior, 8 procesos frente a 16 en la distribución bidimensional).
- Aumenta la frecuencia de las comunicaciones (su número se duplica por cada dimensión extra). Sin embargo, el volumen de comunicaciones por proceso disminuye. En la malla de computadores, las comunicaciones se reparten mejor y se aprovecha mejor la localidad de los cálculos.

# Distribución de matrices por bloques

## Generalización multidimensional

En general, siguiendo la misma técnica, una matriz  $s$ -dimensional puede distribuirse por bloques sobre  $p$  procesos troceando hasta  $k$  dimensiones, con  $k \leq s$ , y organizando los procesos como una malla  $k$ -dimensional:

- Tamaño de la submatriz local en la dimensión  $d$  ( $0 \leq d < s$ ):

$$m_{b_d} = \begin{cases} \left\lceil \frac{n_d}{p_d} \right\rceil & \text{si } d < k \\ n_d & \text{en otro caso} \end{cases}$$

$$\hat{m}_{b_d} = \begin{cases} n_d - m_{b_d}(p_d - 1) & \text{si } d < k \\ n_d & \text{en otro caso} \end{cases}$$



# Distribución de matrices por bloques

## Generalización multidimensional

Localización de un elemento:

Índice global

$$(i_0, \dots, i_{s-1})$$

---

Proceso asignado

$$\left( \frac{i_0}{m_{b_0}}, \dots, \frac{i_{k-1}}{m_{b_{k-1}}} \right)$$

---

Índice local

$$(i_0 \bmod m_{b_0}, \dots, i_{k-1} \bmod m_{b_{k-1}}, i_k, \dots, i_{s-1})$$

# Distribución de matrices por bloques

## Distribución cíclica por bloques

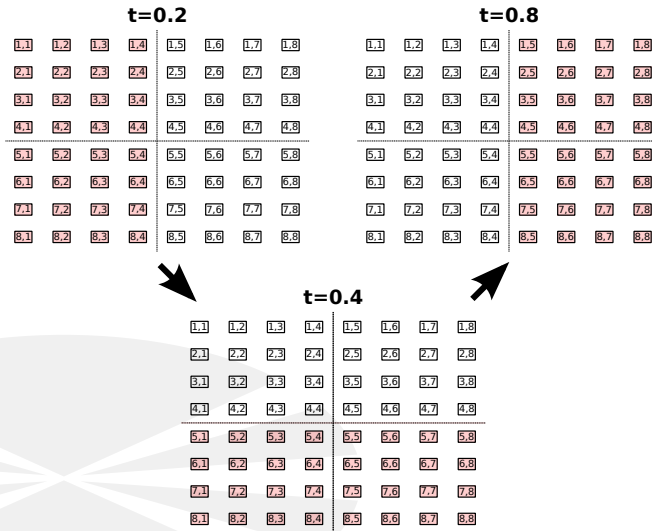
Supongamos que se modifica la fórmula de diferencias finitas del ejemplo anterior para hacerla variable en el tiempo de la siguiente manera:

- Si  $t \leq 0.25$ ,  $a_{i,j}^{(k+1)}$  sólo se actualiza si  $j < \frac{n}{2}$  (mitad izquierda de la matriz), de lo contrario permanece inalterada.
- Si  $0.25 < t \leq 0.5$ ,  $a_{i,j}^{(k+1)}$  sólo se actualiza si  $i \geq \frac{n}{2}$  (mitad inferior de la matriz), de lo contrario permanece inalterada.
- Si  $t > 0.5$ ,  $a_{i,j}^{(k+1)}$  sólo se actualiza si  $j \geq \frac{n}{2}$  (mitad derecha de la matriz), de lo contrario permanece inalterada.

En este caso, la distribución bloque provoca grandes desequilibrios de carga.

# Distribución de matrices por bloques

## Distribución cíclica por bloques

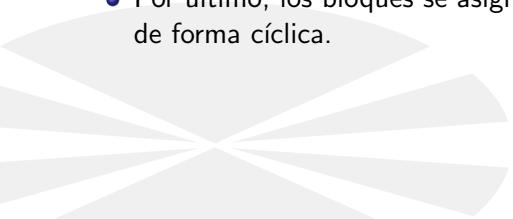


# Distribución de matrices por bloques

## Distribución cíclica por bloques

En estos casos la *distribución cíclica por bloques* consigue equilibrar mejor la carga de trabajo:

- Definimos un tamaño de bloque en cada una de las dimensiones de la matriz.
- La matriz se trocea en bloques de dicho tamaño.
- Por último, los bloques se asignan a los procesos de la malla de forma cíclica.



# Distribución de matrices por bloques

## Distribución cíclica por bloques

Distribución cíclica por bloques de dos elementos de una matriz de 8x8 (2x2) procesadores.

**t=0.2**

1,1	1,2	1,5	1,6	1,3	1,4	1,7	1,8
2,1	2,2	2,5	2,6	2,3	2,4	2,7	2,8
5,1	5,2	5,5	5,6	5,3	5,4	5,7	5,8
6,1	6,2	6,5	6,6	6,3	6,4	6,7	6,8
3,1	3,2	3,5	3,6	3,3	3,4	3,7	3,8
4,1	4,2	4,5	4,6	4,3	4,4	4,7	4,8
7,1	7,2	7,5	7,6	7,3	7,4	7,7	7,8
8,1	8,2	8,5	8,6	8,3	8,4	8,7	8,8

**t=0.8**

1,1	1,2	1,5	1,6	1,3	1,4	1,7	1,8
2,1	2,2	2,5	2,6	2,3	2,4	2,7	2,8
5,1	5,2	5,5	5,6	5,3	5,4	5,7	5,8
6,1	6,2	6,5	6,6	6,3	6,4	6,7	6,8
3,1	3,2	3,5	3,6	3,3	3,4	3,7	3,8
4,1	4,2	4,5	4,6	4,3	4,4	4,7	4,8
7,1	7,2	7,5	7,6	7,3	7,4	7,7	7,8
8,1	8,2	8,5	8,6	8,3	8,4	8,7	8,8

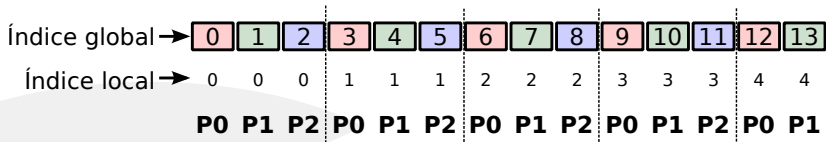
**t=0.4**

1,1	1,2	1,5	1,6	1,3	1,4	1,7	1,8
2,1	2,2	2,5	2,6	2,3	2,4	2,7	2,8
5,1	5,2	5,5	5,6	5,3	5,4	5,7	5,8
6,1	6,2	6,5	6,6	6,3	6,4	6,7	6,8
3,1	3,2	3,5	3,6	3,3	3,4	3,7	3,8
4,1	4,2	4,5	4,6	4,3	4,4	4,7	4,8
7,1	7,2	7,5	7,6	7,3	7,4	7,7	7,8
8,1	8,2	8,5	8,6	8,3	8,4	8,7	8,8

# Distribución de matrices por bloques

## Distribución cíclica por bloques

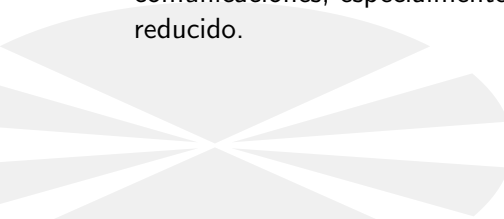
Cuando se toman bloques de un elemento, la distribución se denomina *distribución cíclica*:



# Distribución de matrices por bloques

## Distribución cíclica por bloques

- El objetivo de esta distribución es tratar de balancear la carga de forma equitativa entre los procesos, independientemente de la regularidad de los cálculos sobre la matriz.
- Debe contrastarse esta ganancia frente a la reducción de localidad y el consiguiente incremento en los costes de comunicaciones, especialmente al utilizar bloques de tamaño reducido.



# Distribución de matrices por bloques

## Distribución cíclica por bloques

Siguiendo una distribución cíclica por bloques, una matriz  $s$ -dimensional puede distribuirse sobre una malla  $k$ -dimensional de procesadores usando tamaños de bloque  $m_{b_0}, \dots, m_{b_{k-1}}$ :

- Sea  $m_{b_d}$  el tamaño de bloque en la dimensión  $d$  ( $0 \leq d < k$ ).
- Número de bloques a repartir en la dimensión  $d$  ( $0 \leq d < k$ ):

$$\#b = \left\lceil \frac{n_d}{m_{b_d}} \right\rceil$$

- Número de bloques recibidos el proceso  $p_j$  en la dimensión  $d$  ( $0 \leq d < k$ ):

$$b_d = \begin{cases} \left\lceil \frac{\#b}{p_d} \right\rceil & \text{si } j < (\#b \bmod p_d) \\ \left\lfloor \frac{\#b}{p_d} \right\rfloor & \text{en otro caso} \end{cases}$$



Localización de un elemento:

Índice global  $(i_0, \dots, i_{s-1})$

---

Proceso asignado  $(pa_0, \dots, pa_{k-1})$   
 donde  $pa_x = \left\lfloor \frac{i_x}{m_{b_x}} \right\rfloor \bmod p_x$

---

Índice local  $(l_0, \dots, l_{s-1})$   
 donde  $l_x = \left\lfloor \frac{\left\lfloor \frac{i_x}{m_{b_x}} \right\rfloor}{p_x} \right\rfloor m_{b_x} + i_x \bmod m_{b_x}$   
 si  $0 \leq x < k$   
 en otro caso  $l_x = i_x$

# Distribución de matrices por bloques

## Distribución cíclica por bloques

La distribución cíclica por bloques es la distribución por bloques más general de una matriz  $n \times m$ :

- Si se escogen bloques de un elemento, se obtiene la distribución cíclica.
- Si se escogen bloques de tamaño  $\left(\left\lceil \frac{n}{p} \right\rceil \times m\right)$  y una malla de procesos  $p \times 1$ , obtenemos una distribución por bloques de filas.
- Si se escogen bloques de tamaño  $\left(n \times \left\lceil \frac{m}{p} \right\rceil\right)$  y una malla de procesos  $1 \times p$  se obtiene una distribución por bloques de columnas.
- Por último, dada una malla de procesos  $p_x \times p_y$ , si se escogen bloques de tamaño  $\left(\left\lceil \frac{n}{p_x} \right\rceil \times \left\lceil \frac{m}{p_y} \right\rceil\right)$  se obtiene una distribución por bloques bidimensionales.

# Distribución de matrices por bloques

## Distribución cíclica por bloques

### Compromiso

- A menor tamaño de bloque, mejor balanceo de carga.
- A mayor tamaño de bloque, menos coste de comunicaciones.

Para ello, habrá que considerar las características del algoritmo y de la plataforma paralela usada para su ejecución.

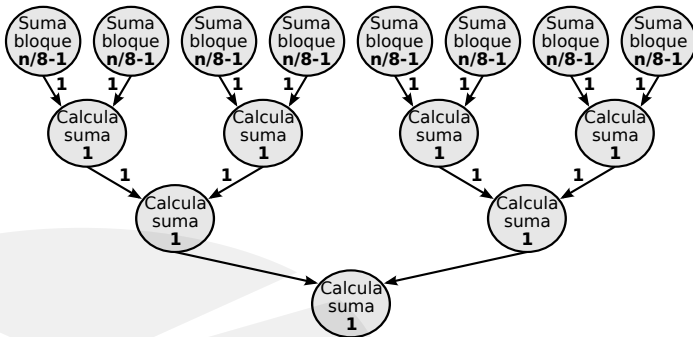
# Asignación de grafos de dependencias estáticos

- En numerosas ocasiones el cálculo a realizar no se define mediante una matriz, por lo que los esquemas de distribución vistos son ineficaces.
- La fase de asignación consistirá en encontrar un reparto de las tareas del grafo de dependencias y un orden de ejecución que minimice el tiempo de ejecución.
- En el caso general, este problema es NP-completo.
- Pero existen situaciones concretas y regulares (y muy comunes) para las que se conocen soluciones óptimas aplicables como patrones.

# Asignación de grafos de dependencias estáticos

## Árbol binario: reducciones

Ejemplo del producto escalar:



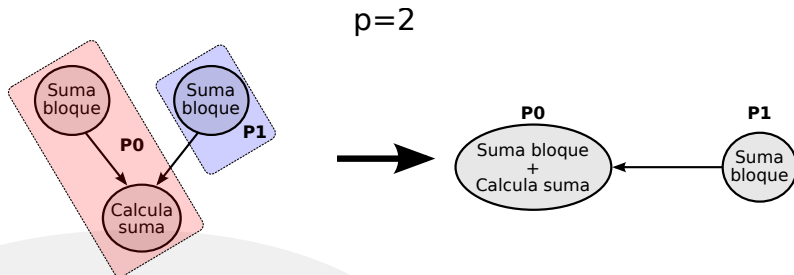
# Asignación de grafos de dependencias estáticos

## Árbol binario: reducciones

- Cada proceso calcula la suma de su bloque.
- La suma acumulada se obtiene siguiendo una estructura recursiva.
- Árbol binario perfecto:  $\log(n)$  niveles, con  $n$  número de tareas hoja en el árbol. Todas las operaciones de reducción tienen un grafo como éste.
- Sólo se pueden ejecutar de forma concurrente las tareas del mismo nivel, por lo que podemos agrupar las de diferentes niveles sin reducir las oportunidades de paralelismo: árbol binomial de orden  $k$ .

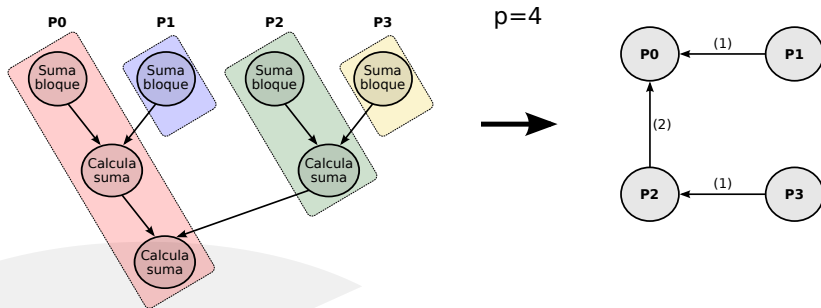
# Asignación de grafos de dependencias estáticos

Árbol binario: reducciones



# Asignación de grafos de dependencias estáticos

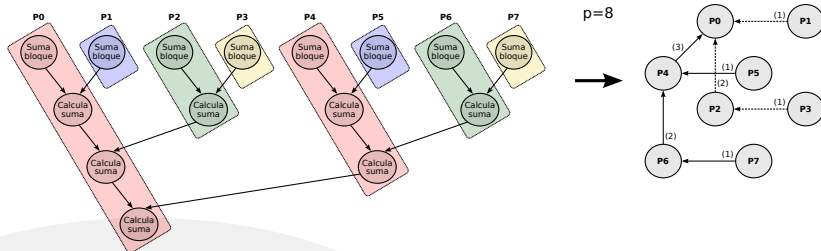
Árbol binario: reducciones





# Asignación de grafos de dependencias estáticos

## Árbol binario: reducciones



# Asignación de grafos de dependencias estáticos

## Árbol binario: reducciones

### Árbol binomial de orden $k$

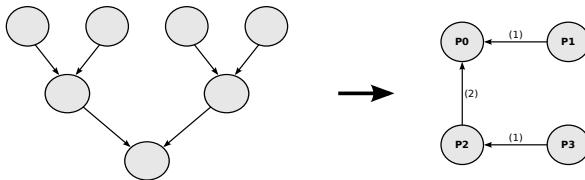
- Se forma uniendo los nodos raíz de dos árboles binomiales de orden  $(k - 1)$ , y haciendo que uno de ellos sea la raíz del nuevo árbol.
  - $2^k$  nodos, profundidad  $k$ .
  - Permite realizar una operación de reducción sobre  $k$  datos en  $\log(k)$  pasos.
- 
- Ciertos procesos dejan de participar en el cálculo tras un determinado número de pasos.
  - Inevitable e impuesto por el grado medio de concurrencia del grafo de dependencias.

# Asignación de grafos de dependencias estáticos

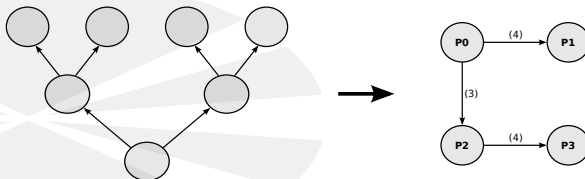
## Árbol binario: reducción con replicación

Supongamos que se desea que el resultado de la operación de reducción sea replicado en todos los nodos:

Fase 1: Combinación



Fase 2: Difusión



# Asignación de grafos de dependencias estáticos

Árbol binario: reducción con replicación

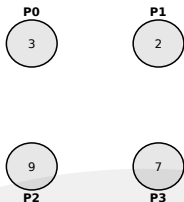
- Usando el enfoque anterior emplearíamos dos árboles binomiales de procesos, llevando en total  $2 \log(k)$  pasos realizar la operación para  $k$  datos de entrada.
- Es posible resolverlo en  $\log(k)$  pasos usando una estructura en hipercubo.



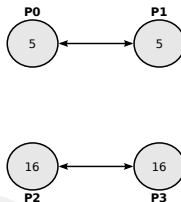
# Asignación de grafos de dependencias estáticos

Árbol binario: reducción con replicación

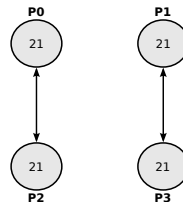
Estado inicial



Paso 1

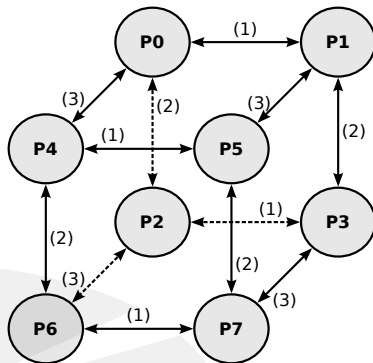


Paso 2



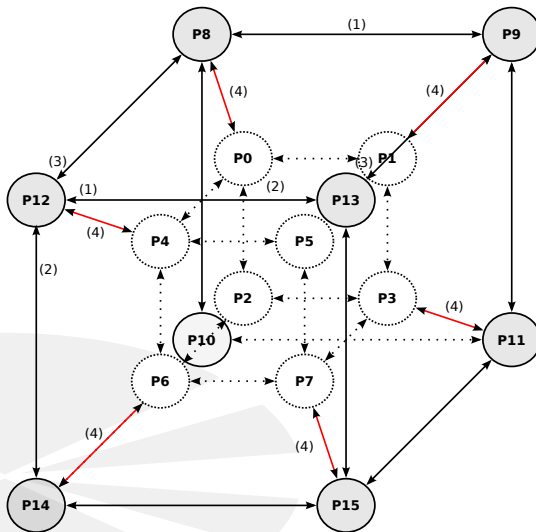
# Asignación de grafos de dependencias estáticos

Árbol binario: reducción con replicación



# Asignación de grafos de dependencias estáticos

Árbol binario: reducción con replicación



# Esquemas de asignación dinámica

- En estos esquemas, el trabajo se reparte durante la ejecución del algoritmo paralelo. Son adecuados cuando:
  - Las tareas se generan dinámicamente (no existe árbol estático de dependencias).
  - El tamaño de las tareas no se conoce a priori (la distribución de la carga no se puede determinar estáticamente).
- Su diseño e implementación es más complejo que el de los esquemas estáticos, e imponen una sobrecarga en tiempo de ejecución. Se necesita un algoritmo de detección de terminación.
- Las tareas obtenidas en la descomposición se convierten en estructuras de datos que representan subproblemas. Éstos se recopilan en una colección desde la que se van asignando a los procesos. En función de cómo se gestione esta colección, los esquemas de asignación pueden ser:
  - Centralizados.
  - Descentralizados.



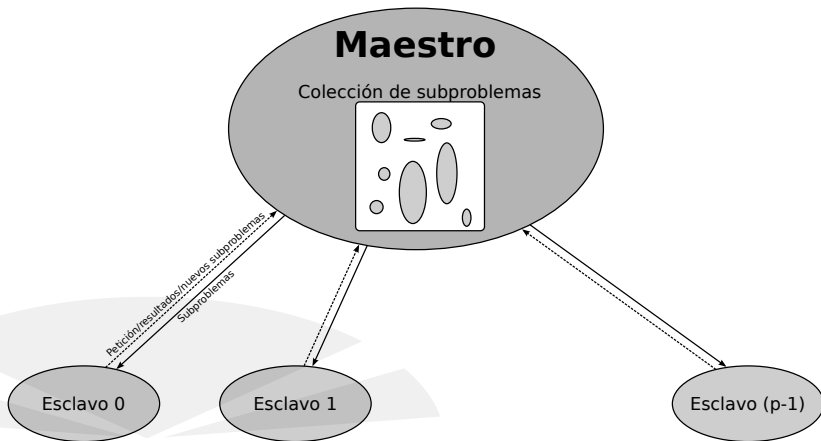
# Esquemas de asignación dinámica

## Esquemas centralizados

- La colección de problemas se mantiene centralizada en una única localización.
- Un proceso *maestro* se encarga de la gestión y reparto de la carga a los procesos de cómputo, denominados *esclavos*.
- El proceso maestro reparte trabajos a cada uno de los  $(p - 1)$  procesos esclavos repetidamente. En respuesta, cada esclavo podrá devolver un resultado y/o generar nuevos subproblemas, que se enviarán al maestro para ser añadidos a la colección centralizada.
- Estrategia efectiva con número moderado de esclavos y cuando el coste de ejecutar subproblemas es alto comparado con el de obtenerlos.
- En caso contrario, las comunicaciones con el proceso maestro se convierten en un cuello de botella.

# Esquemas de asignación dinámica

## Esquemas centralizados



# Esquemas de asignación dinámica

## Esquemas centralizados básicos

- Planificación por bloques:
  - Los procesos esclavos captan las tareas en bloques de subproblemas.
  - Se reduce la contención en las comunicaciones con el maestro.
  - Si los bloques son demasiado grandes se provocarán desequilibrios de carga, especialmente hacia el final de la ejecución.
- Colecciones locales de subproblemas:
  - Cada esclavo almacena localmente para su resolución algunos de los nuevos subproblemas generados.
  - Reduce la contención en las comunicaciones con el maestro.
  - Nuevamente, puede producir desequilibrios de carga si los esclavos almacenan demasiados subproblemas sin enviarlos al maestro.
- Captación anticipada: la recepción de nuevos problemas se solapa con el cálculo de los anteriores para mejorar el grado medio de concurrencia.

# Esquemas de asignación dinámica

## Detección de terminación en esquemas centralizados

- Se determina que el algoritmo ha finalizado cuando se satisfacen las siguientes condiciones:
  - ① La colección de subproblemas centralizada está vacía.
  - ② Todos los procesos esclavos están ociosos y han solicitado nuevos subproblemas al maestro.



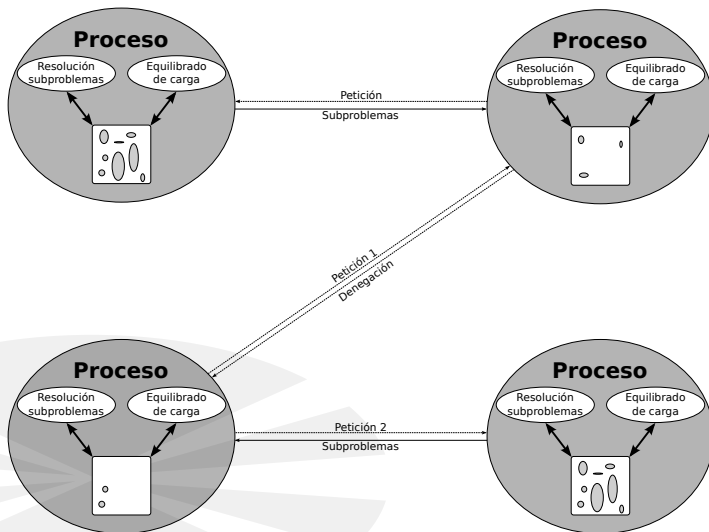
# Esquemas de asignación dinámica

## Esquemas completamente descentralizados

- No existe un maestro central, sino que los subproblemas se encuentran distribuidos entre los repositorios locales de cada uno de los procesos.
- Todos los procesos participan tanto en la resolución de problemas como en el equilibrado de la carga.
- Equilibrado de la carga:
  - Iniciado por el receptor: es el proceso que necesita trabajos el que los solicita al resto.
  - Iniciado por el emisor: un proceso que mantiene un número elevado de subproblemas en su repositorio local decide repartirlos entre otros procesos.
  - Esquemas mixtos: mezclar ambas estrategias dependiendo de la carga global del sistema (si ésta puede ser estimada).

# Esquemas de asignación dinámica

Esquemas completamente descentralizados



# Esquemas de asignación dinámica

## Esquemas completamente descentralizados

Modos de seleccionar el proceso al que se solicita trabajo en un esquema iniciado por el receptor:

- Sondeo aleatorio:
  - Se elige aleatoriamente al proceso receptor de la petición.
  - Equilibra las solicitudes de trabajo recibidas por cada proceso.
- Sondeo cíclico:
  - Cada proceso mantiene una variable  $x$ , inicializada a  $(i + 1)$ , indicando el proceso al que debe solicitar trabajo.
  - En cada petición, la variable  $x$  se incrementa cíclicamente, saltando al proceso emisor.
  - Ventaja: en caso de no obtener trabajo de un determinado proceso, se solicita a uno diferente.
  - Inconveniente: puede provocar múltiples peticiones simultáneas al mismo proceso si existe un alto acoplamiento.
  - Solución: centralizar  $x$ , lo que incrementa los costes de interacción.

# Esquemas de asignación dinámica

## Detección de terminación en esquemas descentralizados sin reactivación

- Un proceso puede estar activo, resolviendo problemas, o en espera, cuando se ha quedado sin trabajo y ningún otro proceso ha podido cederle tareas.
- Cuando el proceso 0 pasa a espera envía un testigo de detección de fin al anterior proceso en el anillo ( $p - 1$ ).
- Cuando un proceso  $j$  recibe el testigo:
  - Si está en espera lo retransmite al proceso ( $j - 1$ ).
  - Si está activo lo retiene hasta pasar a espera.
- Cuando el proceso 0 recibe nuevamente el testigo, sabe que todos han terminado.



# Esquemas de asignación dinámica

## Algoritmo de terminación de Dijkstra

- Si se permite la reactivación, el algoritmo anterior puede fallar si el proceso  $i$  reenvía el testigo tras quedarse en espera pero es más tarde reactivado por recibir tareas de otro proceso  $j$  para  $j > i$ .
- Se incluye más información en el esquema: cada proceso tendrá un color (blanco o negro), al igual que el testigo.
- Inicialmente el testigo y todos los procesos son blancos.
- Cuando un proceso  $i$  envía un mensaje de trabajo a otro  $j$ , siendo  $j > i$ , cambia su color a negro.
- Un proceso en espera con color negro reenviará el testigo con color negro y, tras ello, cambiará su color a blanco.
- Si el proceso 0, siendo blanco, recibe un testigo negro, cambia su color a blanco.
- El proceso termina cuando el proceso 0, siendo de color blanco, recibe un testigo blanco.

# Esquemas de asignación dinámica

Algoritmo de terminación de Dijkstra

