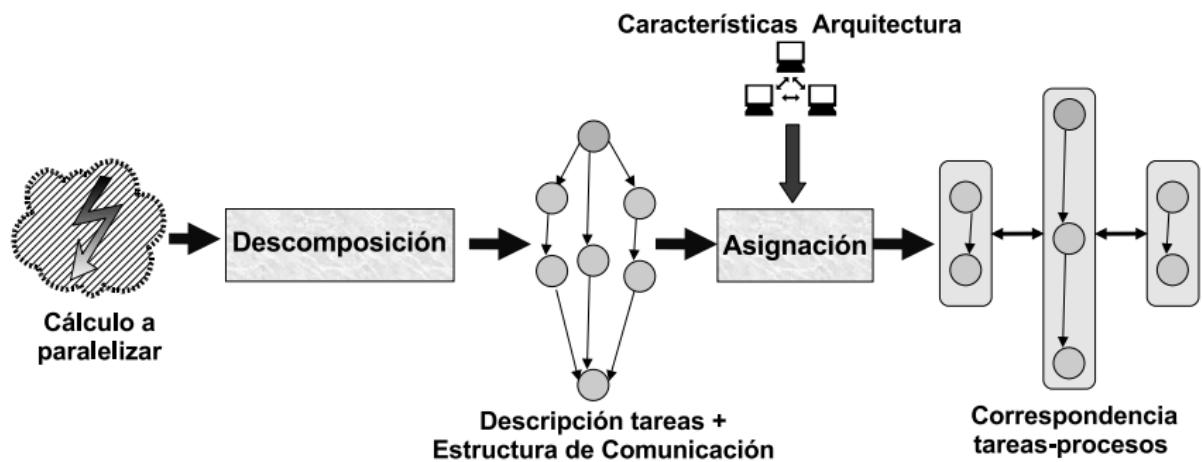


Tema 2. Metodología de diseño de algoritmos paralelos



CONTENIDOS

1 Descomposición en tareas

- Conceptos previos
- Técnicas de descomposición

2 Asignación de tareas

- El problema de la asignación. Objetivos
- Estrategias generales
- Esquemas de asignación estática
- Esquemas de equilibrado dinámico de la carga

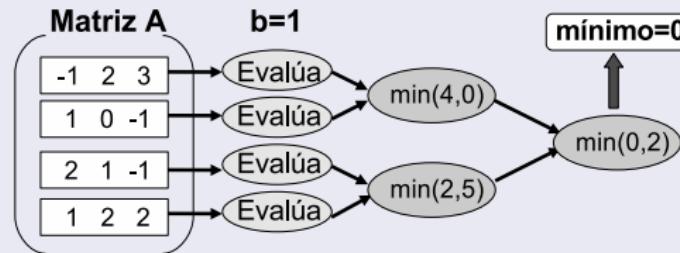
DESCOMPOSICIÓN EN TAREAS

Ejemplo: Evaluación polinomial

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

Se desea evaluar obtener un $v = \min_{i=0}^{m-1} \{f^i(b)\}$.

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



Granularidad de una descomposición

Tipos de granularidad

- **Granularidad fina:** la descomposición genera un número elevado de pequeñas tareas.
- **Granularidad gruesa:** se obtienen pocas tareas de gran tamaño.

Ejemplos

- Evaluación de 2000 polinomios
 - 500 polinomios por tarea de evaluación: grano grueso (7 tareas).
 - 1 polinomio por tarea: grano fino (sobre 4000 tareas):
- Suma de vectores. $W = U + V, \quad U, V \in \mathbb{R}^n$
 - Mínima granularidad aceptable para calcular : Una tarea separada calcula cada w_i

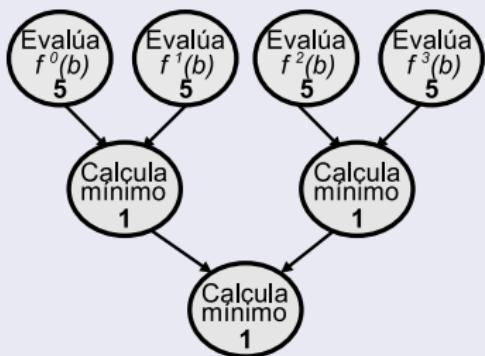
Grafo de dependencias

Grafo de dependencias

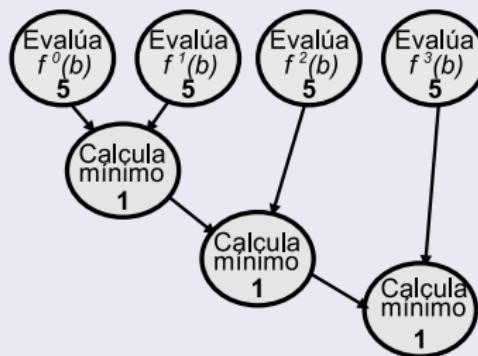
- Grafo dirigido acíclico donde los nodos representan tareas y una arista conectando una tarea fuente con otra destino representa que para poder ejecutarse la tarea destino debe ejecutarse previamente la tarea fuente.
 - Cada nodo del grafo suele etiquetarse con un valor proporcional al coste computacional de la tarea.
 - Dependiendo de la estrategia de resolución, se pueden obtener diferentes grafos.



Dos grafos de dependencias diferentes que resuelven el mismo problema



a)



b)

Grafo de dependencias y grado de concurrencia

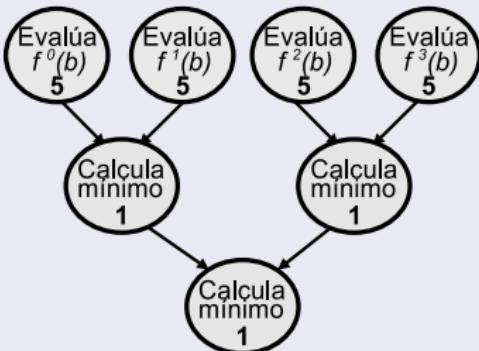
Grado de concurrencia

- **Máximo grado de concurrencia:** Mayor número de tareas cuya ejecución se podría realizar al mismo tiempo en el grafo de dependencias.
 - **Camino crítico** Camino más largo en el grafo desde un nodo de comienzo hasta un nodo de finalización. La longitud L se obtiene sumando los costes de los nodos componentes.
 - **Grado medio de concurrencia:** Número medio de tareas que se podrían ejecutar en paralelo, considerando todas las fases del algoritmo. Para un grafo con N nodos:

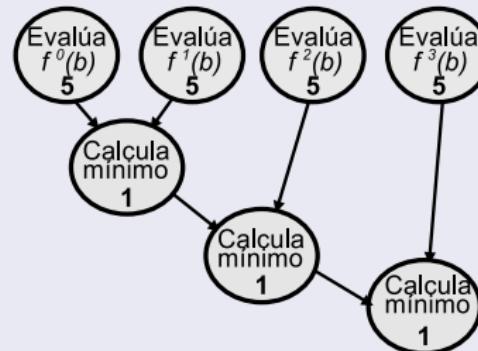
$$M = \frac{\sum_{i=1}^N \text{coste}(nodo_i)}{L}$$

Grafo de dependencias y grado de concurrencia

Grado medio de concurrencia para dos grafos de dependencias



a)



b)

$$M(\text{grafo}(a)) = \frac{23}{7} = 3.28$$

$$M(\text{grafo}(b)) = \frac{23}{8} = 2.875$$

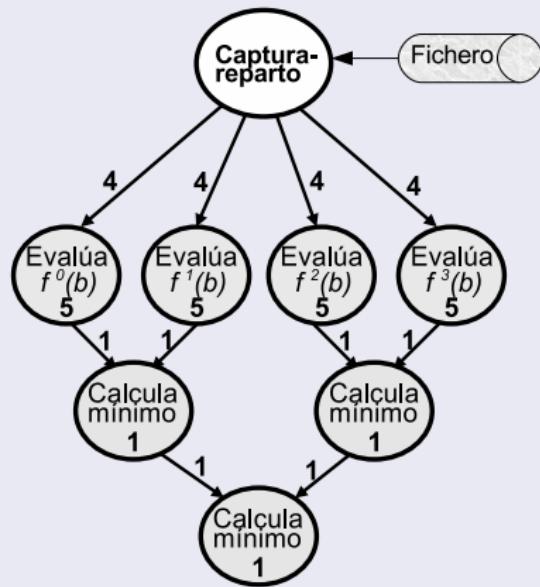
Comunicación y sincronización. Estructura de comunicación

Estructura de comunicación

- Existen relaciones de sincronización y comunicación entre las tareas que no tienen por qué aparecer en el grafo.
- **Estructura de comunicación:** los nodos representan tareas, mientras las aristas conectan tareas entre las que existe una relación de comunicación o sincronización.
- Los nodos se etiquetan con valor proporcional a la carga computacional y las aristas con valores proporcionales a la cantidad de datos que se comunican.
- En muchos casos las aristas son dirigidas para indicar el sentido del flujo de datos.

Comunicación y sincronización. Estructura de comunicación

Estructura de comunicación con paso de mensajes



Fase de descomposición: Visión global

Resultado de la fase de descomposición

- Obtiene un conjunto de tareas, junto con una descripción de su estructura de comunicación que exhibe un grado de concurrencia alto.
- Describe un algoritmo paralelo ideado para una máquina con un número ingente de procesadores capaces de interactuar entre sí sin apenas sobrecarga.
- Esta solución algorítmica será evaluada y restringida a una arquitectura particular en la fase de asignación.

Técnicas de Descomposición

Métodos para descomponer la resolución de un problema

- **Generales**

- Descomposición de dominio.
- Descomposición funcional.
- Descomposición recursiva.

- **Específicos**

- Descomposición exploratoria.

- **Mixtos:** Combinación de los anteriores.

Descomposición de Dominio

Se utiliza cuando es posible resolver un problema aplicando la misma operación sobre partes diferentes de su dominio de datos.

Fases

- a) Se trocean los datos de forma homogénea para obtener particiones del dominio original.
- b) Se estudia cómo asociar computación a cada subdominio de datos.
- c) Se asocia la gestión de cada partición a una tarea que contendrá los datos y un conjunto de operaciones sobre dichos datos.

Descomposición de Dominio

Datos a descomponer

- a) **Datos de salida:** cada componente de los datos de salida se puede calcular de forma independiente del resto.
- b) **Datos de entrada**
- c) **Datos intermedios:** es posible a veces obtener un mayor grado de concurrencia centrándose en datos intermedios entre etapas.

Consejos

- Centrarse primero en la estructura de datos más grande o en la más usada, analizando distintas posibilidades.
- **Regla del propietario:** Casos a) y b).
La tarea propietaria se encarga de los cálculos ligados a su subdominio.

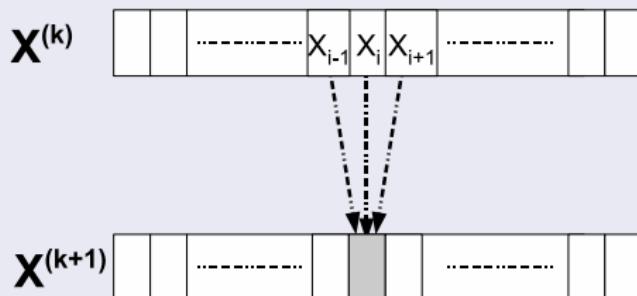


Descomposición centrada en los datos de salida

Transformación iterativa de un vector

$$x_i^{(k+1)} = \frac{x_{i-1}^{(k)} - x_i^{(k)} + x_{i+1}^{(k)}}{2}, \quad i = 0, \dots, n-1 ,$$

$$x_{-1}^{(k)} = x_{n-1}^{(k)}, \quad x_n^{(k)} = x_0^{(k)}.$$



Descomposición centrada en los datos de salida

Asignamos porciones disjuntas de elementos consecutivos del vector de salida a cada tarea

Tarea i(Bloque)

Para $k = 0$ **hasta** *numero_iteraciones* **hacer**

Envia(*Bloque*[0], $(i - 1) \bmod p$);

Envia(*Bloque*[$n/p - 1$], $(i + 1) \bmod p$);

Recibe(*izquierda*, $(i - 1) \bmod p$);

Recibe(*derecha*, $(i + 1) \bmod p$);

Para $j = 0$ **hasta** $n/p - 2$ **hacer**

tmp=*Bloque*[*j*];

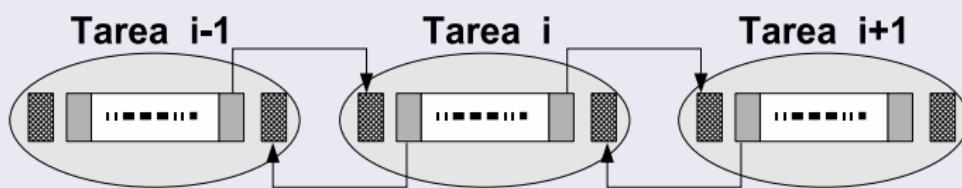
Bloque[*j*]=(*izquierda* − *Bloque*[*j*] + *Bloque*[*j* + 1])/2;

izquierda=*tmp*;

Bloque[$n/p - 1$]=(*izquierda* − *Bloque*[$n/p - 1$] + *derecha*)/2;

Descomposición centrada en los datos de salida

Patrón de comunicación en cada iteración



Descomposición centrada en los datos de entrada

Producto escalar de dos vectores

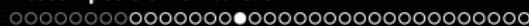
$$X = (x_0, x_1, \dots, x_{n-1}), \quad Y = (y_0, y_1, \dots, y_{n-1}),$$

$$Z = X \cdot Y = x_0y_0 + x_1y_1 + \dots + x_{n-1}y_{n-1}$$

- Descomposición razonable: asignar aproximadamente el mismo número de elementos de X y de Y a cada tarea, alineando.
- Tarea i ($i = 0, 1, \dots, p - 1$) gestiona bloque de X e Y

$$Z_i = \sum_{j=in/p}^{(i+1)n/p} x_j y_j.$$

- Suma de valores locales: descomposición recursiva.



Descomposición de Algoritmos matriciales por bloques

Multiplicación matriz-vector por bloques

- $A = (a_{i,j}) \in \mathbb{R}^{n \times n}$, $x = [\hat{x}_0, \dots, \hat{x}_{n-1}]^T \in \mathbb{R}^{n \times 1}$.

$$y = [\hat{y}_0, \dots, \hat{y}_{n-1}]^T = Ax, \quad \hat{y}_i = \sum_{j=0}^{n-1} a_{i,j} \hat{x}_j.$$

- **Formulación por bloques:**

Sea m tal que n es divisible entre m .

$A = (A_{ij})$ matriz $m \times m$ de submatrices $\frac{n}{m} \times \frac{n}{m}$.

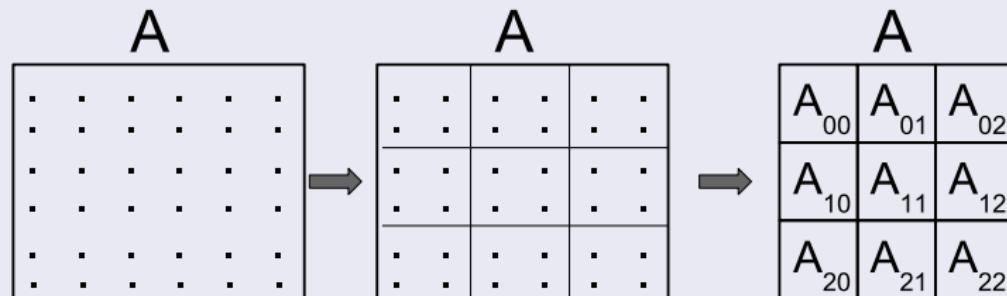
$x = [x_0, \dots, x_{m-1}]^T$ donde cada x_i es un subvector $\frac{n}{m} \times 1$.

$$y_i = \sum_{j=0}^{m-1} A_{i,j} x_j.$$



Descomposición de Algoritmos matriciales por bloques

Descomposición de la multiplicación matriz-vector por bloques



$$\begin{array}{c}
 \text{A} \quad \text{b} \quad \text{X} \\
 \begin{array}{|c|c|c|} \hline
 A_{00} & A_{01} & A_{02} \\ \hline
 A_{10} & A_{11} & A_{12} \\ \hline
 A_{20} & A_{21} & A_{22} \\ \hline
 \end{array} \times \begin{array}{|c|} \hline
 b_0 \\ \hline
 b_1 \\ \hline
 b_2 \\ \hline
 \end{array} = \begin{array}{|c|} \hline
 x_0 \\ \hline
 x_1 \\ \hline
 x_2 \\ \hline
 \end{array}
 \end{array}$$

Decomposition equations:

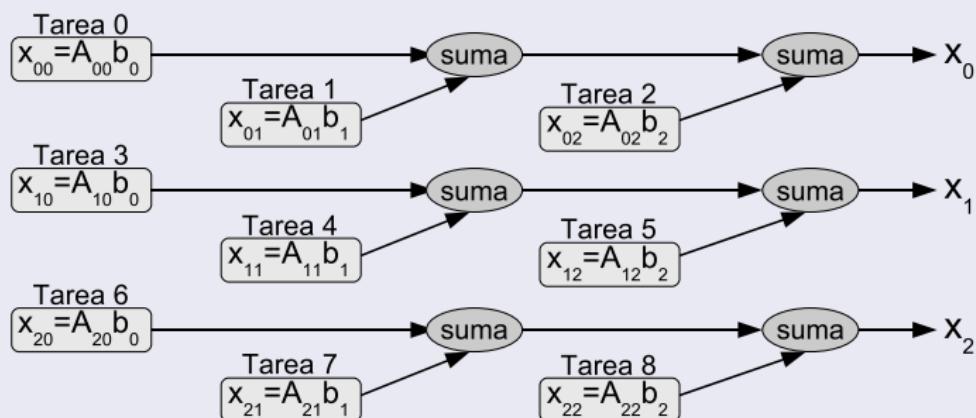
$$\begin{aligned}
 x_0 &= A_{00} b_0 + A_{01} b_1 + A_{02} b_2 && \text{Tarea 0} \\
 x_1 &= A_{10} b_0 + A_{11} b_1 + A_{12} b_2 && \text{Tarea 1} \\
 x_2 &= A_{20} b_0 + A_{21} b_1 + A_{22} b_2 && \text{Tarea 2}
 \end{aligned}$$

Descomposición centrada en los datos intermedios

Descomposición de la multiplicación matriz-vector

$$\begin{array}{c}
 \textbf{A} \quad \textbf{b} \quad \textbf{x} \\
 \begin{array}{|c|c|c|} \hline A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \\ \hline \end{array} \times \begin{array}{|c|} \hline b_0 \\ \hline b_1 \\ \hline b_2 \\ \hline \end{array} = \begin{array}{|c|} \hline x_0 \\ \hline x_1 \\ \hline x_2 \\ \hline \end{array}
 \end{array}$$

$x_1 = A_{10}b_0 + A_{11}b_1 + A_{12}b_2 = x_{10} + x_{11} + x_{12}$
 $x_2 = A_{20}b_0 + A_{21}b_1 + A_{22}b_2 = x_{20} + x_{21} + x_{22}$



Descomposición funcional dirigida por el flujo de datos

La descomposición se ajusta a la propia arquitectura de la aplicación a paralelizar.

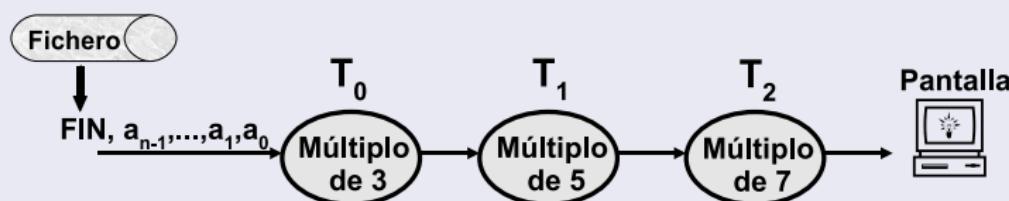
Pasos

- a) Se identifican las partes funcionales del cálculo.
- b) Se asigna una tarea para la realización de cada fase identificada.
- c) Se examinan los requisitos de datos de dichas tareas.
 - c1 Superposición de datos mínima y volumen de flujo de datos pequeño → **FIN**.
Resultado: Grafo de tareas con cierta topología.
 - c2 Excesivo solapamiento de datos → demasiada comunicación.
Consejo: analizar aplicación otra técnica de descomposición..

Descomposición funcional dirigida por el flujo de datos

Cauce paralelo para filtrar una lista de enteros

- Dada una serie de m primos p_0, p_1, \dots, p_{m-1} y una lista de n enteros, $a_0, a_1, a_2, \dots, a_{n-1}$, encontrar aquellos números de la lista que son múltiplos de todos los m primos ($n \gg m$)
- **Descomposición:** La tarea $T_i, i = 0, \dots, m - 1$ mantiene el primo p_i y chequea multiplicidad con p_i .



Descomposición funcional dirigida por el flujo de datos

Tarea $i(p_i)$

Repite

Si $i = 0$ **entonces**

// Obtiene dato de fichero de entrada

Lee(izquierda);

sino

Recibe(izquierda, $i - 1$);

$FIN = izquierda < 0$;

Si $izquierda \bmod p_i = 0$ o FIN **entonces**

Si $i \neq m - 1$ **entonces**

Envía(izquierda, $i + 1$);

sino

// Envía resultado a la salida

Imprime(izquierda);

hasta FIN ;

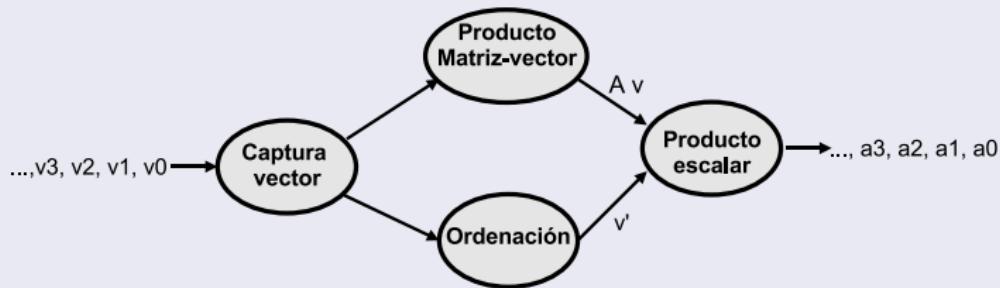


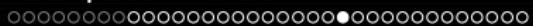
Descomposición funcional dirigida por el flujo de datos

Descomposición funcional de un cálculo sobre vectores

Transformar vectores $v \in \mathbb{R}^n$ en $a = (Av) \cdot v \in \mathbb{R}$.

$A \in \mathbb{R}^{n \times n}$, " \cdot " representa el producto escalar y v = ordenación ascendente de v .





Descomposición recursiva

Técnica secuencial *divide-y-vencerás* para descubrir concurrencia:
Generación recursiva de subproblemas y combinación resultados parciales.

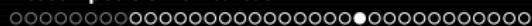
Principal fuente de paralelismo

La resolución de diferentes subproblemas es completamente independiente → se pueden resolver concurrentemente por tareas diferentes sin interacción.

Cada tarea divide el problema o combina resultados de subproblemas.

Esquema de planificación basado en granja de tareas.

Las tareas captan subproblemas de una estructura compartida. La estructura de datos compartida se puede implementar de diferentes formas dependiendo del modelo de programación.



Descomposición recursiva

Integración numérica por cuadratura adaptativa

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

- **Métodos de cuadratura**

Dividisión en subintervalos + aproximación trapezoidal.

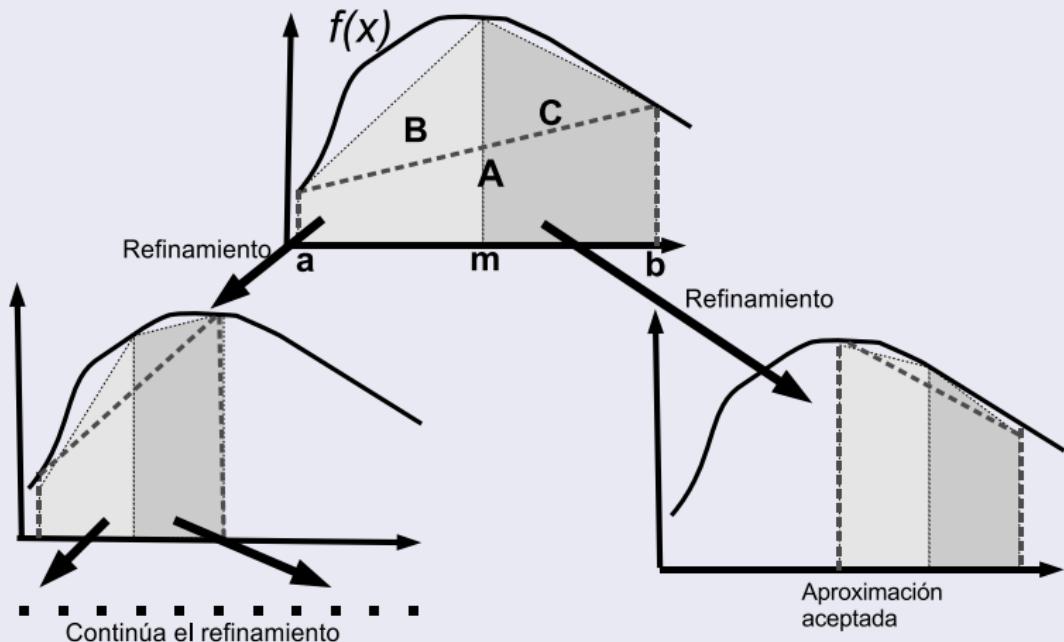
Subintervalo $[p, q]$ con $\delta = q - p$: $A_{p,q} = \frac{(f(p)+f(q))\delta}{2}$

- **Cuadratura adaptativa** : Número variable de intervalos.

- Dividir $([a,b])$ en dos mitades con áreas trapezoidales B y C .
- Si $A - (B + C) < \text{Tolerancia}$, Aproximación= $B + C$.
- Sino, aplica proc. recursivamente a cada mitad.

Descomposición recursiva

Cuadratura adaptativa



Descomposición recursiva

Algoritmo paralelo de cuadratura adaptativa

Entrada: Extremos del subintervalo $[a, b]$ y función a integrar f .

Salida: $AREA =$ aproximación a $\int_a^b f(x)dx$.

inicio

$fa = f(a);$

$fb = f(b);$

$aprox_area = (fa + fb) * (b - a) / 2;$

Crea_tarea:ComputaIntervalo(a, b, fa, fb, approx_area, f);

fin

Descomposición recursiva

ComputaIntervalo(a, b, fa, fb, aprox:Real, f: Realfuncion(Real))

Entrada: Extremos $[a, b]$, $fa = f(a)$ y $fb = f(b)$, aproximación trapezoidal $aprox$ y f .

Salida: Suma al valor de $AREA$ la aproximación obtenida.

$$m = (a + b)/2;$$

$$fm = f(m);$$

$$areaizq = (fa + fm) * (m - a)/2;$$

$$areader = (fm + fb) * (b - m)/2;$$

$$aprox2 = areaizq + areader;$$

Si $|aprox - aprox2| < \epsilon$ **entonces**

$$AREA = AREA + aprox;$$

sino

Crea_tarea:ComputaIntervalo(a, m, fa, fm, areaizq, f);

Crea_tarea:ComputaIntervalo(m, b, fm, fb, areader, f);



Descomposición recursiva

Descomposición recursiva para cálculo del mínimo

Procedimiento recursivo para calcular el mínimo (operación MIN) de una secuencia de reales de longitud n ($a_1, a_2, \dots, a_n \in \mathbb{R}$):

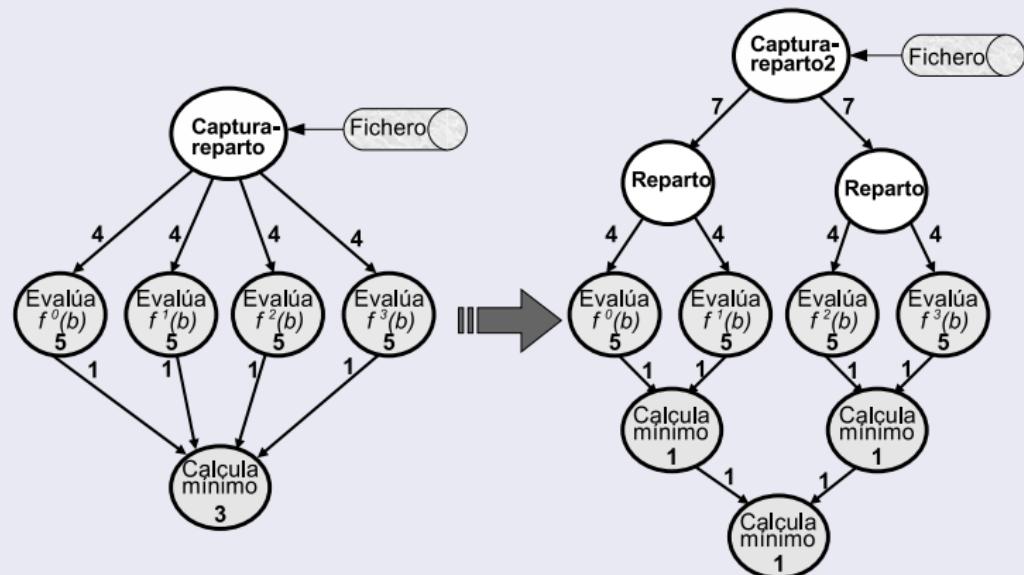
$$\text{MIN}(a_1) = a_1.$$

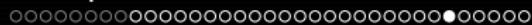
$$\text{MIN}(a_1, a_2) = \min(a_1, a_2).$$

$$\begin{aligned} \text{MIN}(a_1, a_2 \dots a_n; n > 2) &= \min[\text{MIN}(a_1, \dots, a_{n/2}), \\ &\quad \text{MIN}(a_{n/2+1}, \dots, a_n)] \end{aligned}$$

Descomposición recursiva

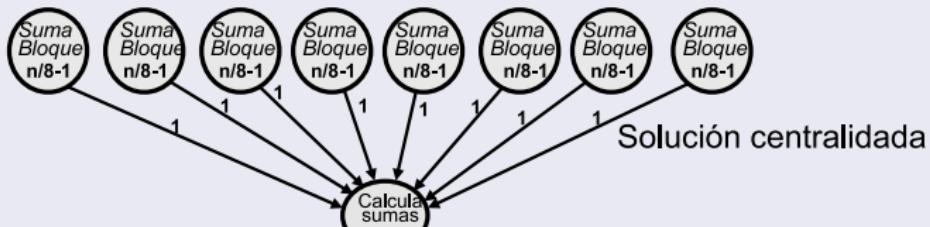
Descomposición recursiva de una estructura centralizada



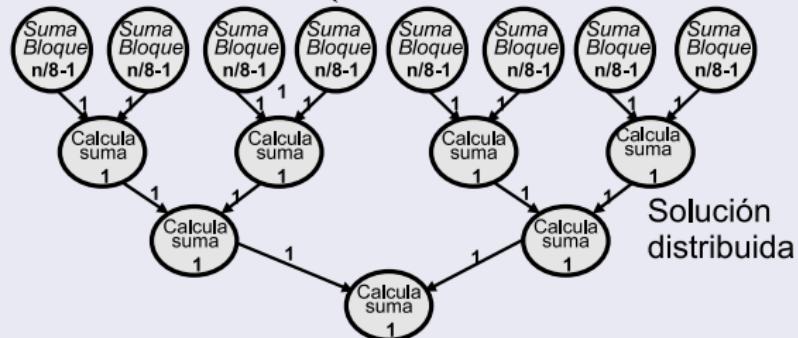


Descomposición recursiva. Patrones Locales/Globales

Descomposición de un patrón global



Descomposición recursiva



Descomposición exploratoria

Ámbito e ideas generales

- **Ámbito:** Búsqueda de soluciones en un espacio de estados.
- **Idea:** Descomposición dinámica del espacio de búsqueda en partes menores + exploración concurrente de cada parte por tarea diferente. Interesa establecer estructura de árbol de posibles soluciones.
- **Estrategia de descomposición común:** Generar niveles de estados desde estado inicial. Cuando se tengan suficientes nodos cada tarea explora un conjunto de nodos diferente.
- **Casos**
 - a) Búsqueda exhaustiva: finaliza cuando no existan nodos.
 - b) Búsqueda primera solución: Tarea que encuentra solución informa resto de final.

Descomposición exploratoria

Problema de la suma del subconjunto

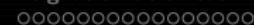
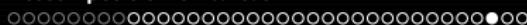
Dado un conjunto de n enteros, $C = \{a_0, a_1, \dots, a_{n-1}\}$, y un entero v , se desea encontrar $\hat{C} \subset C$ tal que:

$$\sum_{a_i \in \hat{C}} a_i = v$$

Representación de un subconjunto $\hat{C} \subset C$

- Se indexan los valores enteros de C . Cada a_i queda identificado únicamente con un i ($i = 0, \dots, n - 1$).
- Un subconjunto \hat{C} de C queda representado de forma única mediante una secuencia de n dígitos.
 - Si la posición i -ésima de la secuencia ($i = 0, \dots, n - 1$) es 1, $a_i \in \hat{C}$.
 - Si la posición i -ésima es 0, a_i no pertenece a \hat{C} .





Descomposición exploratoria. Problema de la suma del subconjunto

Estructuración del espacio. Árbol binario

- *Nodo hoja*: representa un subconjunto diferente de $C =$ secuencia binaria particular de n cifras.
- *Nodo interno*: representa a todos los subconjuntos alcanzables a partir de sus descendientes. La representación es similar a un nodo hoja pero usando símbolo comodín (X).

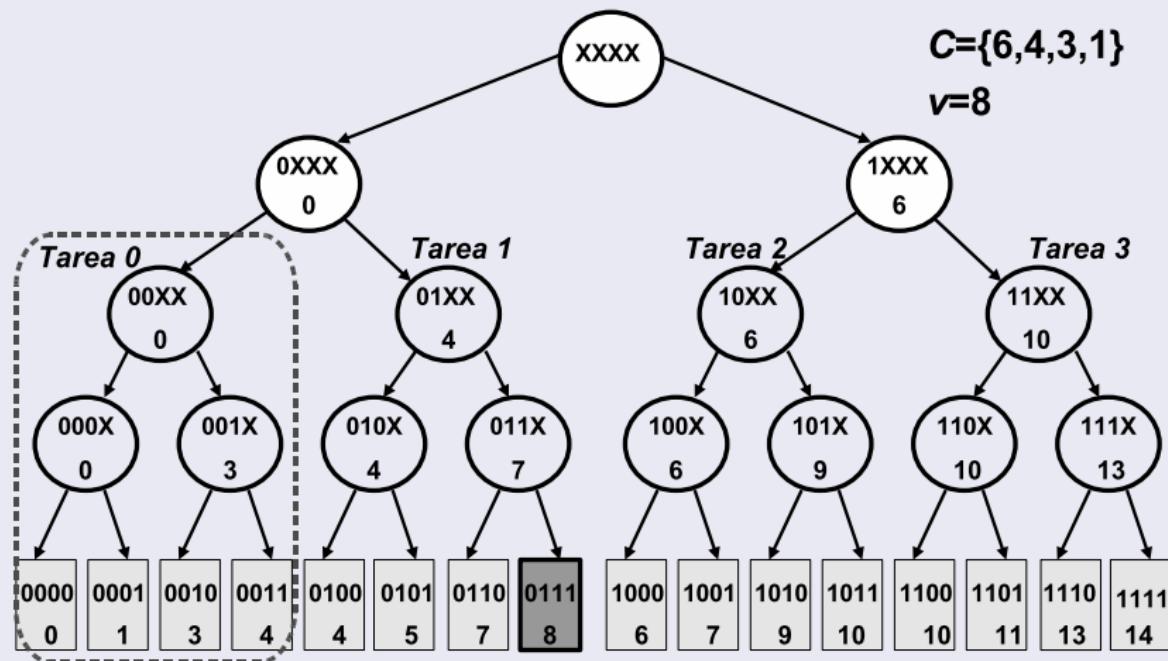
Exploración de un nodo interno

Se determina el símbolo comodín X situado más a la izquierda en la etiqueta.

- Hijo izquierdo se crea fijando dicha posición a 0 .
- Hijo derecho se obtiene fijando dicha posición a 1.

Descomposición exploratoria. Problema de la suma del subconjunto

Descomposición de una instancia del problema



Enfoques mixtos

Diferentes esquemas de descomposición para diferentes fases de procesamiento en una aplicación.

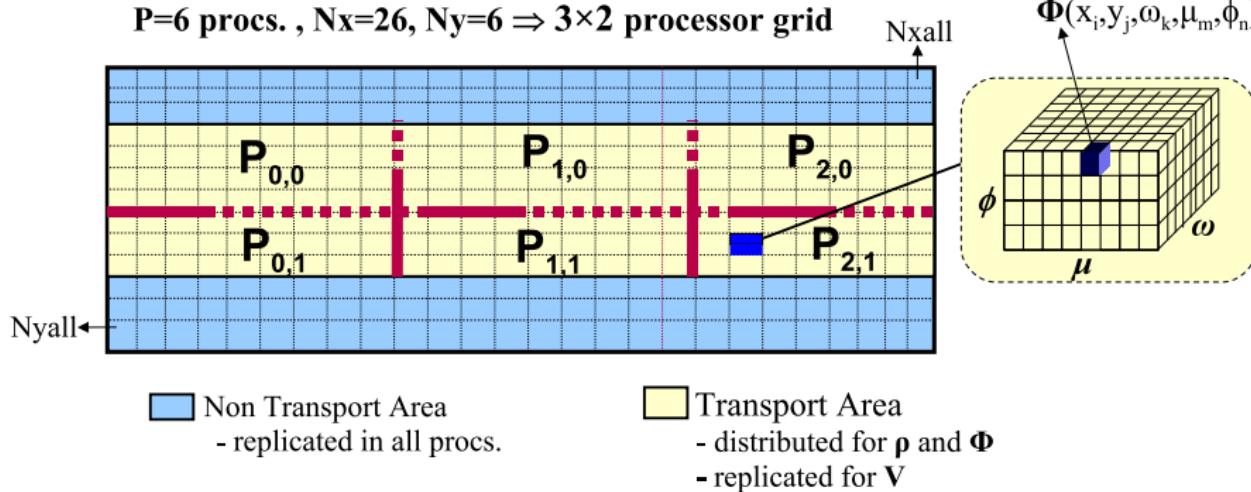
Ejemplos

- **Evaluación polinomial mínima:** Evaluación (dominio) y cálculo del mínimo (recursiva).
- **Producto escalar:** Evaluación sumas parciales (dominio) y suma de términos locales (recursiva).



ASIGNACIÓN DE TAREAS

$P=6$ procs., $Nx=26$, $Ny=6 \Rightarrow 3 \times 2$ processor grid



El problema de la asignación. Objetivos

Problema

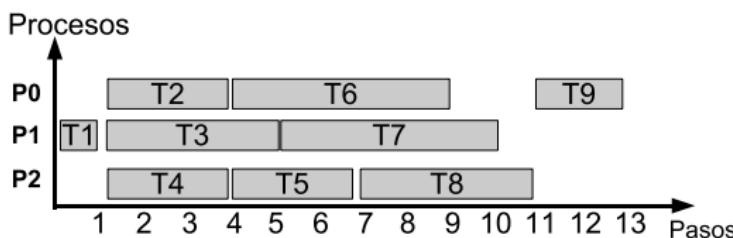
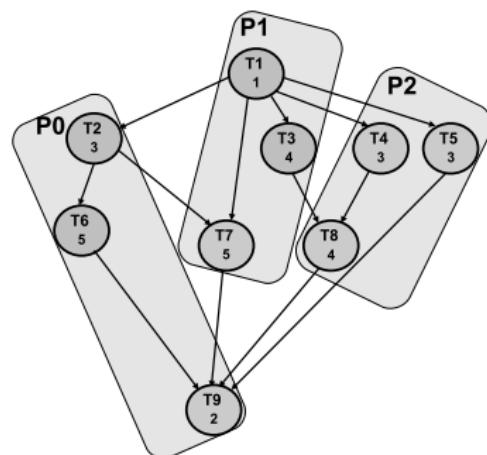
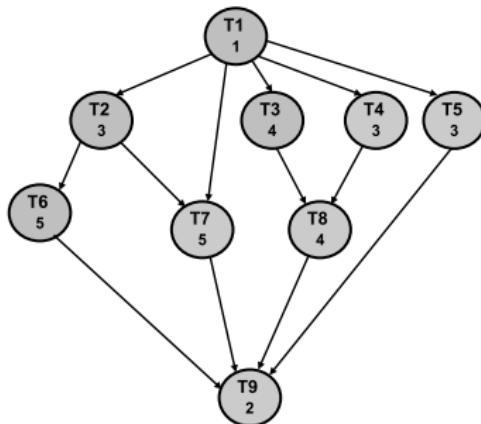
- Correspondencia tareas-procesos.
- Agrupamiento previo de algunas tareas en una única tarea.
- Selección de un orden de ejecución para las tareas.

Objetivo: Minimizar tiempo de ejecución total.

Estrategias

- **Tiempo de computación:** maximizar la concurrencia, asignando tareas independientes a diferentes procesos.
- **Tiempo de comunicación:** asignando tareas que interactúen mucho al mismo proceso.
- **Tiempo de ocio:** evitar fuentes de ociosidad:

El problema de la asignación. Objetivos



El problema de la asignación. Objetivos

Problema

- Correspondencia tareas-procesos.
- Agrupamiento previo de algunas tareas en una única tarea.
- Selección de un orden de ejecución para las tareas.

Objetivo: Minimizar tiempo de ejecución total.

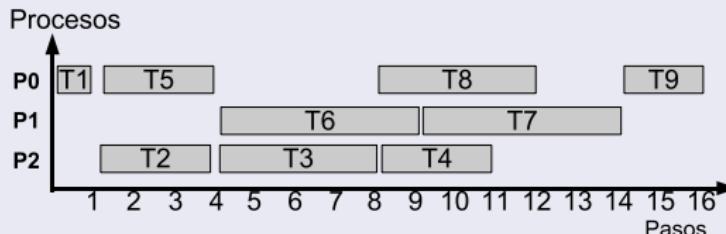
Estrategias (conflictivas entre sí)

- **Tiempo de computación:** maximizar la concurrencia, asignando tareas independientes a diferentes procesos.
- **Tiempo de comunicación:** asignando tareas que interactúen mucho al mismo proceso.
- **Tiempo de ocio:** evitar fuentes de ociosidad:

El problema de la asignación. Objetivos

Fuentes de ociosidad

- a) **Desequilibrios de carga:** Se busca equilibrar tanto los cálculos como las comunicaciones.
- b) **Espera entre procesos debida a dependencias entre tareas:** Asignar una carga de trabajo similar a cada proceso no es suficiente.
Es el grafo de dependencias el que determina qué tareas se pueden ejecutar en paralelo.



Esquemas de asignación

Tipos de Esquemas

- **Estáticos** : Asignación tarea-proceso y orden de ejecución conocidos antes de ejecución.
- **Dinámicos**: Reparto del trabajo computacional entre los procesos en ejecución.

Asignación estática

- ① Estimación número de tareas, tiempo de ejecución y costes de comunicación.
- ② Agrupación tareas para reducir costes.
- ③ Asociar tareas con procesos.

Problema *NP-completo* aunque existen estrategias especializadas y heurísticas.

Esquemas de asignación

Asignación dinámica

Cuándo se utilizan estas técnicas

- Las tareas se generan dinámicamente.
- Tamaño de las tareas no se conoce a priori.

Problemas a resolver

- ¿Qué procesos recopilan la información para tomar decisiones de redistribución?
- ¿Qué condiciones activan la transferencia de carga?
- ¿Qué procesos deben recibir carga?

Desventaja: Añaden sobrecargas en tiempo de ejecución.

Principal ventaja: no se necesita conocer comportamiento antes de la ejecución.

Estrategias generales

a) Agrupamiento

Agrupar un conjunto de tareas en una única para mejorar la localidad

- Minimización del volumen de datos.
- Reducción de la frecuencia de interacciones.

b) Replicación

- Replicación de datos.
- Replicación de cómputo y comunicación

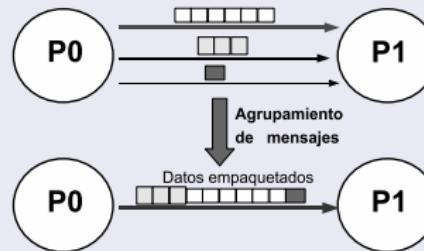
Estrategias generales basadas en Agrupamiento

Minimización del volumen de datos

- Agrupar datos por bloques de elementos contiguos.
- Agrupar tareas que se comuniquen mucho.
- Usar datos locales para almacenar resultados intermedios.

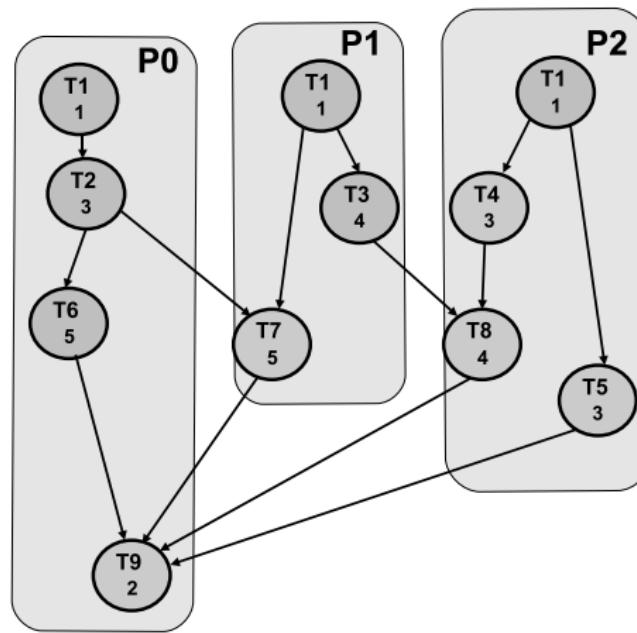
Reducción de la frecuencia de interacciones

- Mem. compartida: Alg. por bloques reduce pérdidas de caché.
- Mem. distribuida: Agrupar mensajes.



Estrategias generales basadas en Replicación

- Replicación de datos en paso de mensajes
- Replicación de cómputo y comunicación



Esquemas de asignación estática

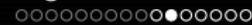
a) Esquemas estáticos en conjunción con descomposición de dominio

Asignación consiste en repartir los datos entre los procesos para minimizar el tiempo de ejecución.

- Distribución por bloques para matrices.
- Esquemas de división estática de grafos.

b) Esquemas para grafos de dependencias estáticas

En conjunción con descomposición funcional del flujo de datos o descomposición recursiva.

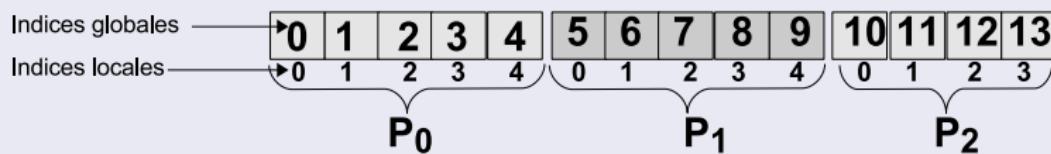


Distribución unidimensional por bloques de una matriz unidimensional

Esquema de asignación

índice global	Bloque	índice proceso	índice local
i	$m_b = \lceil n/p \rceil$	$\lfloor i/p \rfloor$	$i \bmod m_b$

Distribución de una matriz unidimensional entre tres procesos

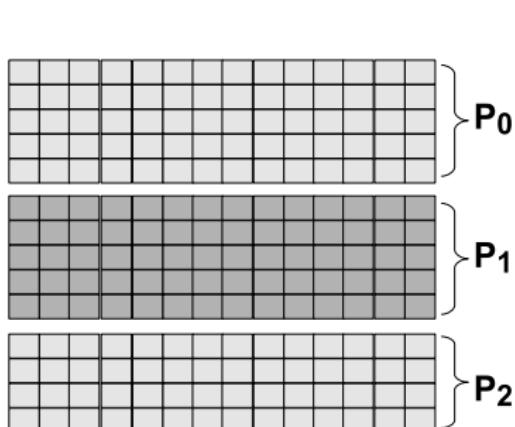




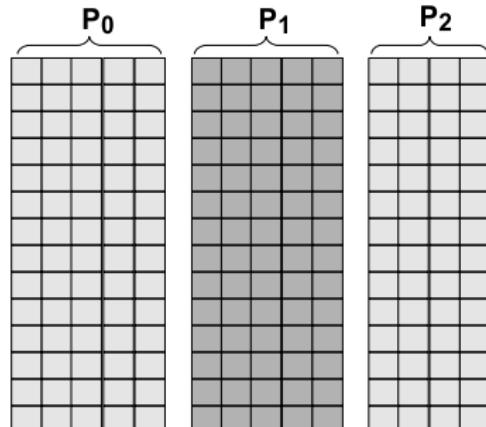
Distribución unidimensional por bloques de una matriz bidimensional

Esquema de distribución por bloques de filas

índices globales	Tamaño Bloque	proceso	índices locales
(i, j)	$m_b = \lceil n/p \rceil$ filas	$\lfloor i/p \rfloor$	$(i \bmod m_b, j)$



a) Por bloques de filas



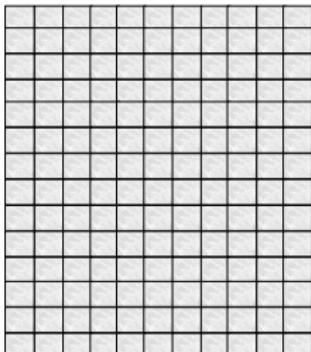
b) Por bloques de columnas

Distribución bidimensional por bloques de una matriz bidimensional

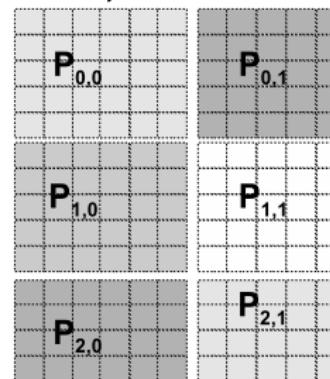
Esquema de distribución

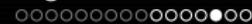
índ. global	Tam. Bloque	proceso	índ. local
(i,j)	$m_{bx} \times m_{by}$ celdas $m_{bx} = \lceil n/p_x \rceil$ $m_{by} = \lceil m/p_y \rceil$	$(\lfloor i/p_x \rfloor, \lfloor j/p_y \rfloor)$	$(i \bmod m_{bx},$ $j \bmod m_{by})$

A[14,11]



P=6, malla 3 × 2



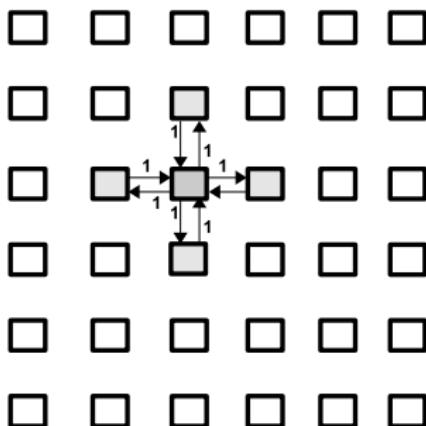


Cálculo de diferencias finitas

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

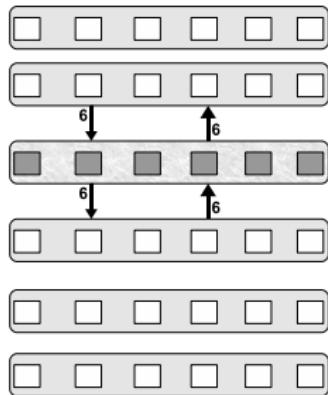
$i = 0, \dots, n-1, j = 0, \dots, n-1,$

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

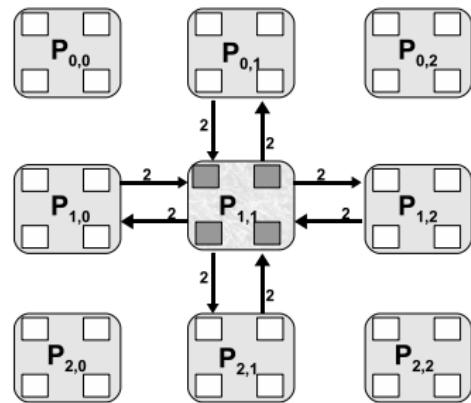


- 4 envíos por tarea
- 1 elemento por envío
- 144 envíos totales
- 144 elementos transferidos

Cálculo de diferencias finitas. Distribución por bloques



Bloques de filas sobre $p = 6$
2 envíos(6 els./envío)/tarea
72 elementos intercambiados



Bloques 3×3 sobre malla 3×3
4 envíos(2 els./envío)/tarea
72 elementos intercambiados

Consideraciones sobre el ejemplo

Efecto Superficie-Volumén

La carga computacional aumenta de forma proporcional a la superficie del subdominio, mientras que el volumen de comunicación aumenta proporcionalmente al perímetro del subdominio

Descomposición 1D vs. 2D

- El máximo número de procesos está más limitado en 1D.
- En 1D, la frecuencia de comunicaciones es menor (la mitad).
- En 2D, el volumen de mensajes se reduce conforme aumenta N y p .

Consideraciones sobre el ejemplo

Comparativa. Num. mensajes transferidos/tarea 1D vs. 2D

p	Unidimensional	Bidimensional
4	$2n$	$2n$
9	$2n$	$2n/3$
16	$2n$	n
25	$2n$	$2n/5$
36	$2n$	$n/3$

Conforme aumenta p , el aumento en la frecuencia de comunicaciones se ve compensado con creces con una sustancial reducción en el número de elementos transferidos

Conclusión: Las distribuciones por bloques de mayor dimensión suelen ser más eficientes y escalables.

Distribución cíclica por bloques

Motivación

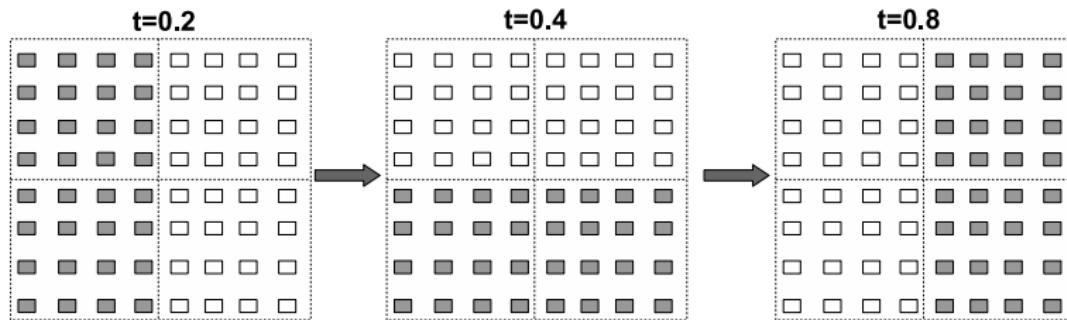
- **Distribuciones por bloques:** ideales cuando existe localidad espacial y todas las celdas tienen el mismo coste.
- A menudo hay bastante localidad espacial pero el coste por celda varía en el tiempo.

Cálculo de diferencias finitas con fórmula variable en el tiempo

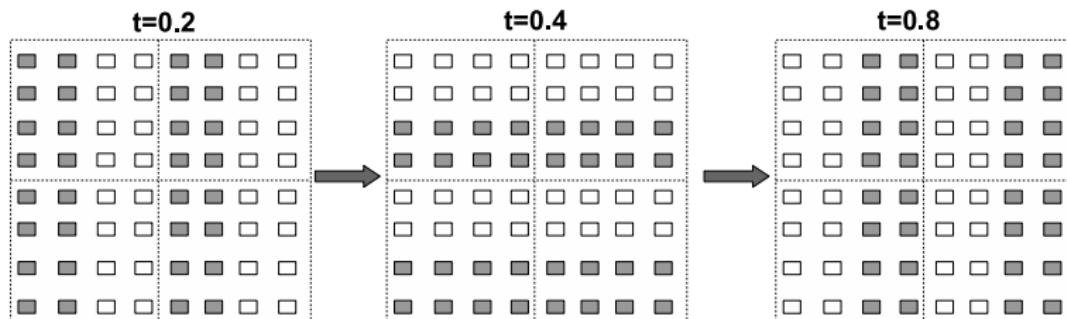
- Si $t \leq 0.25$, $a_{i,j}^{(k+1)}$ sólo se actualiza en base a la fórmula si $j < n/2$. En otro caso, $a_{i,j}^{(k+1)} = a_{i,j}^{(k)}$.
- Si $0.25 < t \leq 0.5$, $a_{i,j}^{(k+1)}$ sólo se actualiza en base a la fórmula si $i \geq n/2$.
- Si $t > 0.5$, $a_{i,j}^{(k+1)}$ sólo se actualiza en base a la fórmula si $j \geq n/2$.



Distribución cíclica por bloques. Motivación



a) Distribución por bloques bidimensionales

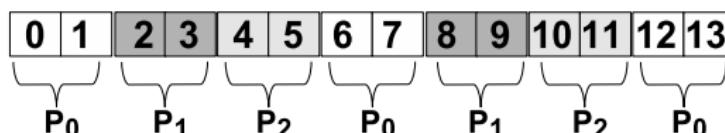


b) Distribución cíclica por bloques 2×2

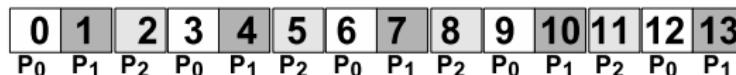
Distribución cíclica por bloques

Estrategia de asignación

- ➊ Trocear matriz en bloques de tamaño especificado por usuario
+ Asignar cíclicamente bloques a malla lógica.
- ➋ Escogiendo bien tamaño bloque, cada proceso mantiene bloques de áreas diferentes sin aumento excesivo comunicación..

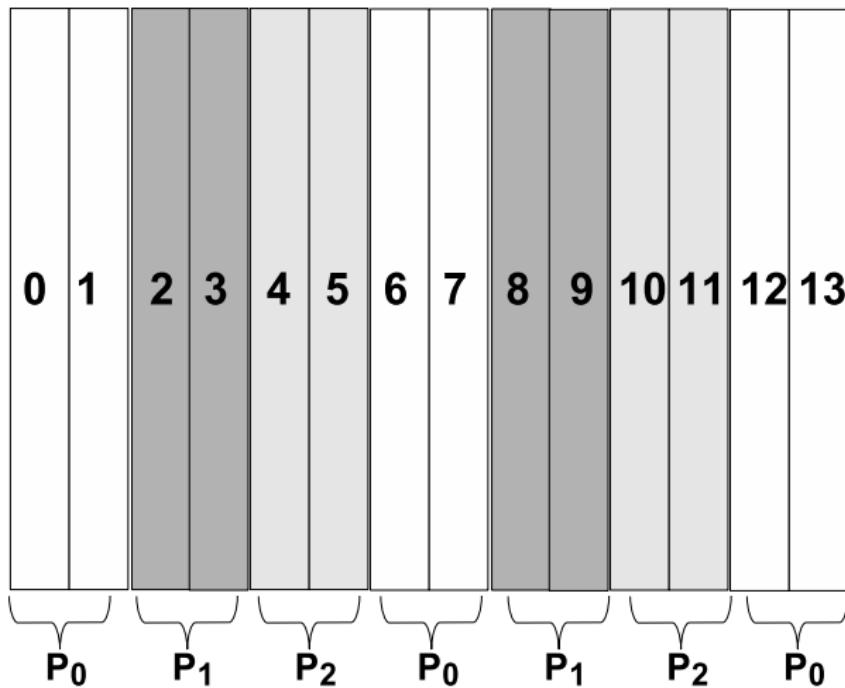


a) Distribución cíclica por bloques



b) Distribución cíclica

Distribución cíclica por bloques de columnas



Distribución cíclica por bloques bidimensionales

Parámetros de la distribución

(p_x, p_y, mb_x, mb_y) .

Esquema de Asignación para una matriz bidimensional A de dimensión $n \times m$

	índ. proceso	índice dentro matriz local
(i, j)	(p_i, p_j) $p_i = \lfloor i / mb_x \rfloor \bmod p_x$ $p_j = \lfloor j / mb_y \rfloor \bmod p_y$	(l_i, l_j) $l_i = \left(\lfloor \frac{i / mb_x}{p_x} \rfloor - 1 \right) mb_x + i \bmod p_x$ $l_j = \left(\lfloor \frac{j / mb_y}{p_y} \rfloor - 1 \right) mb_y + j \bmod p_y$

Distribución cíclica por bloques bidimensionales

Distribución cíclica por bloques 2×3 de una matriz bidimensional 8×11 entre una malla de procesos 3×4

Matriz A
descompuesta
en bloques

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3



Distribución de los bloques
en la malla de procesos

	0	1	2	3
0	0,0 3,0	0,1 3,1	0,2 3,2	0,3 3,3
1	1,0 2,0	1,1 2,1	1,2 2,2	1,3 2,3
2	2,0 3,0	2,1 3,1	2,2 3,2	2,3 3,3

Distribución cíclica por bloques bidimensionales

Distribución por bloque más general

- **Distribución cíclica:** Si se escogen bloques de un elemento, se obtiene la .
- **Distribución por bloques de filas:** Si se escogen bloques de tamaño $\lceil n/p \rceil \times n$ y malla $p \times 1$.
- **Distribución por bloques de columnas:** Si se escogen bloques de tamaño $n \times \lceil m/p \rceil$ y una malla $1 \times p$.
- **Distribución por bloques bidimensionales:** Dada una malla de procesos $p_x \times p_y$, si se escogen bloques de tamaño $\lceil n/p_x \rceil \times \lceil m/p_y \rceil$.

La carga está mejor equilibrada cuanto menor es el tamaño de bloque, y los costes globales de comunicación son menores cuanto mayor es el tamaño de bloque Hay que encontrar un compromiso óptimo.

División estática de grafos

Motivación

- Simulaciones científicas que trabajan sobre grafos no estructurados e irregulares.
- Se trabaja con una estructura de datos irregular pero existe localidad espacial.
- Los esquemas de distribución de bloque no son efectivos ya que las matrices son dispersas y de estructura irregular.

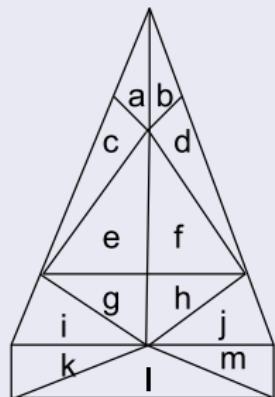
Objetivos asignación

- Equilibrar la carga computacional entre los elementos de procesamiento.
- Minimizar comunicaciones entre procesos. Las comunicaciones son necesarias cuando puntos adyacentes a un punto de malla se asignan a un proceso diferente.

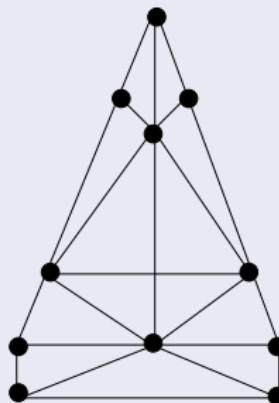


División estática de grafos

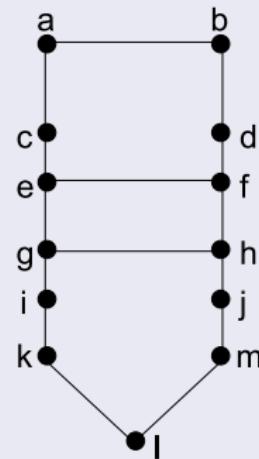
Posibles grafos asociados a una malla 2D basada en triángulos



a)



b)



c)

División estática de grafos

Problema de subdivisión óptima, conocidos $G = (V, E)$ y p

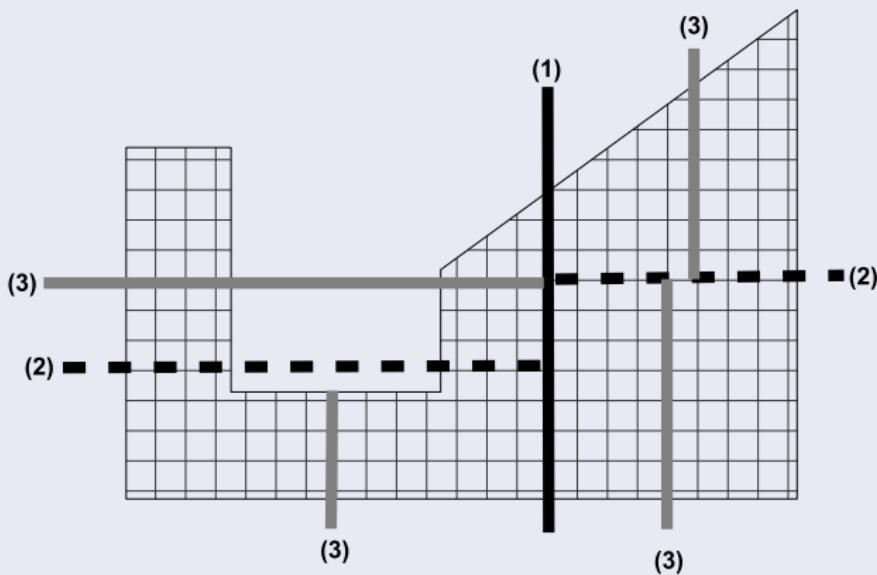
- Se pretende trocear V en p subdominios disjuntos con la misma cantidad de vértices, minimizando el número total de aristas que cruzan los límites de los subdominios.
- Si la carga por vértice y/o el costo por arista es diferente, G es un grafo etiquetado. Se pretende que la suma de las etiquetas de los vértices de cada subdominio sean iguales, minimizando la suma de las aristas que conectan diferentes subdominios.

NP-completitud del problema

El problema es en general NP-completo pero se han desarrollado numerosos enfoques heurísticos que permiten obtener rápidamente buenas soluciones para grandes grafos.

División estática de grafos

Ejemplo de enfoque heurístico. Bisección recursiva de coordenadas



Técnica simple y poco costosa, llevando a cabo una buen troceado de las computaciones pero no optimiza las comunicaciones.



Asignación basada en grafos de dependencias estáticos

Se aplica cuando el resultado de la descomposición es un grafo de dependencias estático en el que el costo de cada tarea es conocido.

Problema de asignación con grafo de dependencias estático

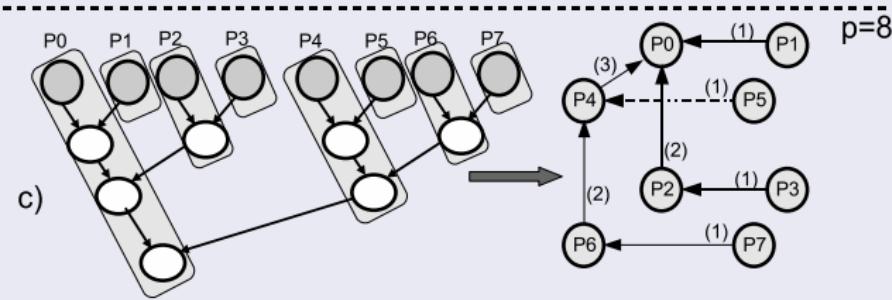
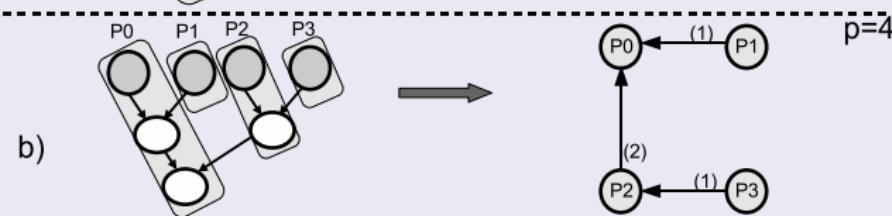
Encontrar un reparto de las tareas del grafo de dependencias entre los procesos, y un orden de ejecución de las tareas, que minimice el tiempo de ejecución.

NP-completitud del problema

El problema es en general NP-completo pero se han desarrollado numerosos enfoques heurísticos y se conocen algoritmos óptimos de tiempo polinomial para situaciones concretas.

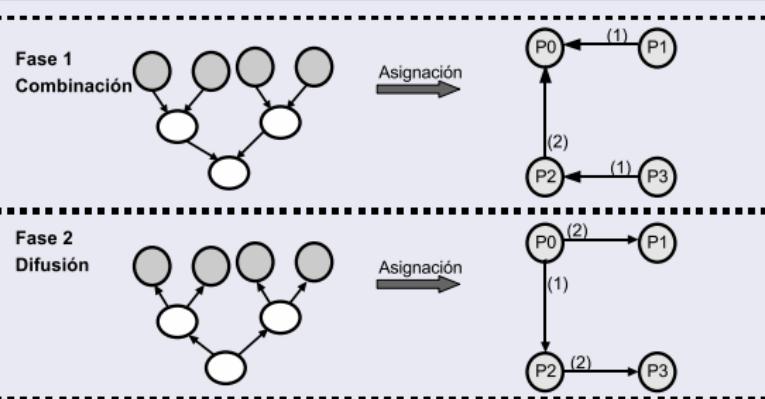
Asignación basada en grafos de dependencias estáticas

Cálculo de reducciones usando una estructura de árbol binomial



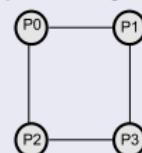
Asignación basada en grafos de dependencias estáticos

Cálculo de reducciones con replicación. Enfoque inicial



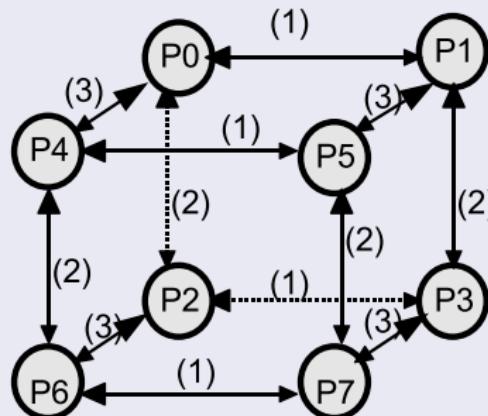
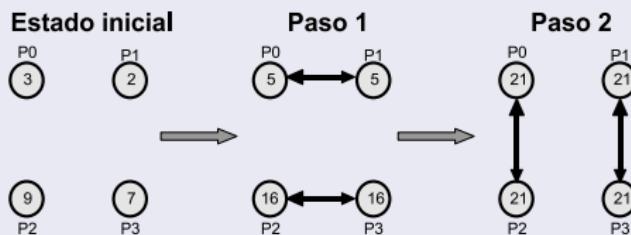
Hipercubo. Construcción incremental

Hipercubo de grado 0 Hipercubo de grado 1 Hipercubo de grado 2



Asignación basada en grafos de dependencias estáticas

Cálculo de reducciones con replicación en hipercubo



Esquemas de equilibrado dinámico de la carga

Ámbito de aplicación

Descomposición recursiva o exploratoria en la que no es eficiente asignación estática.

Reformulación del algoritmo obtenido en la descomposición

- Las tareas pasan a ser estructuras de datos que representan subproblemas.
- La resolución de subproblemas se lleva a cabo por un conjunto de procesos trabajadores.
- Los subproblemas se mantienen en una colección desde la cual se van asignando. La colección puede ser centralizada (localización única) o distribuida entre varios procesos.

Esquemas de equilibrado dinámico de la carga

Ejemplos

- **Integración numérica por cuadratura adaptativa:** la aproximación de la integral en un subintervalo se considera como un subproblema. Su resolución puede requerir la resolución de nuevos subproblemas.
- **Problema de la suma del subconjunto:** la exploración de un nodo se considera como un subproblema.

Mecanismo de detección de fin

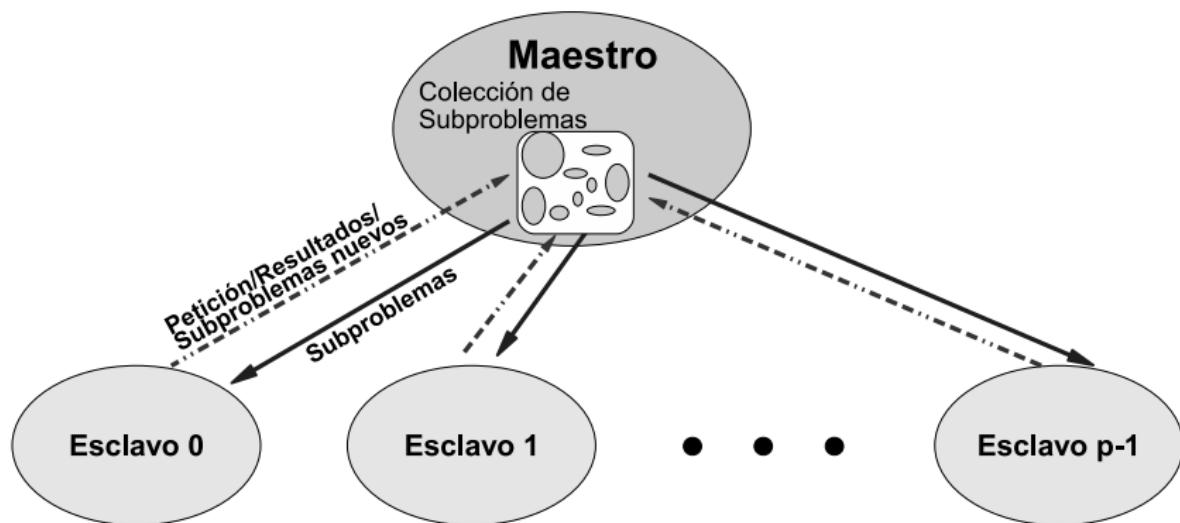
Los procesos necesitan un mecanismo para determinar cuando el trabajo ha terminado. En caso contrario, los procesos ociosos estarían siempre solicitando trabajo. El mecanismo de *detección de fin* dependerá de la aplicación y de la estrategia de planificación.

Esquemas de equilibrado dinámico de la carga

Tipos de esquemas

- **Esquemas centralizados:** La colección de problemas se mantiene centralizada en un única localización y es gestionada por un proceso maestro. El resto de procesos (esclavos) resuelve subproblemas.
- **Esquemas completamente descentralizados:** Los problemas se encuentran distribuidos entre todos los procesos. Se mantiene una colección local de problemas en cada proceso.
- **Esquemas mixtos:** A caballo entre una estrategia centralizada y descentralizada.

Esquemas centralizados de equilibrado dinámico de la carga



Esquemas centralizados de equilibrado dinámico de la carga

Planificación centralizada para la cuadratura numérica adaptativa

- **Descomposición:** Solución basada en memoria compartida que crea nueva tarea para cada subintervalo. Ineficiente en una máquina real.
- **Asignación:** se han de agrupar tareas para minimizar sobrecargas. Se pretende que cada proceso se encargue de aproximar la integral de la función en varios subintervalos.
- El Maestro inicialmente genera tantos subproblemas iniciales como esclavos y los reparte entre esclavos.
- Cada subproblema identifica el calculo numérico de la integral en un subintervalo.
- Los esclavos solicitan subproblemas y los resuelven. La resolución puede generar más subproblemas que son enviados al Maestro.



Esquemas centralizados de equilibrado dinámico de la carga

Proceso Maestro ()

AREA=0; Completado=FALSO;

Generar tantos subproblemas como n_esclavos;

Actualiza AREA y Completado con los resultados parciales obtenidos;

Meter subproblemas generados en Cola_Subproblemas;

Para $i = 0$ hasta $n_esclavos-1$ hacer

Saca Subproblema de Cola_Subproblemas y Envia(Subproblema, Esclavo_i);

Mientras No Completado hacer

Recibe(Resultados, Esclavo_i);

Actualiza AREA y Completado con Resultados;

Si Hay subproblemas en Resultados entonces

Mientras Cola_Ociosos no vacía y Existan subproblemas hacer

Sacar j de Cola_Ociosos;

Envia(Mensaje_con_Subproblema, Esclavo_j);

Meter resto de subproblemas en Cola_Subproblemas;

Si Cola_Subproblemas no vacía entonces

Saca Subproblema de Cola_Subproblemas;

Envia(Mensaje_con_Subproblema, Esclavo_i);

sino Meter i en Cola_Ociosos;

Para $i = 0$ hasta $n_esclavos-1$ hacer

Envia(Mensaje_Fin, Esclavo_i);



Esquemas centralizados de equilibrado dinámico de la carga

Proceso Esclavo_i(), i=0,...,num_esclavos-1

fin=FALSO;

Mientras *no fin* **hacer**

Recibe(*Mensaje, Maestro*);

Si *Mensaje_Fin entonces*

fin=VERDADERO;

Si *Mensaje_con_Subproblema entonces*

Resuelve *Subproblema* generando *Resultados*;

Envia(*Resultados, Maestro*);

Esquemas centralizados de equilibrado dinámico de la carga

¿Cuándo interesa usar un enfoque centralizado?

Cuando el número de esclavos es moderado y el coste de ejecutar problemas es alto comparado con el coste de obtenerlos.

Mejoras para reducir cuello de botella

- **Planificación por bloques**
- **Colecciones locales de subproblemas**
- **Captación anticipada de subproblemas**

Detección de Fin sencilla

La lleva a cabo el maestro cuando detecta:

- a) Colección de subproblemas se agota.
- b) Todos los esclavos están ociosos y han solicitado trabajo.

Esquemas completamente descentralizados

Características

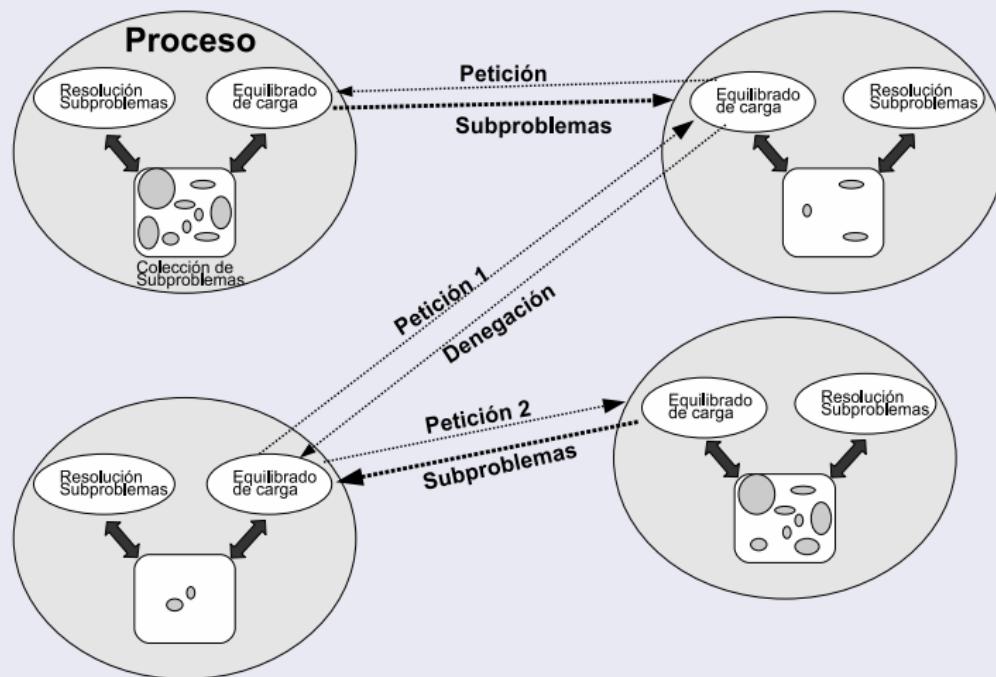
- Colección subproblemas distribuida entre los procesos, habiendo una colección local en cada proceso.
- Los procesos acceden asíncronamente a la colección distribuida para equilibrar carga.

Estrategias generales para transferencia de carga

- **Transferencia iniciada por el receptor**: cuando proceso necesita trabajo, selecciona una serie de procesos y contacta con ellos para obtener parte de su carga. Funciona bien con carga global alta.
- **Transferencia iniciada por el emisor**: proceso cargado transfiere a los que deseen aceptar. Funciona bien con carga global baja.

Esquemas completamente descentralizados

Ejemplo: Transferencia iniciada por el receptor



Esquemas completamente descentralizados

Aspectos a tener en cuenta

- **Selección de donantes de carga candidatos**
 - Sondeo aleatorio
 - Sondeo cíclico
- **Troceado de carga en emisor:** Lo ideal es partir la colección en dos trozos con el mismo coste.
- **Mecanismo de detección de fin:** Más complejo que centralizado.
 - No hay registro central de procesos ociosos
 - Puede haber mensajes en tránsito con subproblemas incluso con procesos ociosos

Esquemas mixtos de equilibrado de carga. Ejemplos

Gestión centralizada jerárquica

- Consiste en dividir el conjunto de esclavos en subconjuntos, cada uno gestionado por un submaestro diferente.
- Los submaestros se comunican periódicamente entre ellos y/o con el maestro central para equilibrar carga.

Colección distribuida con gestión de carga centralizada

:

- Colección distribuida pero existe un maestro al que los trabajadores comunican periódicamente carga y resultados.
- Un trabajador ocioso envía petición al maestro y el maestro le devuelve el identificador del proceso al que debe solicitar trabajo.

Esquemas completamente descentralizados. Ejemplo

Estrategia iniciada por el receptor

- Cada proceso mantiene una colección local de subproblemas
- Cada proceso se comunica con otro para:
 - Pedir trabajo
 - Ofrecer parte de sus subproblemas al resto

Mecanismo de detección de fin

- Paso de testigo en anillo
- Integrado con el mecanismo de equilibrado de carga

Esquemas completamente descentralizados. Ejemplo

Proceso id ($id = 0, \dots, P - 1$)

Inicializa $Cola_local$; $AREA_local = 0$; $fin = falso$;

Si $id == 0$ **entonces**

$Subproblema = Lee(Problema_Global)$;

sino

$Equilibrado_Carga (Cola_local, fin, id)$;

Si $no\ fin$ **entonces** Saca $Subproblema$ de $Cola_local$;

Mientras $no\ fin$ **hacer**

 Resuelve $Subproblema$ generando resultados y subproblemas ;

 Actualiza $AREA_local$ y Almacena subproblemas en $Cola_local$;

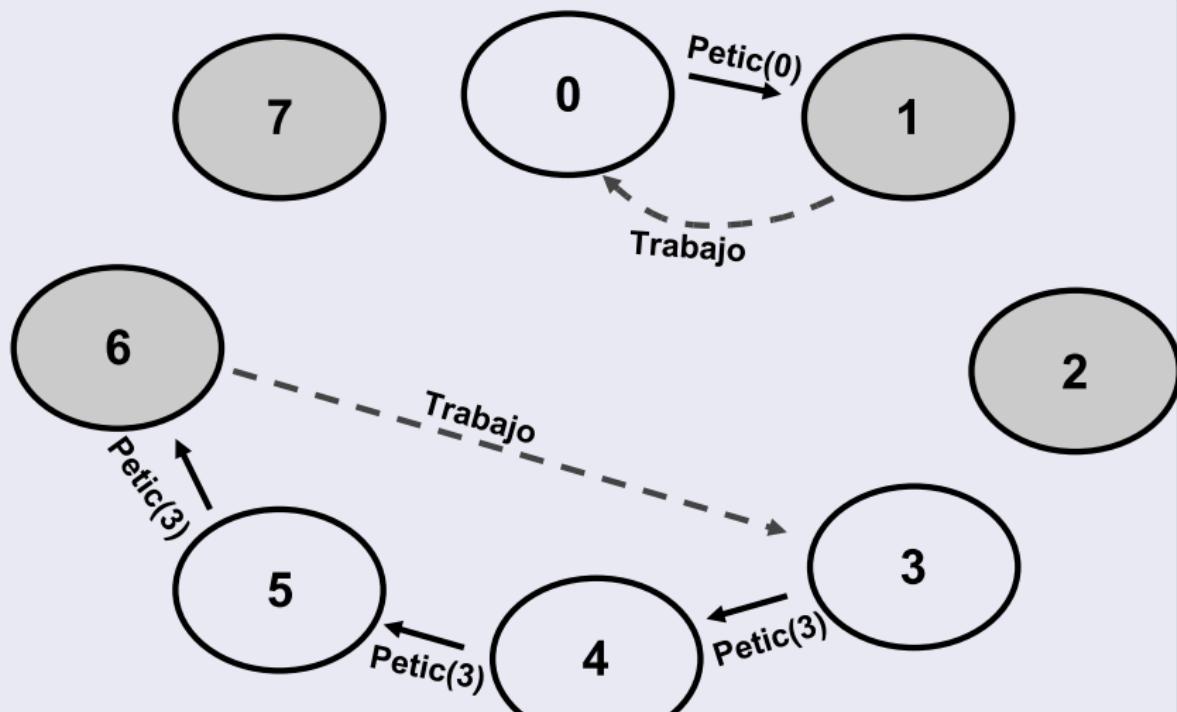
$Equilibrado_Carga (Cola_local, fin, id)$;

Si $no\ fin$ **entonces** Saca $Subproblema$ de $Cola_local$;

$AREA = Reducción(AREA_local)$;

Esquemas completamente descentralizados. Ejemplo

Equilibrado de carga en anillo





Esquemas completamente descentralizados. Ejemplo

Equilibrado_Carga (Cola_local, fin, id)

Si *Cola_local vacía entonces*

Enviar Mensaje_Petición[*id*] al proceso siguiente;

Mientras *Cola_local vacía y no fin hacer*

Recibir Mensaje de otro proceso;

Si *Mensaje_trabajo entonces*

Almacenar subproblemas en *Cola_local*;

Si *Mensaje_Petición de proc. anterior entonces*

Reenviar Mensaje_Petición[*id*] al proc. siguiente;

Si *solicitante == id entonces ... Iniciar posible detección de fin;*

Si *no fin entonces*

Sondear si hay mensajes pendientes de otros procesos;

Mientras *hay mensajes hacer*

Recibir Mensaje_peticion;

Si *hay suficientes subproblemas en Cola_local entonces*

Enviar subconjunto de subproblemas al proceso solicitante

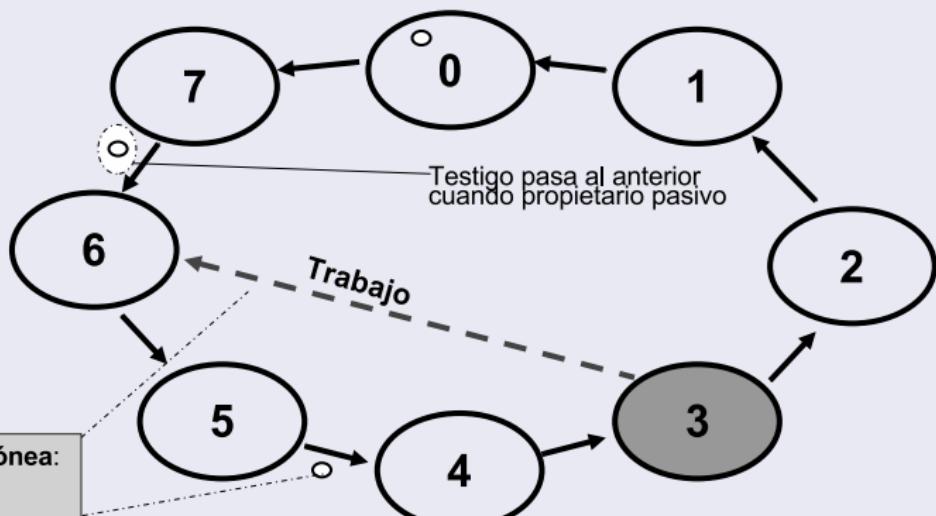
sino Reenviar Mensaje_Petición[*id*] al proc. siguiente;

Sondear si hay mensajes pendientes de otros procesos;



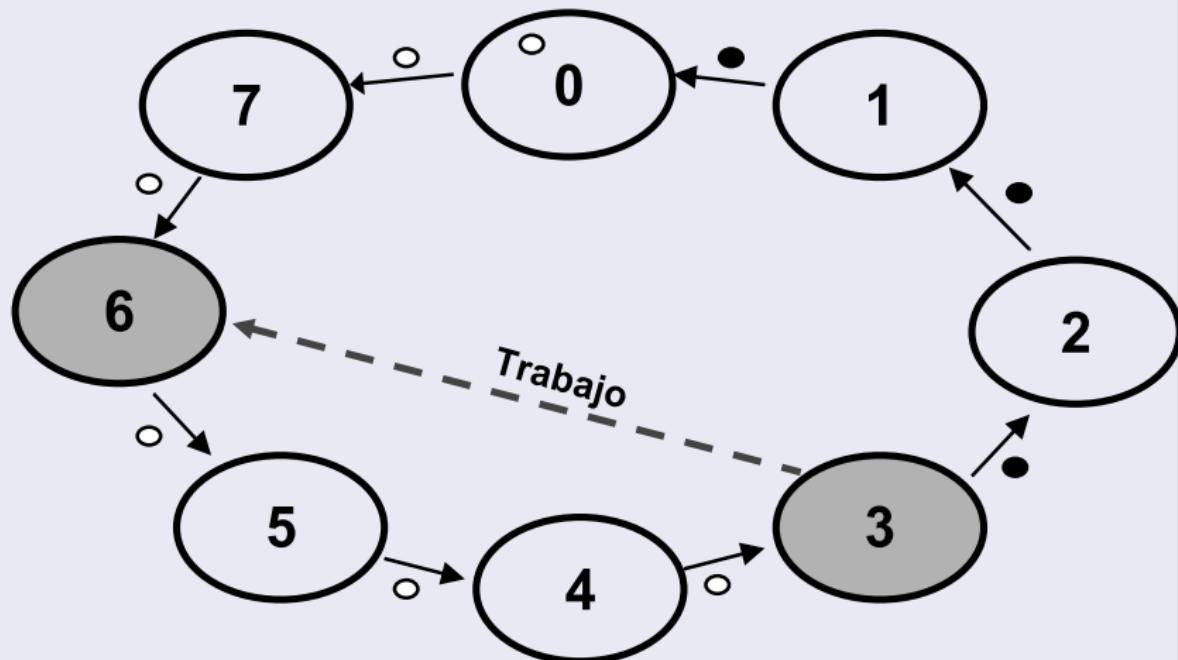
Esquemas descentralizados. Detección de fin en anillo

Aproximación inicial



Esquemas descentralizados. Detección de fin en anillo

Algoritmo de Dijkstra





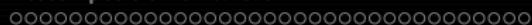
Esquemas completamente descentralizados. Ejemplo

Equilibrado_Carga (Cola_local, fin, id)

```

mi_color = BLANCO;
Si Cola_local vacía entonces
    Enviar Mensaje_Petición[id] al proc. siguiente;
    Mientras Cola_local vacía y no fin hacer
        Si Mensaje_Trabajo entonces . . . ; estado = ACTIVO;
        Si Mensaje_Petición de proc. anterior entonces
            Reenviar Mensaje_Petición[id] al proc. siguiente;
            Si solicitante == id entonces
                estado = PASIVO;
            Si testigo_mio entonces
                Si id == 0 entonces color_testigo = BLANCO; sino color_testigo = mi_color;
                Enviar Mensaje_testigo a anterior; testigo_mio = FALSO; mi_color = BLANCO;
            Si Mensaje_testigo desde proc. siguiente entonces
                testigo_mio = VERDADERO;
                Si estado == PASIVO entonces
                    Si id == 0 y mi_color == BLANCO y color_testigo == BLANCO entonces
                        fin = VERDADERO; Recibir Mensajes de Petición pendientes ;
                        Enviar Mensaje_fin al proc. anterior; Recibir Mensaje_fin del proc. siguiente;
                    sino
                        Si id == 0 entonces color_testigo = BLANCO; sino color_testigo = mi_color;
                        Enviar Mensaje_testigo a anterior; testigo_mio = FALSO; mi_color = BLANCO;
            Si Mensaje_fin desde proc. siguiente entonces
                fin = VERDADERO; Enviar Mensaje_fin al proc. anterior;

```



Esquemas completamente descentralizados. Ejemplo

Equilibrado_Carga (Cola_local, fin, id) Continuación

Si no fin entonces

Sondear si hay mensajes pendientes de otros procesos;

Mientras hay mensajes hacer

. . .; **Si Mensaje_petición entonces**

Si hay suficientes subproblemas entonces

Enviar subproblemas al proc. *solicitante*;

Si id < solicitante entonces

mi_color = NEGRO;

sino

Reenviar Mensaje_Petición[*id*] al proc. siguiente;

Si Mensaje_testigo de proc. siguiente entonces

testigo_mio = VERDADERO;