

SISTEMAS PARALELOS

Clase 8 – Diseño y modelos de algoritmos
paralelos

Prof Dr Enzo Rucci



FACULTAD DE INFORMATICA



UNIVERSIDAD
NACIONAL
DE LA PLATA

Agenda de la clase anterior

- Análisis de rendimiento

Agenda de esta clase

- Diseño de algoritmos paralelos
 - Etapa de descomposición de tareas
 - Etapa de mapeo de tareas a procesadores
 - Métodos para reducir overhead de las interacciones
- Modelos de algoritmos paralelos

DISEÑO DE ALGORITMOS PARALELOS

Diseño de algoritmos paralelos

- Un algoritmo secuencial es esencialmente una receta o secuencia de pasos para resolver un determinado problema empleando una única unidad de procesamiento.
- En forma similar, un algoritmo paralelo se puede ver como una receta que indica cómo resolver un problema *empleando múltiples unidades de procesamiento*.
- Desarrollar un algoritmo paralelo es más que determinar los pasos para resolver el problema → el programador debe al menos considerar la concurrencia y especificar qué pasos pueden llevarse a cabo simultáneamente (esencial para obtener beneficios)

Diseño de algoritmos paralelos

- En la práctica, el diseño de algoritmos paralelos puede incluir algunas de las siguientes actividades:
 - Identificar porciones de trabajo (tareas) que puedan resolverse en paralelo
 - Asignar tareas a procesos que se ejecutan en diferentes procesadores
 - Distribuir datos de entrada, de salida e intermedios asociados con el programa
 - Administrar accesos a datos compartidos
 - Sincronizar procesos en diferentes etapas del programa
- Pueden existir diferentes opciones para cada paso aunque usualmente pocas combinaciones son las que producen buen rendimiento
- Pasos fundamentales: ***Descomposición en tareas y Mapeo de tareas a procesos***

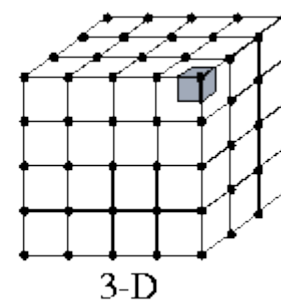
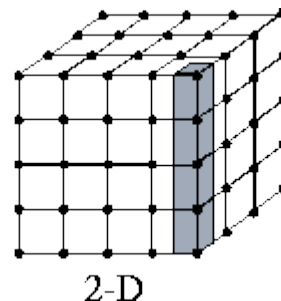
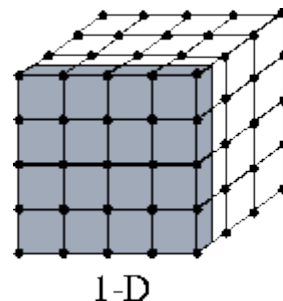
ETAPA DE DESCOMPOSICIÓN EN TAREAS

Descomposición en tareas

- La *Descomposición* es el proceso de dividir el cómputo en partes más pequeñas (*Tareas*), de las cuales algunas o todas podrán ser potencialmente ejecutadas en paralelo.
- Se trata de definir un gran número de tareas pequeñas para obtener una descomposición de grano fino → brinda mayor flexibilidad a los potenciales algoritmos paralelos
- Probablemente, en etapas posteriores, la evaluación de los requerimientos de comunicación, la plataforma destino, o cuestiones de ingeniería de software, pueden llevar a descartar algunas opciones de descomposición consideradas inicialmente
 - En esos casos, la partición original es revisada y sus tareas son aglomeradas para incrementar su tamaño o granularidad.
- La descomposición se puede realizar de diferentes modos. Una primera aproximación consiste en pensar tareas de igual código (***paralelismo de datos o de dominio***) o tareas de código diferente (***paralelismo funcional***)

Descomposición de datos

- Consiste en descomponer los datos asociados a un problema en pequeñas porciones (usualmente del mismo tamaño) y luego asociarle el cómputo relacionado a las mismas para generar las tareas.
- Esta división llevará a un número determinado de tareas, donde cada una comprende de algunos datos y operaciones a realizar sobre los mismos
- Una operación puede requerir datos de diferentes tareas, lo que implicará comunicación y sincronización
- Diferentes particiones son posibles de acuerdo a la estructura de datos disponible



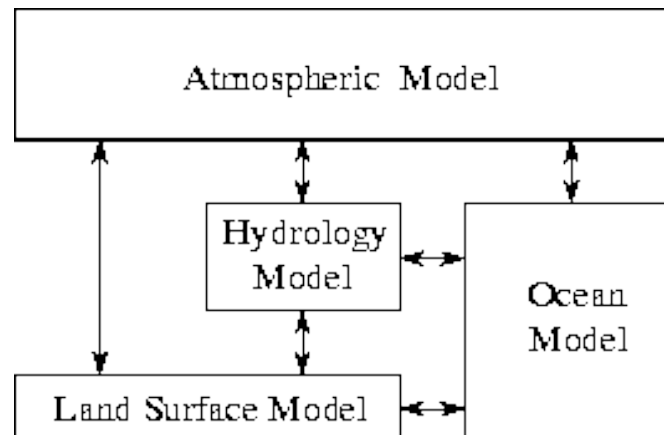
Podría representar el estado de la atmósfera en un modelo climático o un espacio tridimensional en un problema de procesamiento de imágenes

Descomposición funcional

- Se enfoca en el cómputo a realizar más que en los datos → Divide al cómputo en tareas disjuntas y luego examina los datos
- Los requerimientos de datos pueden ser disjuntos (caso ideal) o superponerse significativamente (peor caso → comunicación requerida para evitar replicación de datos).
- La descomposición de datos es la más antigua y, a su vez, la más usada. De todas formas, la descomposición funcional tiene valor como una forma diferente de pensar los problemas
 - Enfocarse en el cómputo a realizar facilita la estructuración del programa y el descubrimiento de oportunidades de optimización (situación no tan obvia cuando uno se enfoca en los datos)

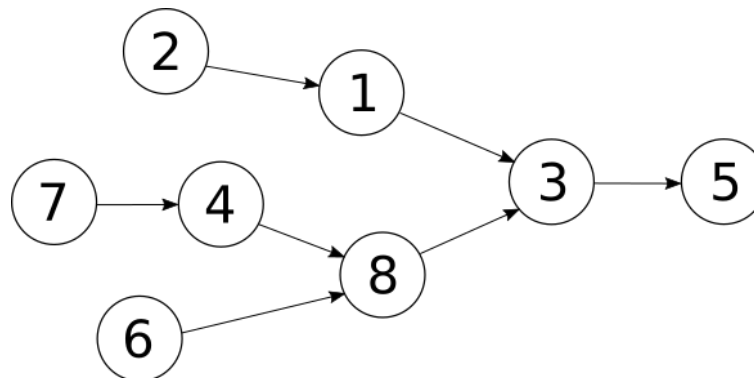
Modelo climático

(1) El modelo atmosférico genera datos sobre la velocidad del viento que son usados por el modelo oceánico; (2) el modelo oceánico genera datos sobre la temperatura de la superficie que son usados por el modelo atmosférico; (3) y así siguiendo...



Descomposición de tareas

- Si el problema lo permite, todas las tareas serán independientes → este es el caso ideal ya que todas podrían computarse a la vez
- En general, esto no es lo usual y existe algún tipo de dependencia entre las tareas.
- Un Grafo de Dependencias de Tareas (GDT) puede ser útil para expresar las dependencias entre las tareas y su orden relativo
 - Es un grafo acíclico dirigido en el que los nodos representan las tareas y las aristas indican las dependencias entre las mismas
 - El grafo puede ser desconexo e inclusive no tener aristas



Ejemplo: consulta en base de datos

- Supongamos que disponemos de la siguiente tabla con información de autos

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	\$18,000
3476	Corolla	1999	White	IL	\$15,000
7623	Camry	2001	Green	NY	\$21,000
9834	Prius	2001	Green	CA	\$18,000
6734	Civic	2001	White	OR	\$17,000
5342	Altima	2001	Green	FL	\$19,000
3845	Maxima	2001	Blue	NY	\$22,000
8354	Accord	2000	Green	VT	\$18,000
4395	Civic	2001	Red	CA	\$17,000
7352	Civic	2002	Red	WA	\$18,000

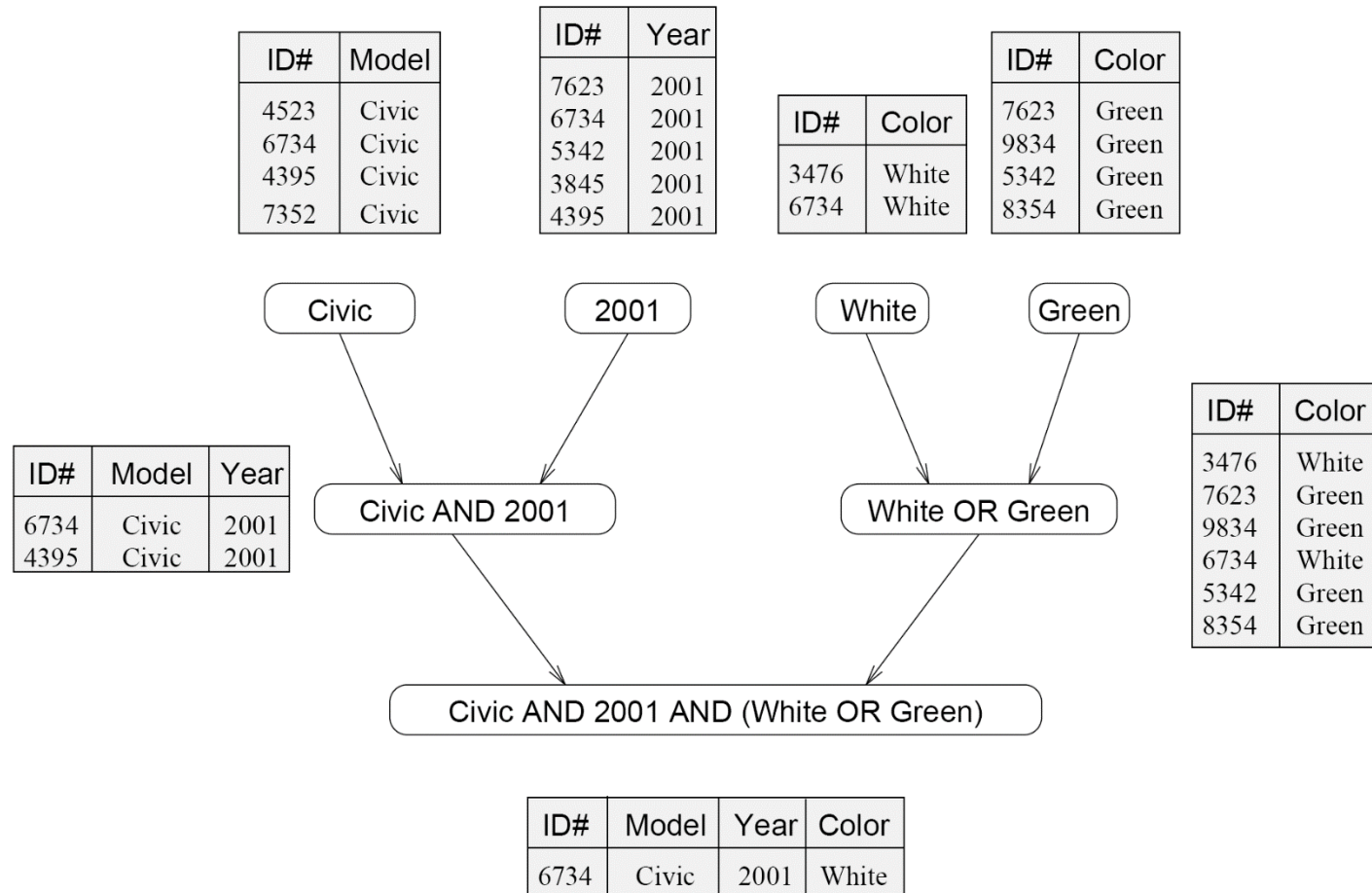
- La consulta a realizar es la siguiente:

Model="Civic" AND Year=2001 AND (Color="Green" OR Color="White")

- La ejecución de la consulta implica crear un número determinado de tablas intermedias, sobre las cuales se ejecutan operaciones de unión o intersección

Ejemplo: consulta en base de datos

- Una posible descomposición:



Ejemplo: consulta en base de datos

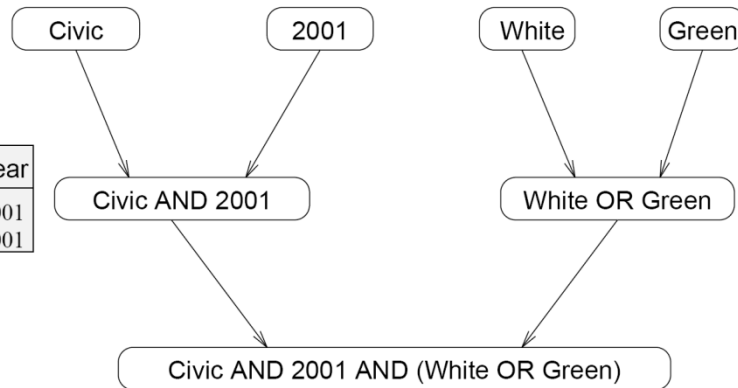
- Una posible descomposición:

ID#	Model
4523	Civic
6734	Civic
4395	Civic
7352	Civic

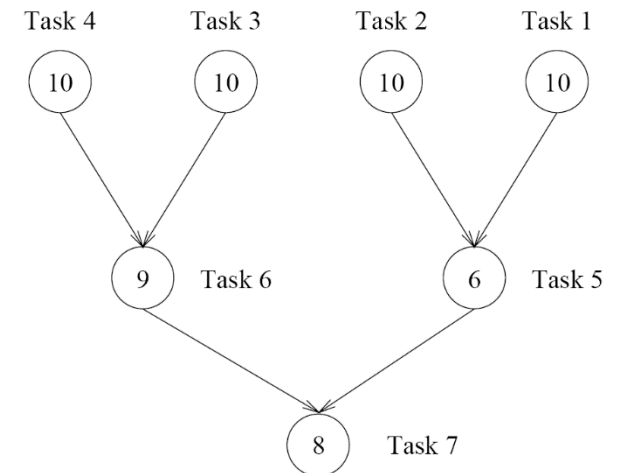
ID#	Year
7623	2001
6734	2001
5342	2001
3845	2001
4395	2001

ID#	Color
3476	White
6734	White

ID#	Color
7623	Green
9834	Green
5342	Green
8354	Green

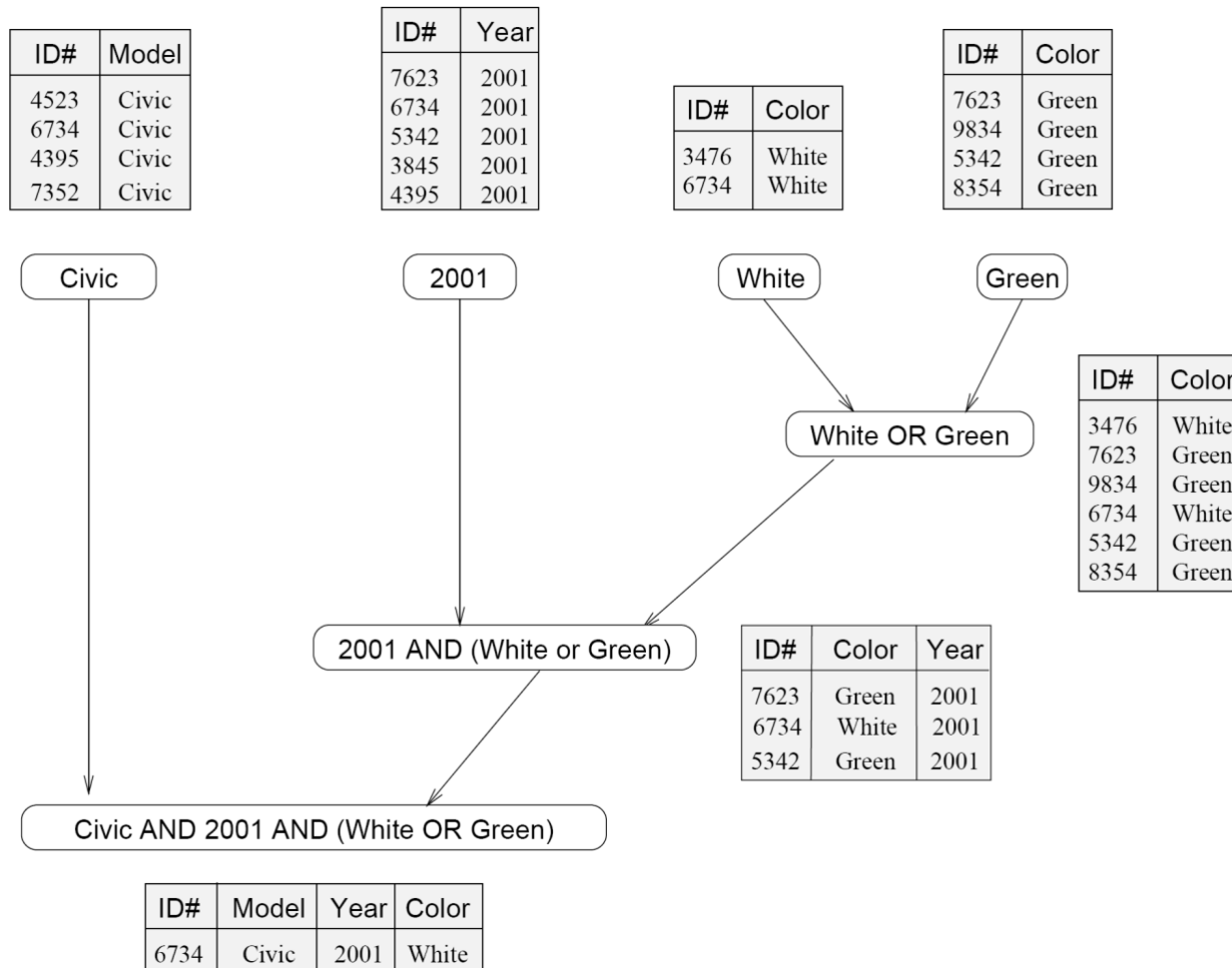


ID#	Color
3476	White
7623	Green
9834	Green
6734	White
5342	Green
8354	Green



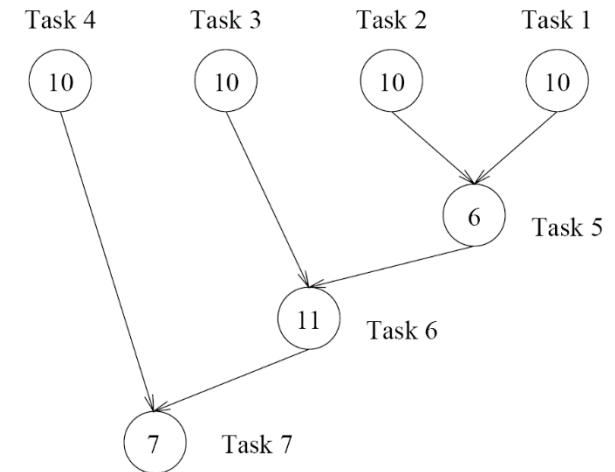
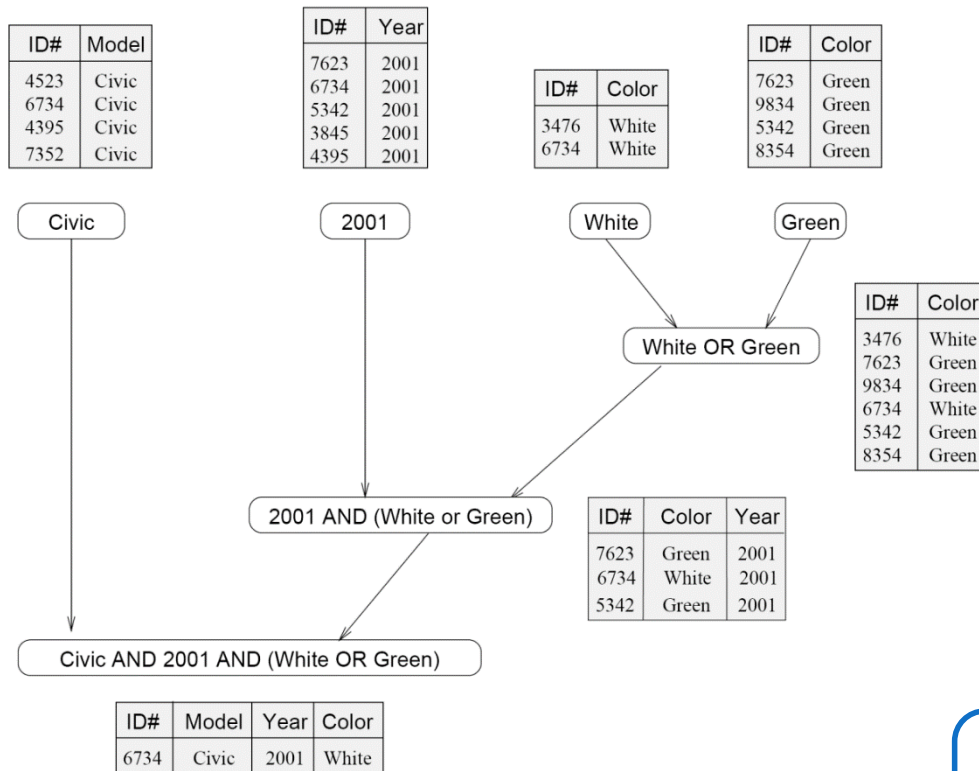
Ejemplo: consulta en base de datos

- Otra posible descomposición:



Ejemplo: consulta en base de datos

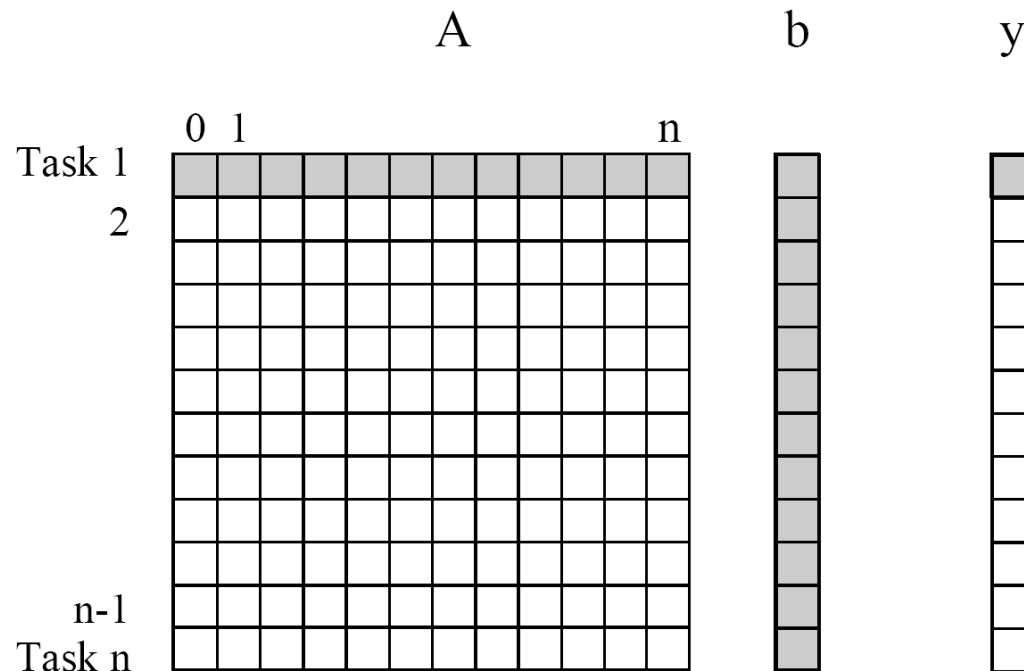
- Otra posible descomposición:



*Diferentes descomposiciones
producen grafos distintos con
características diferentes*

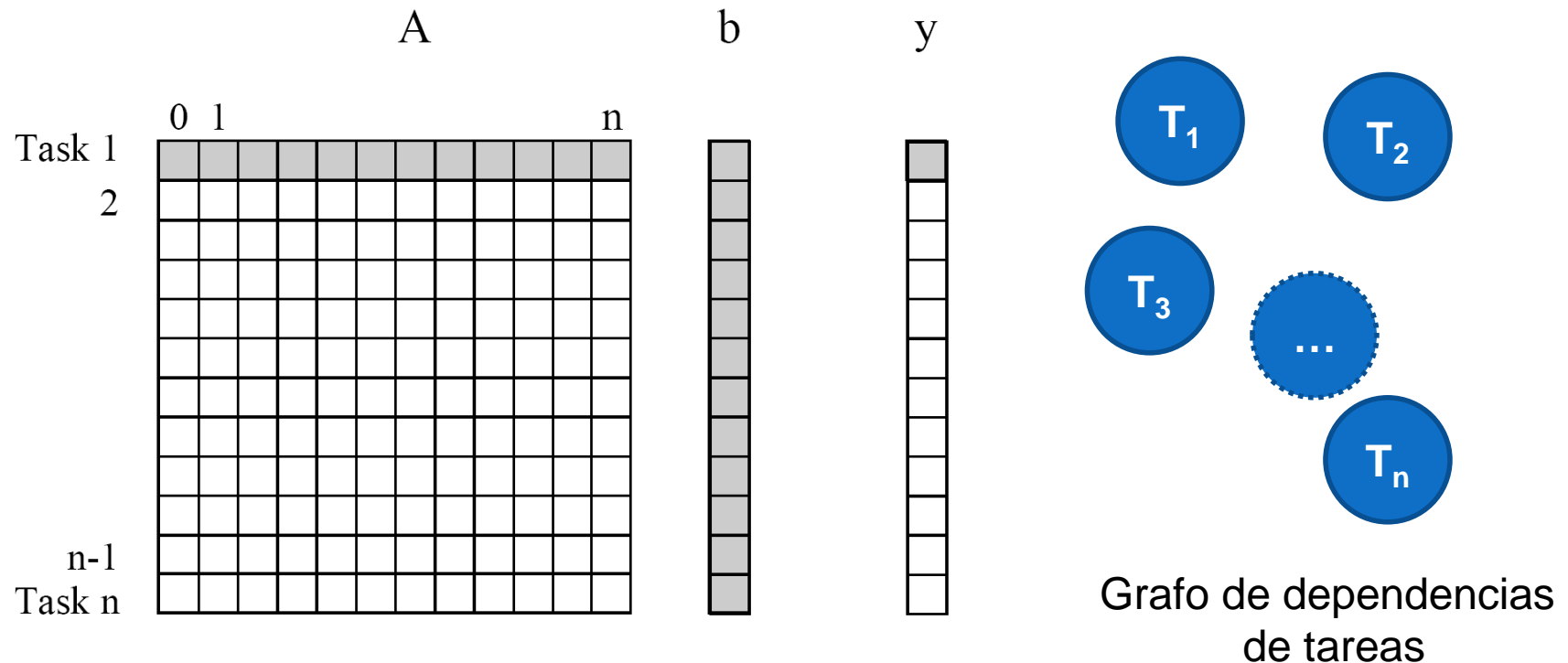
Ejemplo: multiplicación de matriz por vector

- El cómputo de cada celda del vector \mathbf{y} es independiente de las demás \rightarrow Podemos descomponer el problema en n tareas
- En este esquema suponemos que el vector \mathbf{b} es compartido y que a cada tarea cuenta con una fila de \mathbf{A}



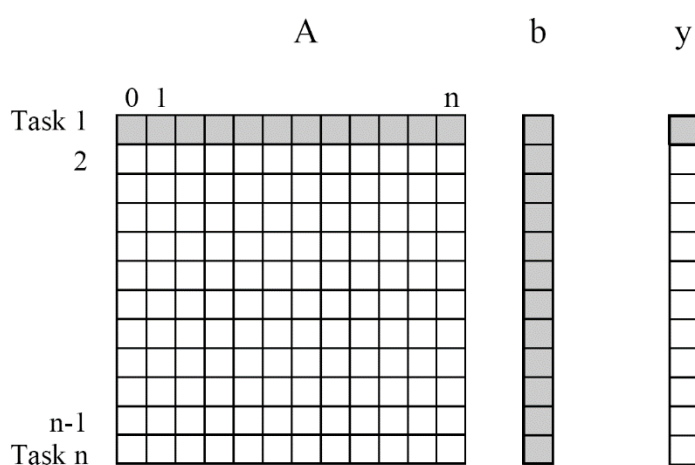
Ejemplo: multiplicación de matriz por vector

- El cómputo de cada celda del vector y es independiente de las demás \rightarrow Podemos descomponer el problema en n tareas
- En este esquema suponemos que el vector b es compartido y que a cada tarea cuenta con una fila de A

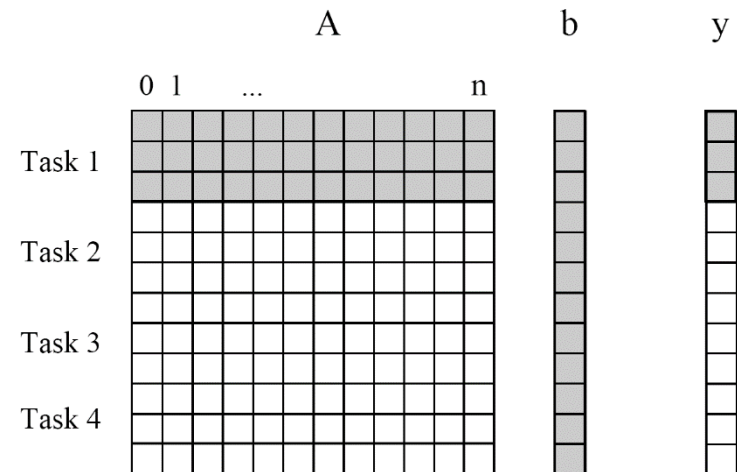


Granularidad de las tareas

- El número y el tamaño de las tareas en las que un problema se divide determinan la granularidad de la descomposición
 - Grano fino*: gran número de pequeñas tareas
 - Grano grueso*: pequeño número de grandes tareas



Grano fino



Grano grueso

Grado de concurrencia

- El número de tareas que se ejecutan en paralelo indica el **grado de concurrencia** de la descomposición
- Como este número puede variar durante la ejecución, un dato interesante a conocer es el **máximo grado de concurrencia** alcanzable por una determinada descomposición.
- Aun más útil que el anterior, es el **grado de concurrencia promedio**, que representa el número promedio de tareas que pueden ejecutarse simultáneamente durante todo el programa → Se relaciona directamente con el número de procesadores que estarán activos y, a su vez, con el balance de carga del programa
- Una característica del grafo de dependencias de tareas que determina el grado de concurrencia promedio para una determinada granularidad es el **camino crítico**.

Grado de concurrencia

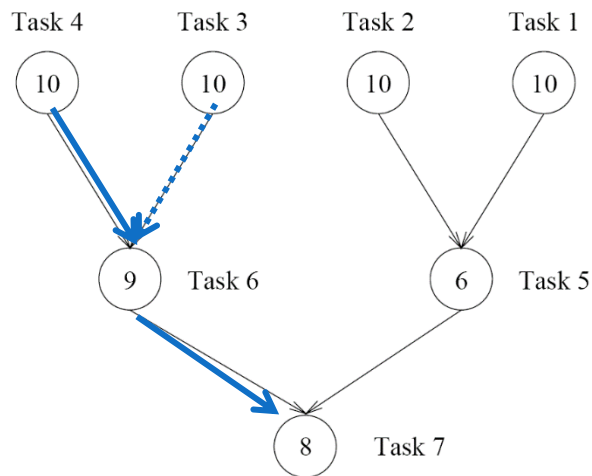
- **Camino crítico:** camino dirigido más largo entre un *nodo inicial* (nodo que no recibe aristas) y un *nodo final* (nodo del que no salen aristas).
- La suma de los pesos de los nodos que integran el camino crítico se conoce como **longitud del camino crítico** → si los pesos indican el tiempo requerido por una tarea, entonces representa el tiempo mínimo requerido para resolver el problema.

$$\text{Grado de concurrencia promedio} = \frac{\text{Peso total}}{\text{Longitud del camino crítico}}$$

- Un camino crítico más corto favorece a un mayor grado de concurrencia

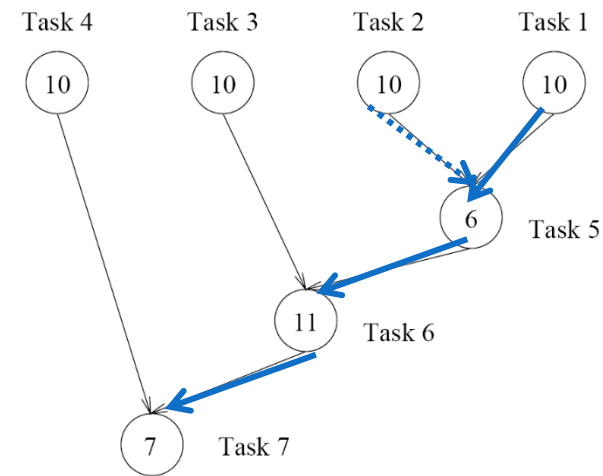
Grado de concurrencia: ejemplo

- Grafos de dependencias de tareas para las dos descomposiciones analizadas de la consulta a la base de datos



(a)

$$GCP = \frac{63}{27} = 2.33$$



(b)

$$GCP = \frac{64}{34} = 1.88$$

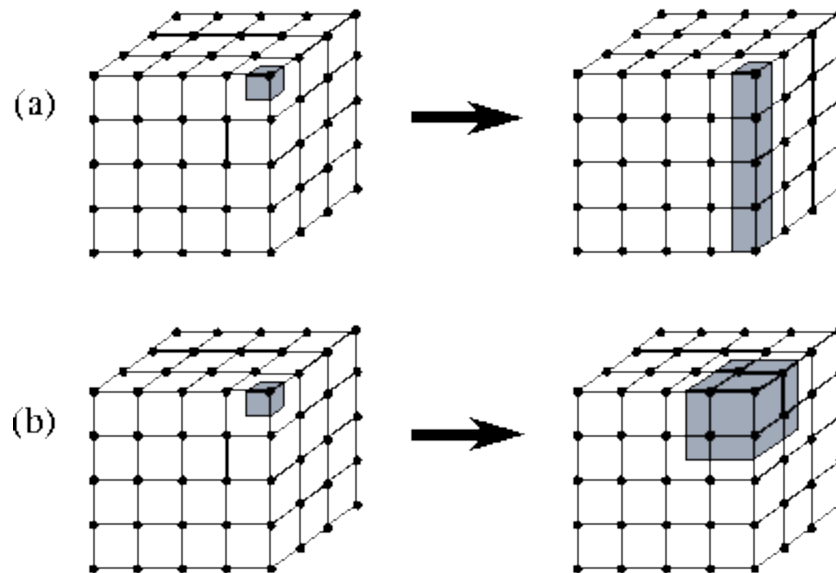
Grado de concurrencia

- Del análisis anterior, podría parecer que el tiempo de ejecución paralela se puede reducir indefinidamente al lograr una descomposición que sea cada vez más fina (más tareas de menor tamaño) → esto no es lo usual
- En general, hay un límite inherente al problema sobre qué tan fina puede ser una descomposición. Por ejemplo, el producto matriz-vector no admite más de n^2 tareas concurrentes.
- Además se debe tener en cuenta que las tareas deben comunicarse y sincronizar → esto significa overhead que limita el speedup alcanzable

Un adecuado balance entre cómputo y comunicación definirá el rendimiento alcanzable

Aglomeración de tareas

- Este paso consiste en analizar si conviene combinar/aglomerar varias tareas para obtener un número de tareas menor pero de mayor tamaño. También se analiza si vale la pena replicar datos o cómputo.

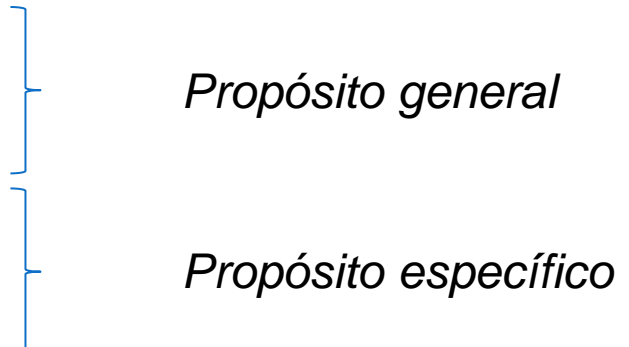


- En general, el número final de tareas como resultado de la aglomeración debería ser igual al número de procesadores a emplear

Aglomeración de tareas

- 3 objetivos, a veces conflictivos entre ellos, que guían las decisiones de aglomeración y replicación:
 - *Incremento de la granularidad*: al combinar varias tareas relacionadas, se elimina la necesidad de comunicar datos entre ellas.
 - *Preservación de la flexibilidad*: al combinar varias tareas se puede limitar la escalabilidad del algoritmo. Si un algoritmo es capaz de crear un número variable de tareas, entonces posee un mayor grado de portabilidad y escalabilidad.
 - *Reducción de costos de desarrollo*: en ocasiones, el costo desde el punto de vista del proceso de ingeniería de software, puede ser muy elevado para la ganancia asociada

Técnicas de descomposición

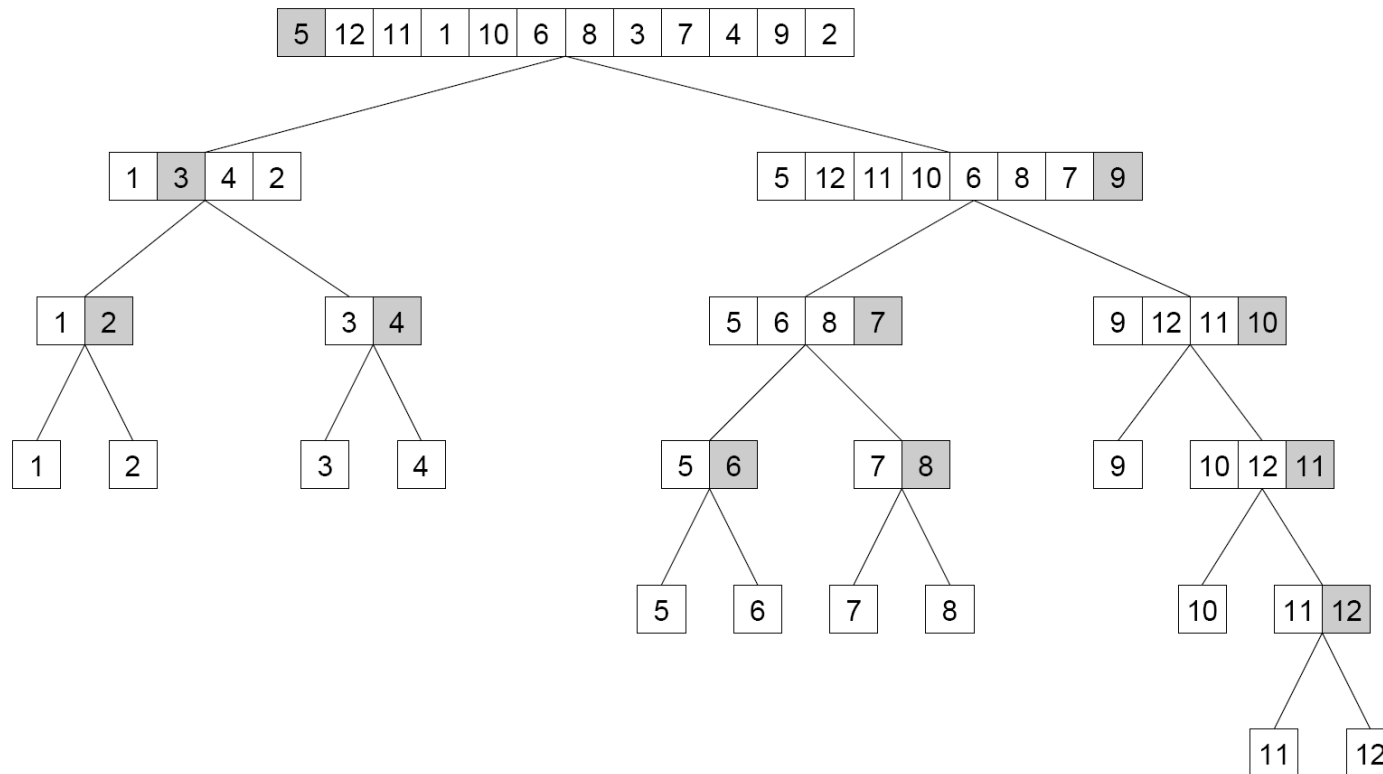
- ¿Cómo descomponer un problema en tareas (y subtareas)?
 - No existe una única forma
 - Algunas técnicas posibles
 - Descomposición recursiva
 - Descomposición basada en los datos
 - Descomposición exploratoria
 - Descomposición especulativa
- 
- Propósito general*
- Propósito específico*

Técnicas de descomposición: Recursiva

- En general se ajusta muy bien a los problemas que se pueden resolver mediante la estrategia *divide y vencerás*.
- El problema inicial es dividido en un conjunto de subproblemas independientes. Luego, cada uno de estos subproblemas son recursivamente descompuestos en otros subproblemas independientes más pequeños hasta alcanzar una determinada granularidad.
- En ocasiones, puede requerirse alguna fase de combinación de resultados parciales.

Técnicas de descomposición: Recursiva

- Un ejemplo de esta descomposición es la ordenación por el método Quicksort, donde una vez que determinamos el pivot, cada porción puede procesarse en forma concurrente y así recursivamente.



Técnicas de descomposición: Recursiva

- A veces puede resultar necesario reestructurar el cómputo de un algoritmo para que sea posible aplicar esta descomposición.
- Un ejemplo es el cálculo del mínimo en una lista desordenada de números

```
1.  procedure SERIAL_MIN ( $A, n$ )
2.  begin
3.     $min = A[0];$ 
4.    for  $i := 1$  to  $n - 1$  do
5.      if ( $A[i] < min$ )  $min := A[i];$ 
6.    endfor;
7.    return  $min$ ;
8.  end SERIAL_MIN
```

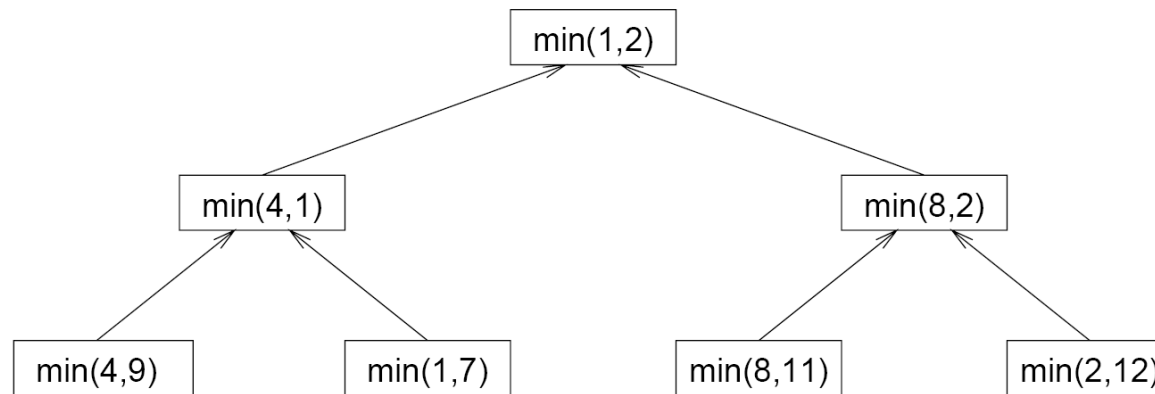
Técnicas de descomposición: Recursiva

- La versión iterativa se puede re-escribir en forma recursiva siguiendo la estrategia *divide y vencerás*.

```
1.  procedure RECURSIVE_MIN ( $A, n$ )
2.  begin
3.  if ( $n = 1$ ) then
4.     $min := A[0];$ 
5.  else
6.     $lmin := \text{RECURSIVE\_MIN} (A, n/2);$ 
7.     $rmin := \text{RECURSIVE\_MIN} (\&(A[n/2]), n - n/2);$ 
8.    if ( $lmin < rmin$ ) then
9.       $min := lmin;$ 
10.   else
11.      $min := rmin;$ 
12.   endelse;
13. endelse;
14. return  $min;$ 
15. end RECURSIVE_MIN
```

Técnicas de descomposición: Recursiva

- Grafo de dependencias de tareas para computar el mínimo de la siguiente lista: {4, 9, 1, 7, 8, 11, 2, 12}



Técnicas de descomposición: Basada en los datos

- Generalmente usada en problemas que operan sobre grandes estructuras de datos
- Requiere de dos pasos:
 - Particionar los datos que se procesarán
 - Usar la partición anterior para inducir una descomposición del cómputo en tareas
- El particionamiento de los datos se puede realizar de diferentes maneras → Se debe analizar las diferentes variantes y elegir la que lleve a una descomposición natural y de buen rendimiento

Técnicas de descomposición: Basada en los datos de salida

- Resulta natural cuando cada elemento de la salida de un programa (resultados) se puede calcular en forma independiente como función de los datos de entrada
- Una partición de los datos de salida lleva inmediatamente a una descomposición en tareas, donde a cada tarea se le asocia el cómputo relacionado a la porción asignada
- Un ejemplo es la multiplicación de matrices

$$\left(\begin{array}{cc} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{array} \right) \cdot \left(\begin{array}{cc} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{array} \right) \rightarrow \left(\begin{array}{cc} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{array} \right)$$

Descomposición de datos

$$\begin{array}{l} \text{Task 1: } C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ \text{Task 2: } C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ \text{Task 3: } C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ \text{Task 4: } C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{array}$$

Descomposición en tareas

Técnicas de descomposición: Basada en los datos de salida

- En general, una determinada descomposición de datos lleva a una dada descomposición del cómputo en tareas, pero puede haber más de una opción.
- En el caso anterior, para una misma descomposición de datos de salida, tenemos al menos dos descomposiciones de tareas:

Descomposición 1	Descomposición 2
Task 1: $C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$ Task 2: $C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$ Task 3: $C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$ Task 4: $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$	Task 1: $C_{1,1} = A_{1,1}B_{1,1}$ Task 2: $C_{1,1} = C_{1,1} + A_{1,2}B_{2,1}$ Task 3: $C_{1,2} = A_{1,2}B_{2,2}$ Task 4: $C_{1,2} = C_{1,2} + A_{1,1}B_{1,2}$ Task 5: $C_{2,1} = A_{2,2}B_{2,1}$ Task 6: $C_{2,1} = C_{2,1} + A_{2,1}B_{1,1}$ Task 7: $C_{2,2} = A_{2,1}B_{1,2}$ Task 8: $C_{2,2} = C_{2,2} + A_{2,2}B_{2,2}$

Técnicas de descomposición: Basada en los datos de entrada

- Particionar los datos de salida no siempre es posible. Ejemplos:
 - Cuando se computa el máximo, el mínimo o la suma de una lista de números → La salida es un único número
 - Cuando se ordena un vector de números, los elementos individuales de la salida no se pueden determinar de antemano
- En estos casos, resulta natural particionar los datos de entrada e inducir concurrencia a partir de ellos
- A cada tarea se le asigna una porción de los datos de entradas y será responsable de realizar todos los cálculos asociados a la misma. En ocasiones, se puede requerir de algún paso posterior de *reducción* de salidas parciales.
- Ejemplos: contar ocurrencia en un vector; búsqueda de un elemento en un vector; descomposición estática para realizar una ordenación, entre otros

Técnicas de descomposición: Basada en los datos intermedios

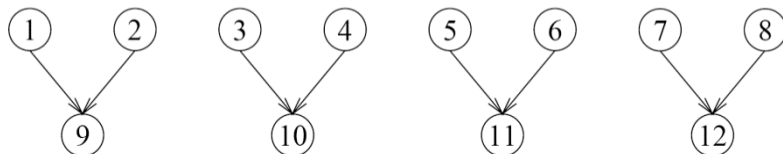
- En algunos casos, los algoritmos pueden ser estructurados en múltiples etapas de forma tal que la salida de una etapa es la entrada de la siguiente.
- La descomposición de un algoritmo de esta clase puede ser derivada al particionar los datos de entrada o de salida de una etapa intermedia, permitiendo un mayor grado de concurrencia.

Stage I

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \left(\begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} \right)$$

Stage II

$$\begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} + \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$



A decomposition induced by a partitioning of D

Task 01: $D_{1,1,1} = A_{1,1} B_{1,1}$

Task 02: $D_{2,1,1} = A_{1,2} B_{2,1}$

Task 03: $D_{1,1,2} = A_{1,1} B_{1,2}$

Task 04: $D_{2,1,2} = A_{1,2} B_{2,2}$

Task 05: $D_{1,2,1} = A_{2,1} B_{1,1}$

Task 06: $D_{2,2,1} = A_{2,2} B_{2,1}$

Task 07: $D_{1,2,2} = A_{2,1} B_{1,2}$

Task 08: $D_{2,2,2} = A_{2,2} B_{2,2}$

Task 09: $C_{1,1} = D_{1,1,1} + D_{2,1,1}$

Task 10: $C_{1,2} = D_{1,1,2} + D_{2,1,2}$

Task 11: $C_{2,1} = D_{1,2,1} + D_{2,2,1}$

Task 12: $C_{2,2} = D_{1,2,2} + D_{2,2,2}$

Técnicas de descomposición: Exploratoria

- Se suele emplear en aquellos problemas cuya solución involucra una búsqueda en un espacio de soluciones.
- Para realizar la descomposición, se particiona el espacio de búsqueda en porciones más pequeñas y se realiza una búsqueda concurrente en cada una de ellas hasta encontrar la solución objetivo.
- Ejemplos clásicos son los problemas de optimización (buscar la mejor configuración para un determinado conjunto de parámetros) y los juegos (puzzle-N, ajedrez, entre otros).

Técnicas de descomposición: Exploratoria

- Ejemplo de Puzzle-15: grilla de 4x4 con 15 bloques (numerados de 1 a 15) y un espacio vacío. El objetivo es que los bloques queden ordenados (de arriba hacia abajo, de izquierda a derecha) en la menor cantidad de movimientos.

1	2	3	4
5	6	7	8
9	10	7	11
13	14	15	12

(a)

1	2	3	4
5	6	7	8
9	10	11	11
13	14	15	12

(b)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	12

(c)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

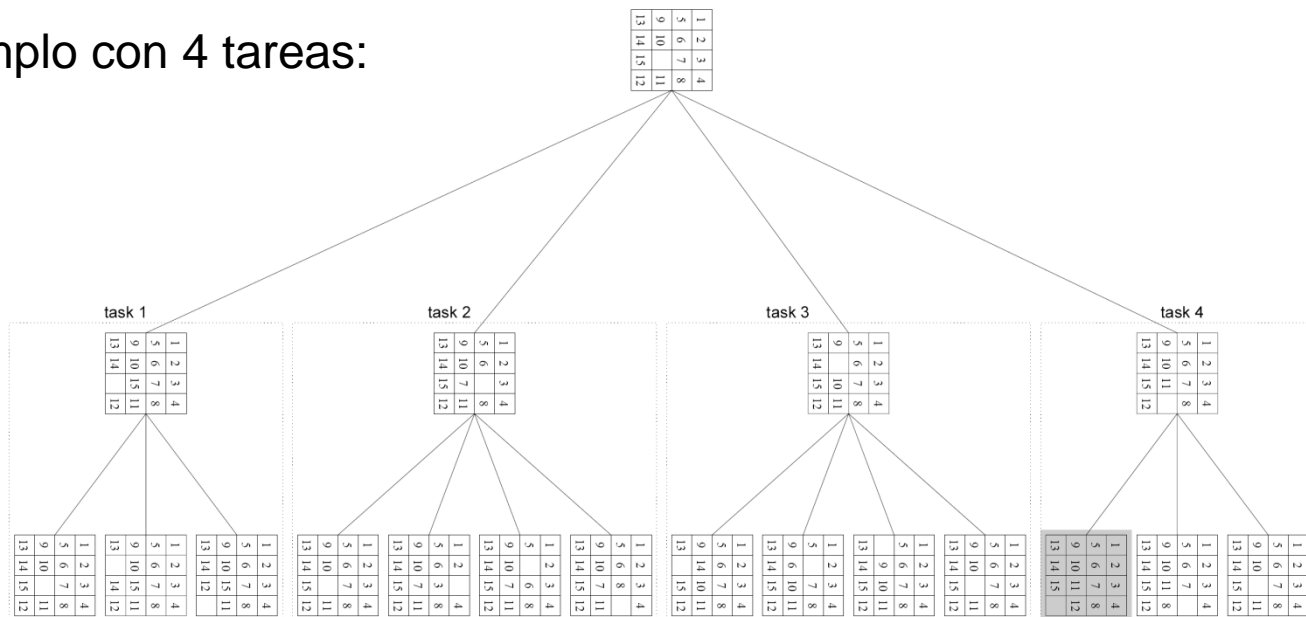
(d)

- Empezando desde la configuración inicial, se generan todas las configuraciones sucesora posibles (entre 2 y 4 posibilidades).
- La tarea de encontrar un camino desde la configuración inicial a la configuración final ahora significa encontrar un camino hasta la configuración final desde alguna de las configuraciones generadas.
- Como alguna de estas configuraciones está más cerca de la solución que la configuración inicial, se ha avanzado hacia la solución.

Técnicas de descomposición: Exploratoria

- Un método usual para resolver el problema en paralelo consiste en desarrollar algunos niveles desde la configuración inicial en forma secuencial. Luego, cada nodo es asignado a una tarea para realizar la búsqueda en forma concurrente. Cuando una la encuentra, le avisa al resto.

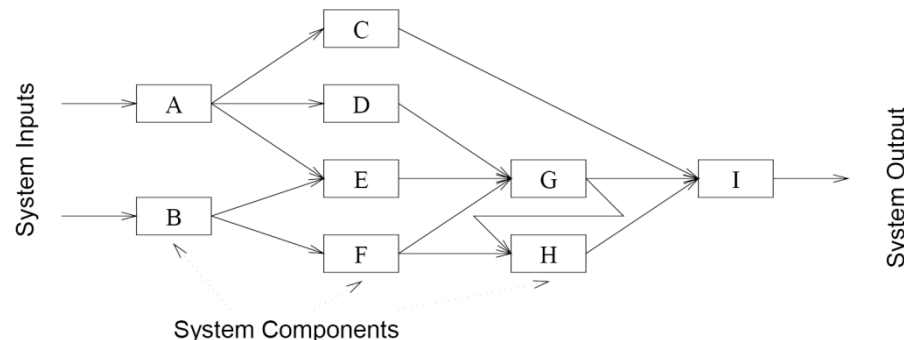
- Ejemplo con 4 tareas:



- Diferencia con descomposición basada en los datos: las tareas son ejecutadas completamente, en exploratoria no.

Técnicas de descomposición: Especulativa

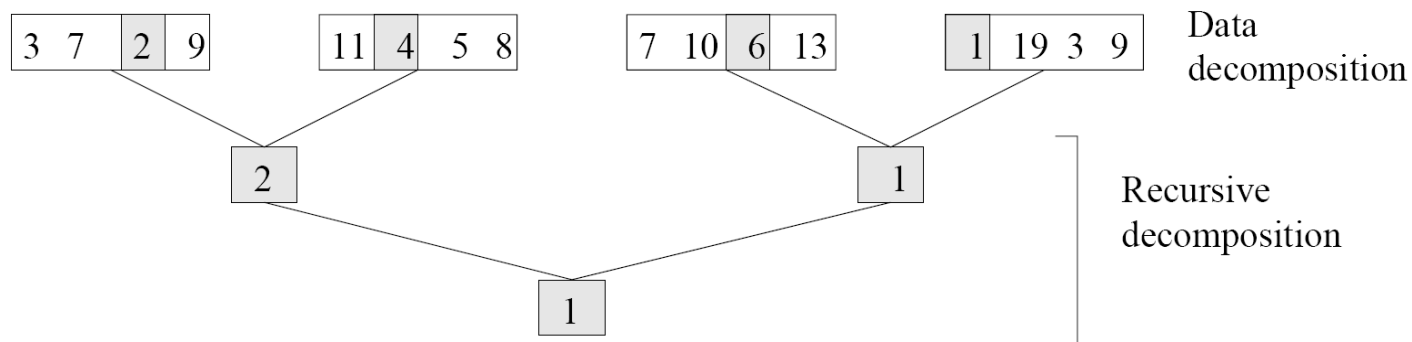
- Se emplea cuando un programa podría tomar uno de varios caminos que implican cómputo significativo pero la decisión depende de la salida de algún cómputo anterior.
- Pensar en un *case* con múltiples opciones que son evaluadas al mismo tiempo pero antes de tener el valor de la entrada. Cuando la entrada del *case* está disponible, se descartan las opciones incorrectas y se continúa la ejecución.
- Ejemplo: simulación de eventos discretos



- Diferencia con exploratoria: En especulativa, la entrada de una bifurcación que lleva a múltiples nuevas tareas es desconocida. En cambio con exploratoria, la salida de las múltiples tareas que salen de una bifurcación son desconocidas.

Técnicas de descomposición: Híbrida

- Las técnicas de descomposición vistas no son exclusivas y se pueden combinar
- En ocasiones, un programa se estructura en múltiples etapas y cada etapa puede ser descompuesta de forma diferente
- Ejemplo: búsqueda del mínimo en un vector.
 - Una descomposición recursiva pura podría generar una cantidad excesiva de tareas si $n \gg P$
 - Una mejor opción consiste en realizar primero una descomposición de datos y luego una recursiva



ETAPA DE MAPEO DE TAREAS A PROCESOS

Características de las tareas

- Las técnicas de descomposición analizadas permiten identificar la concurrencia disponible en un problema y descomponerlo en tareas que podrán ser ejecutadas en paralelo
- El próximo paso en el diseño de algoritmo paralelos consiste en el mapeo (asignación) de las tareas a los procesos del programa.
- Para realizar un buen mapeo, hay que tener en cuenta las características de las tareas:
 - Modo de generación
 - Tamaño y conocimiento del mismo
 - Volumen de datos asociado

Características de las tareas: modo de generación

- Las tareas que constituyen un programa se pueden generar en forma ***estática*** o ***dinámica***.
- En la generación estática, las tareas que se generan se conocen previo a la ejecución
 - Ejemplos: son la multiplicación de matrices o la búsqueda del mínimo de una lista de números
- En la generación dinámica, las tareas se generan durante la ejecución, por lo que no se conoce de antemano cuál será el número final
 - Ejemplo: quicksort recursivo

Características de las tareas: tamaño y conocimiento del mismo

- Las tareas que constituyen un programa pueden ser ***uniformes*** o ***no uniformes***.
 - Cuando las tareas requieren aproximadamente el mismo tiempo de cómputo, se dice que son *uniformes*. Ejemplo: multiplicación de matrices
 - Cuando el tiempo requerido entre una tarea y otra puede variar significativamente se dice que son *no uniformes*. Ejemplo: quicksort recursivo
- Conocer el tamaño de las tareas previo a la ejecución es otro factor que puede influir en el mapeo. Ejemplos:
 - En la multiplicación de matrices conocemos el tamaño de cada tarea previo a la ejecución
 - En el Puzzle-15 no es posible saberlo → no sabemos a priori cuántos movimientos debemos realizar para llegar a la configuración final

Características de las tareas: volumen de datos asociado

- El volumen de datos asociado a una tarea tiene que ver muchas veces con la granularidad elegida.
- A su vez, la granularidad impacta directamente en la relación cómputo-comunicación.
 - Usualmente, con bajos niveles de comunicación se tiende a afinar la granularidad y a asignar un menor volumen de datos por proceso
 - Cuando tenemos mucho intercambio de datos, se suele optar por aumentar la granularidad o emplear memoria compartida (si la arquitectura lo permite)

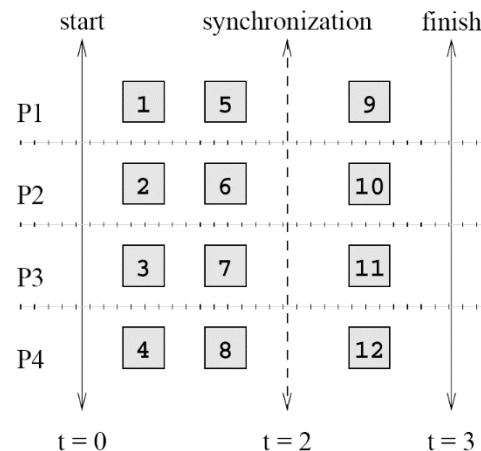
Técnicas de mapeo de tareas a procesos

- Una vez que el cómputo ha sido descompuesto en tareas, el siguiente paso consiste en mapearlas a los procesos del programa.
- El mapeo debe llevarse a cabo buscando que ***el tiempo requerido para completar las tareas sea el mínimo posible***. Para ello se deben considerar 2 estrategias:
 - Asignar tareas independientes en diferentes procesadores para lograr un mayor grado de concurrencia
 - Asignar tareas que se comunican frecuentemente en el mismo procesador reducir overhead y mejorar localidad
- Estas 2 estrategias claramente entran en conflicto entre sí y la clave está en encontrar el balance adecuado.
- El problema de encontrar un mapeo óptimo es NP-completo → esto significa que no existe un algoritmo de complejidad polinomial que evalúe los diferentes compromisos entre las estrategias en el caso general y determine cuál es el mejor
- Sin embargo, existen heurísticas para determinadas clases de problema que suelen dar buen resultado

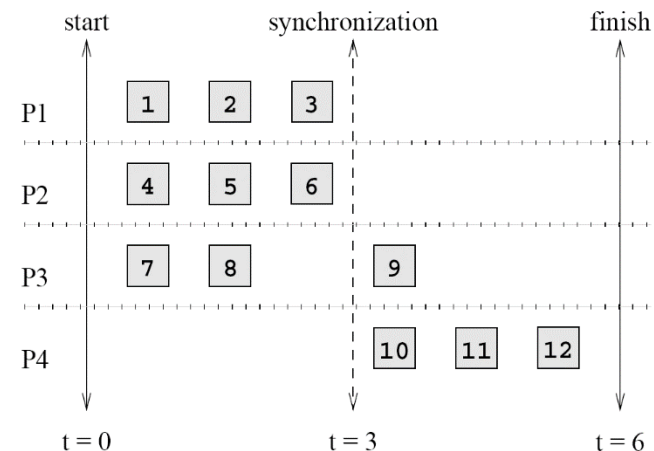
Mapeo de tareas a procesos

- Para analizar el mapeo, debemos tener en cuenta el grafo de dependencias de tareas y la interacción entre las mismas
 - Las dependencias entre las tareas puede condicionar el balance de carga entre los procesos
 - La interacción entre tareas debe tender a minimizar la comunicación entre los procesos

Las tareas 9-12 sólo pueden ejecutarse cuando 1-8 hayan sido completadas



(a)



(b)

- Una carga balanceada no necesariamente significa mínimo tiempo de ejecución

Técnicas de mapeo para balance de carga

- Las técnicas de mapeo usadas en algoritmos paralelos se pueden clasificar en **estáticas** y **dinámicas**.
- Las técnicas **estáticas**:
 - Distribuyen las tareas entre los procesos previo a la ejecución
 - Es fundamental conocer las características de las tareas (generación, tamaño, volumen de datos asociado).
 - Para casos complejos se emplean heurísticas (mapeo óptimo es NP-completo)
 - En general los algoritmos son más fáciles de diseñar y programar

Técnicas de mapeo para balance de carga

- Las técnicas ***dinámicas***:
 - Distribuyen las tareas entre los procesos durante la ejecución.
 - Si las tareas se generan dinámicamente, entonces deben mapearse dinámicamente también.
 - Si no se conoce de antemano el tamaño de las tareas, el mapeo dinámico suele dar mejor resultado
 - Si el volumen de datos asociado a cada tarea es grande pero el cómputo no es significativo, un mapeo dinámico podría incurrir en un alto overhead por la migración de datos

Técnicas de mapeo para balance de carga: esquemas de mapeo estático

- El mapeo estático suele ser utilizado en problemas que emplean descomposición basada en los datos
- Como las tareas están fuertemente relacionadas con los datos, mapear los datos a los procesos es de alguna forma equivalente a mapear las tareas a los procesos

row-wise distribution

P_0
P_1
P_2
P_3
P_4
P_5
P_6
P_7

column-wise distribution

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
-------	-------	-------	-------	-------	-------	-------	-------

Distribución por bloques (1D)

Técnicas de mapeo para balance de carga: esquemas de mapeo estático

- El mapeo estático suele ser utilizado en problemas que emplean descomposición basada en los datos
- Como las tareas están fuertemente relacionadas con los datos, mapear los datos a los procesos es de alguna forma equivalente a mapear las tareas a los procesos

P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}

(a)

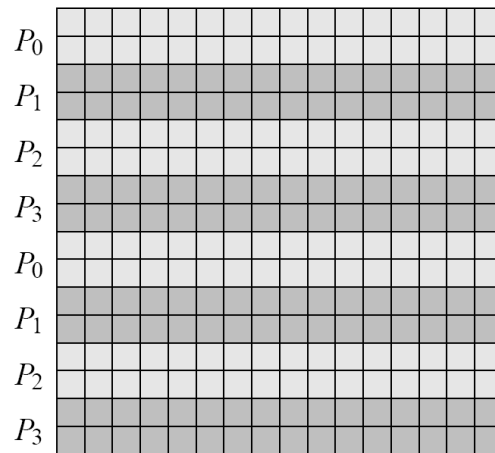
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}

(b)

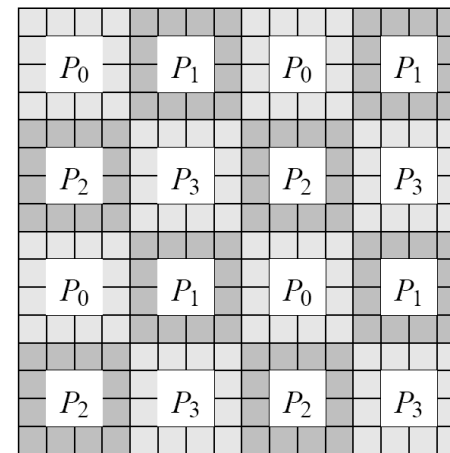
Distribución por bloques (2D)

Técnicas de mapeo para balance de carga: esquemas de mapeo estático

- El mapeo estático suele ser utilizado en problemas que emplean descomposición basada en los datos
- Como las tareas están fuertemente relacionadas con los datos, mapear los datos a los procesos es de alguna forma equivalente a mapear las tareas a los procesos



(a)



(b)

Distribución por bloques cíclicas

Técnicas de mapeo para balance de carga: esquemas de mapeo estático

- El mapeo estático suele ser utilizado en problemas que emplean descomposición basada en los datos
- Como las tareas están fuertemente relacionadas con los datos, mapear los datos a los procesos es de alguna forma equivalente a mapear las tareas a los procesos

P_0	P_1	P_2	P_3	P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7	P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}	P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}	P_{12}	P_{13}	P_{14}	P_{15}
P_0	P_1	P_2	P_3	P_0	P_1	P_2	P_3
P_4	P_5	P_6	P_7	P_4	P_5	P_6	P_7
P_8	P_9	P_{10}	P_{11}	P_8	P_9	P_{10}	P_{11}
P_{12}	P_{13}	P_{14}	P_{15}	P_{12}	P_{13}	P_{14}	P_{15}

Distribución por bloques aleatorios

Técnicas de mapeo para balance de carga: esquemas de mapeo dinámico

- El mapeo dinámico resulta necesario cuando:
 - emplear un mapeo estático puede llevar a una distribución desbalanceada de trabajo entre los procesos
 - el grafo de dependencias de tareas es dinámico en sí mismo
- Se suele referir a sus técnicas como *balance de carga dinámico*, ya que equilibrar la carga entre procesos es la principal razón de su uso.
- Los esquemas de mapeo dinámico se clasifican en **centralizados** o **distribuidos**.

Técnicas de mapeo para balance de carga: esquemas de mapeo dinámico

- En los esquemas **centralizados**:
 - Existe un proceso especial llamado *maestro* (o *master*) que administra las tareas a realizar; al resto de los procesos se los denomina *esclavos* (o *workers*).
 - Cuando un proceso *worker* no tiene trabajo, le pide al *master* que le asigne una tarea y así sucesivamente hasta que no queden tareas por completar.
 - Suele ser más fácil de implementar que los esquemas distribuidos pero sufren de escalabilidad limitada → el *master* se puede volver un cuello de botella cuando la cantidad de procesos es muy grande.

Técnicas de mapeo para balance de carga: esquemas de mapeo dinámico

- En los esquemas ***distribuidos***:
 - Se evita el cuello de botella potencial del master, delegando la distribución entre varios procesos *pares*.
 - Más difícil de implementar
 - Los problemas que surgen son de sincronización:
 - ¿Qué proceso inicia la distribución de carga?
 - ¿Cómo se comunican para decidir qué proceso transfiere a otro?
 - ¿Cuándo se asigna una tarea a un proceso?
 - ¿Cómo se mantiene un control distribuido de las tareas ya realizadas?
 - Entre otros

MÉTODOS PARA REDUCIR OVERHEAD DE LAS INTERACCIONES

Overhead de las interacciones

- Reducir el overhead asociado a las interacciones entre procesos es un factor clave para mejorar la eficiencia de los programas paralelos.
- Existen diferentes métodos:
 - Minimizar volumen de datos intercambiados → A mayor volumen de datos intercambiados, mayor tiempo de comunicación.
 - Minimizar frecuencia de las interacciones → Cada interacción tiene un costo inicial de preparación. Siempre que sea posible, conviene combinar varias comunicaciones en una sola.
 - Minimizar competencia entre recursos y *zonas críticas (hotspots)*: Evitar posibles cuellos de botella mediante el uso de técnica descentralizadas. Replicar datos si es necesario.

Métodos para reducir overhead de las interacciones

- Existen diferentes métodos:
 - Solapar cómputo con comunicaciones: mediante el uso de operaciones no bloqueantes en pasaje de mensajes y técnicas de multi-hilado y prebúsqueda en memoria compartida.
 - Replicar datos o cómputo: si permite reducir las interacciones (mensajes o sincronización)
 - Usar operaciones de comunicación colectiva
 - Solapar comunicaciones con otras comunicaciones: siempre y cuando el hardware de soporte lo permita, solapar diferentes comunicaciones puede reducir el overhead

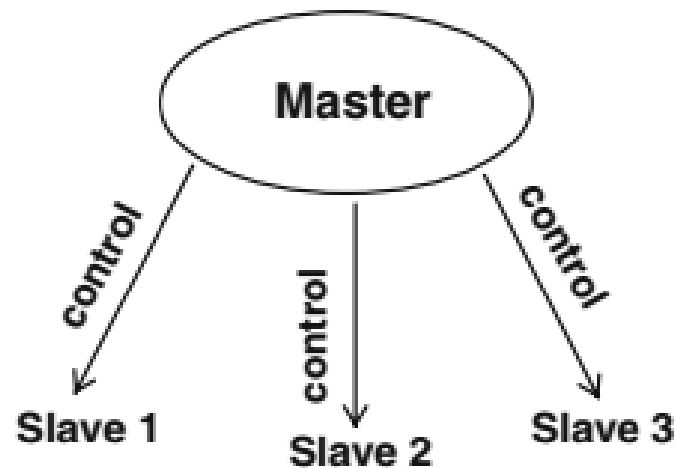
MODELOS DE ALGORITMOS PARALELOS

Modelos de algoritmos paralelos

- Un modelo de algoritmo representa una estructura usual de código que combina técnicas de descomposición de problema y de mapeo de tareas junto a la aplicación de métodos para minimizar overhead.
- Existen diferentes modelos. Entre los más comunes:
 - Maestro-Esclavo
 - Pipeline
 - Single Program Multiple Data (SPMD)
 - Divide y vencerás

Modelo Maestro-Esclavo

- También conocido como *Master-Slave*, *Master-Worker* o *Manager-Worker*.
- El proceso Maestro es el responsable de generar trabajo y asignárselo a los Workers.



Modelo Maestro-Esclavo

- Dos opciones para la distribución de trabajo:
 - Si el Maestro puede estimar de antemano el tamaño de las tareas, un mapeo estático será una buena opción. Ejemplo: multiplicación de matrices.
 - En otro caso, el mapeo dinámico es la opción elegida → tareas pequeñas son asignadas a los workers (posiblemente) en múltiples instancias. Ejemplo: ordenación de un vector
- Se debe tener en cuenta que el Master puede convertirse en un *cuello de botella*, si las tareas son muy pequeñas o los Workers son muy rápidos → la granularidad de las tareas debe ser elegida de forma tal que el tiempo de procesar la tarea sea mucho mayor que su comunicación o sincronización asociada.
- Puede ser generalizado a múltiples niveles.
- Resulta adecuado tanto para memoria compartida como para pasaje de mensajes.

Modelo Pipeline

- El cómputo se descompone en una secuencia de procesos.
- Los datos suelen ser particionados y *pasados* entre los procesos, donde cada uno realiza una tarea sobre ellos.
- Usualmente se organizan en forma de arreglo lineal o multi-dimensional.



- Organizaciones menos comunes incluyen árboles o grafos.

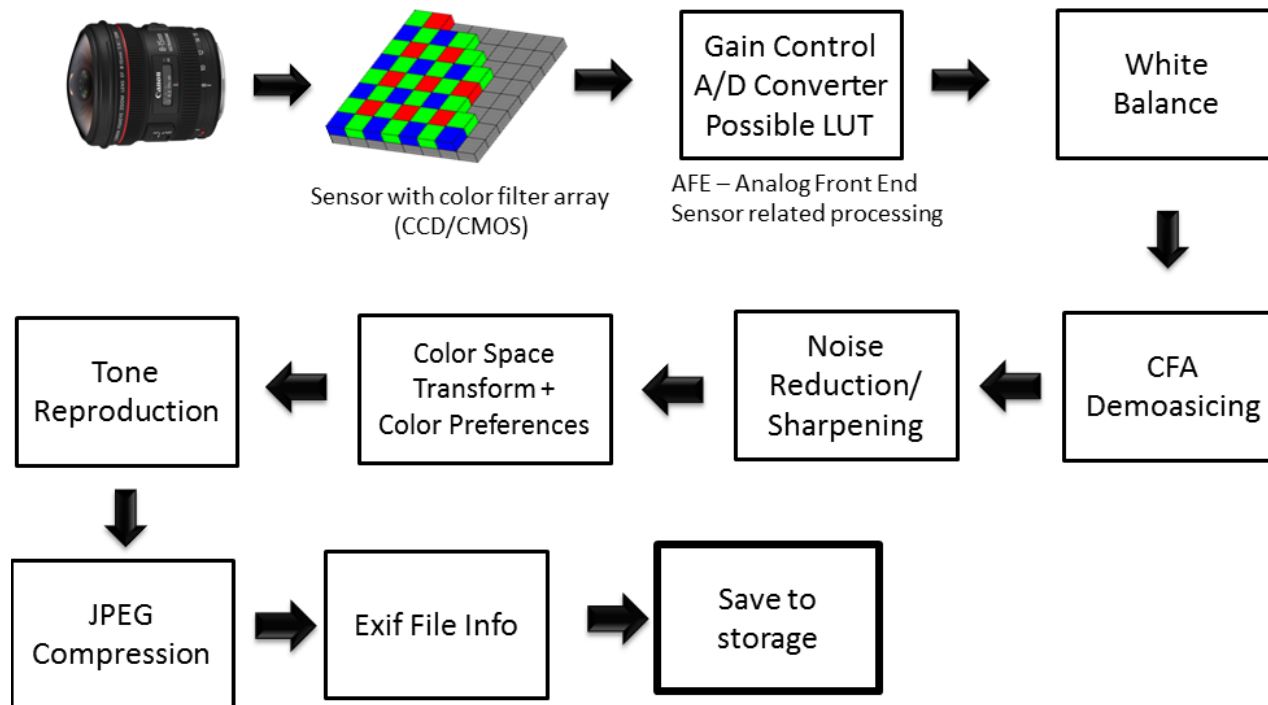
Modelo Pipeline

- Un pipeline puede ser visto como una cadena de productores y consumidores
 - Cada proceso consume los datos que genera el anterior en el pipe
 - Al mismo tiempo, produce los datos que serán consumidos por el siguiente proceso
- El balance de carga depende de la granularidad de las tareas
 - A mayor granularidad, más tiempo tardará el pipeline en llenarse (paralelismo ideal)
 - A menor granularidad, mayor interacción entre los procesos del pipeline

Modelo Pipeline

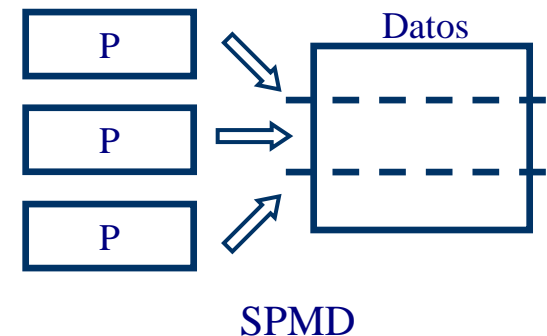
- Usado generalmente en el procesamiento de imágenes, donde se aplican diferentes filtros u operaciones sobre una o más imágenes

Pipeline for sRGB (JPEG)



Modelo Single Program Multiple Data (SPMD)

- Cada proceso realiza el *mismo* cómputo sobre una porción de datos diferentes
 - Mediante sentencias condicionales es posible que los procesos tomen diferentes caminos
- En general, la carga de trabajo es proporcional a la cantidad de datos asignados a un proceso
 - Dificultades en problemas irregulares o donde la arquitectura de soporte es heterogénea
- El cómputo puede involucrar diferentes fases, las cuales son usualmente intercaladas con comunicación/sincronización

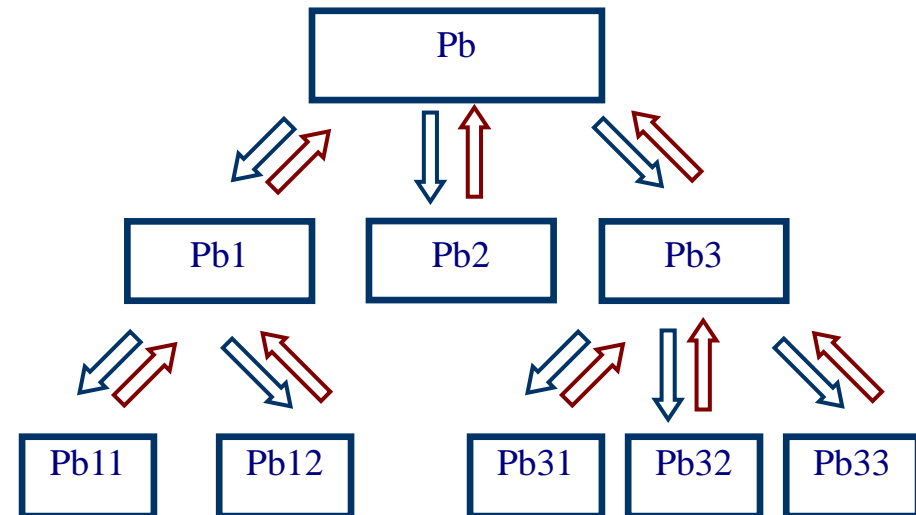


Modelo Single Program Multiple Data (SPMD)

- Resulta adecuado tanto en memoria compartida como en pasaje de mensajes
- En memoria compartida, el esfuerzo de programación suele ser menor
- En pasaje de mensajes:
 - Cuando el espacio de direcciones está particionado, usualmente se tiene un mayor control sobre la ubicación de los datos → mayor localidad de datos
 - El overhead de las comunicaciones puede ser aliviado mediante el uso de operaciones no bloqueantes, siempre y cuando las dependencias lo permitan

Modelo Divide y Vencerás

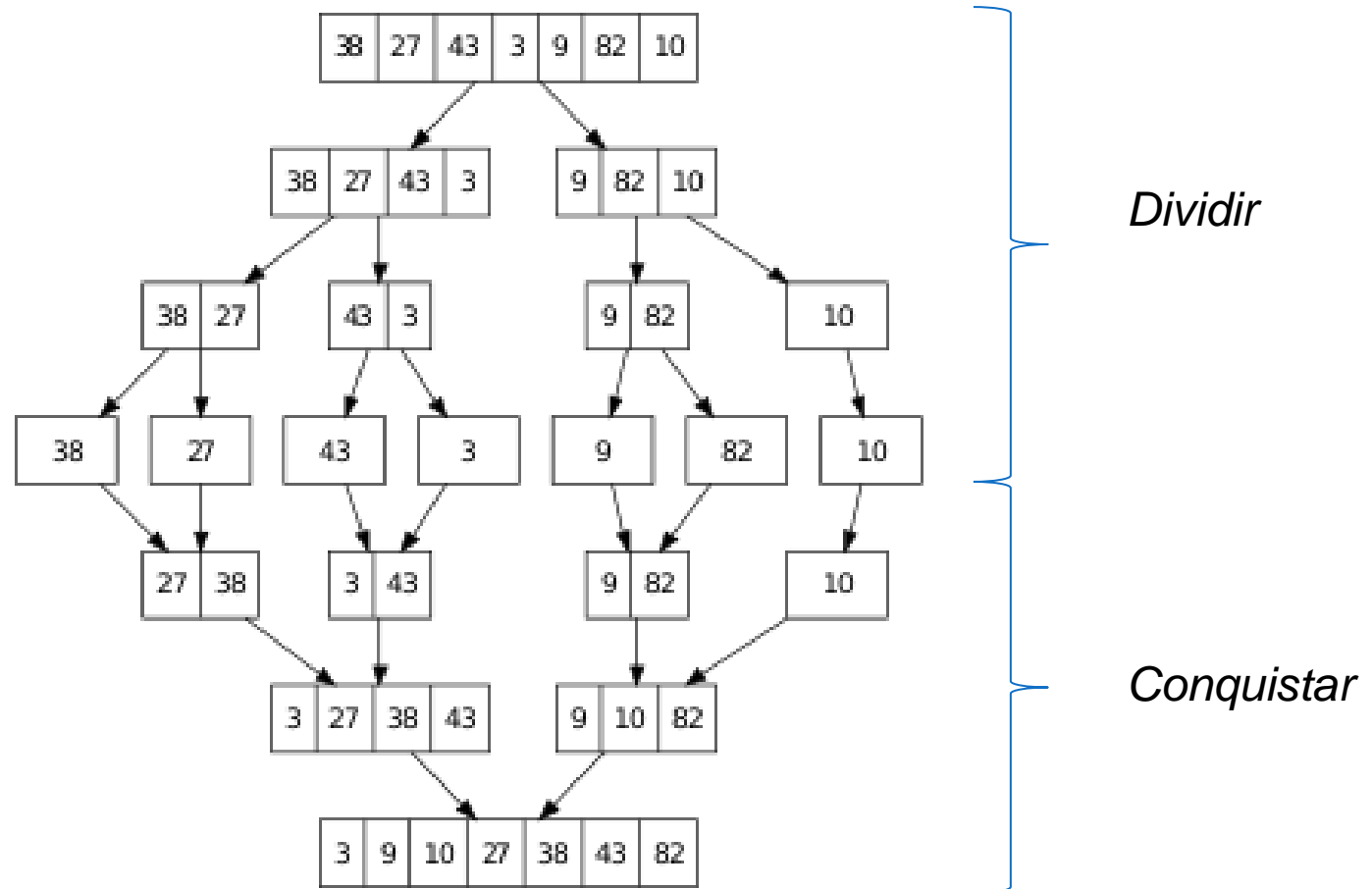
- También llamado Dividir y conquistar
- *Dividir*: fase en la que se particiona sucesivamente el problema en sub-problemas más pequeños hasta obtener una granularidad deseada
- *Conquistar*: fase en la que se resuelven los subproblemas en forma independiente.
- En ocasiones, se requiere una fase adicional de combinación de resultados parciales para llegar al resultado final.



Divide and Conquer

Modelo Divide y Vencerás

- Ejemplo: *Merge-sort*



Bibliografía usada para esta clase

- Capítulo 3, An Introduction to Parallel Computing. Design and Analysis of Algorithms (2da Edition). Grama A., Gupta A., Karypis G. & Kumar V. (2003) Inglaterra: Pearson Addison-Wesley.
- Capítulo 2, Designing and Building Parallel Programs. Foster I. (1996) EEUU: Addison-Wesley