Sistemas Distribuidos y Paralelos

Ingeniería en Computación

Diseño de algoritmos paralelos

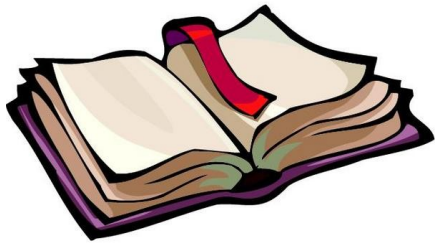
Universidad Nacional de La Plata

Facultad de Informática

Prof. Dr. Adrian Pousa

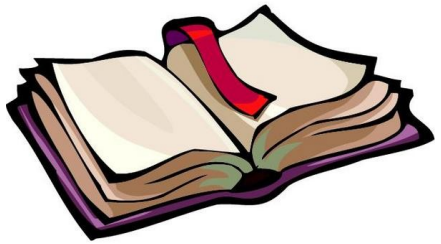
Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



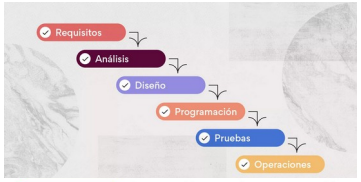
Agenda

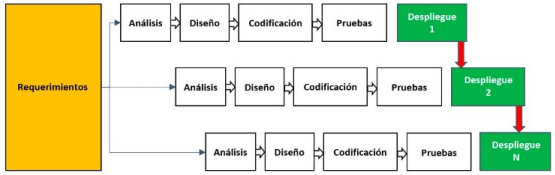
I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación

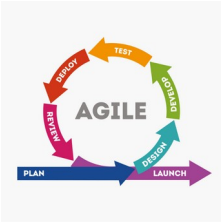


Introducción al diseño de algoritmos paralelos

- En el área de la **Ingeniería de Software** se proponen varios **modelos para desarrollo de software:**
  - Cascada
  - Evolutivo – En espiral
  - Metodologías ágiles
  - Otros...







- Cualquiera sea el modelo elegido, la ingeniería de software nos destaca la importancia de realizar un **buen diseño** antes de llevar a cabo cualquier implementación.

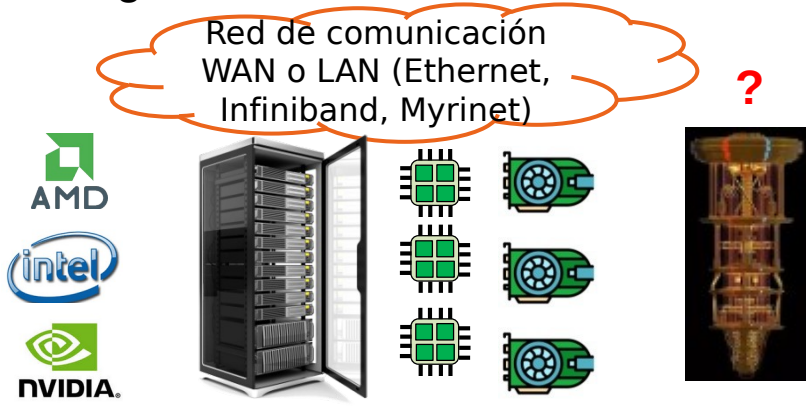
- El desarrollo de algoritmos paralelos no es la excepción al desarrollo tradicional de software, aunque tiene características muy específicas.
  - En particular, no nos podemos abstraer demasiado del hardware a utilizar, ya que es importante obtener el mayor beneficio de este para mejorar el rendimiento.

Introducción al diseño de algoritmos paralelos

¿Como paralelizamos un algoritmo secuencial?

- Diseñar algoritmos paralelos no es fácil y lleva mucho esfuerzo.
- Es un proceso altamente "creativo" que, en general, consiste en distribuir el trabajo del algoritmo secuencial entre procesos o hilos, ejecutados sobre hardware paralelo, que eventualmente deben comunicarse y sincronizarse.

- La diversidad de arquitecturas paralelas actuales presenta varios desafíos.
- Un buen diseño dependerá de una combinación de estrategias en función de la arquitectura.
  - **Ventaja:** flexibilidad para lograr mayor paralelismo
  - **Desventaja:** dificulta la etapa de diseño.



Introducción al diseño de algoritmos paralelos

- Durante el diseño es recomendable abstraerse de aspectos de implementación. Sin embargo, deberían tenerse en cuenta varios aspectos en diferentes niveles para orientar el diseño hacia nuestra infraestructura:

Sistema Operativo

Linux
FreeBSD
Windows
...

Lenguaje de programación

C/C++
Fortran
...

Modelo de programación

Memoria compartida:
- OpenMP
- Pthreads
- Cuda
- ...

Memoria distribuida:
- MPI
- ...

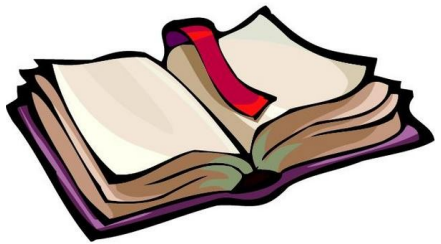
Híbridos:
- MPI-OpenMP
- MPI-Pthreads
- MPI-OpenMP-CUDA
- ...

Bibliotecas

Blas
Atlas
...

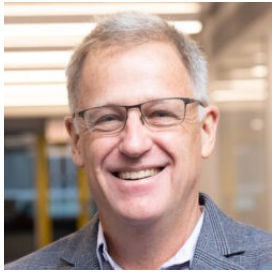
Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Etapas de diseño

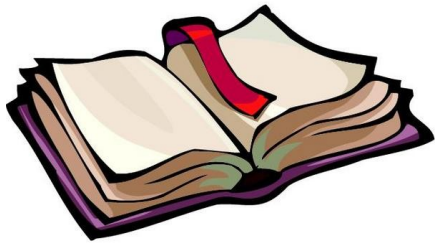
- No existe un mecanismo estandarizado que se pueda seguir
- Ian Foster[1] propuso una serie de pasos, en particular para algoritmos ejecutados sobre arquitecturas de memoria distribuida (clusters):
  - **Descomposición o Particionado:** Identificar el trabajo que puede hacerse en paralelo dividiéndolo en tareas
  - **Comunicación:** Determinar los patrones de comunicación (y/o sincronización) entre las tareas identificadas en el paso previo
  - **Aglomeración:** las tareas y las comunicaciones identificadas en pasos anteriores se combinan en tareas de mayor tamaño para adaptarlas al hardware paralelo disponible
  - **Mapeo:** Asignar cada tarea a una unidad de procesamiento para maximizar la utilización del mismo y reducir la comunicación



[1]Libro: Designing and Building Parallel Programs - 2003

Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Descomposición

- Para desarrollar un algoritmo paralelo el primer paso es descomponer el problema en sus partes concurrentes (**tareas**).
- Se trata de definir un gran número de pequeñas tareas (descomposición de grano fino), para brindar la mayor flexibilidad a los “potenciales” algoritmos paralelos
- Se ignoran cuestiones como el número de unidades de procesamiento y aspectos específico en la máquina de destino.
- La atención se centra en reconocer oportunidades de ejecución paralela.

En esta etapa:

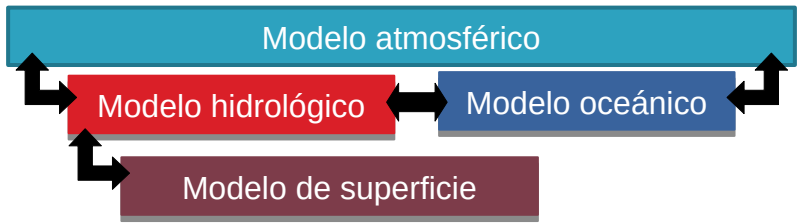
Las tareas **NO** son procesos o hilos

Tareas = cosas para hacer

Tipos de Descomposición

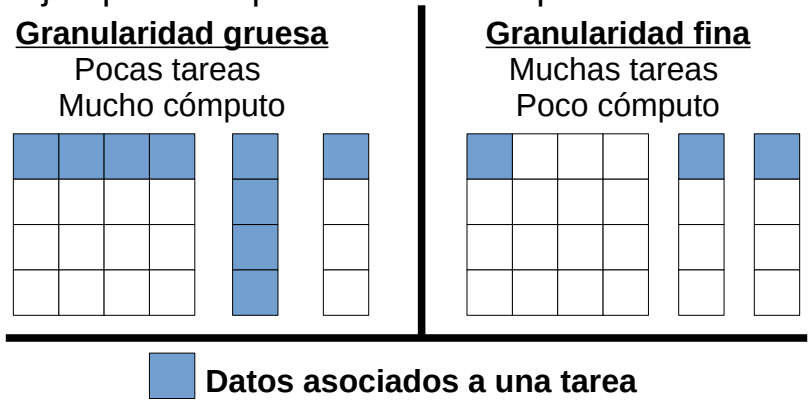
Descomposición Funcional

- Se enfoca principalmente en la funcionalidad.
- Divide el cómputo y luego asocia los datos.
- Generalmente tareas disjuntas (código diferente).
- Reduce la complejidad del diseño general: Modelos computacionales complejos pueden estructurarse como conjuntos de modelos más simples interconectados.
- Habituales en simulación:



Descomposición de datos o de dominio

- Se enfoca principalmente en los datos.
- Divide los datos y luego asocia el cómputo a tareas.
- Generalmente tareas de código igual o similar.
- Granularidad: volumen de trabajo de una tarea
- Ejemplo: Multiplicar una matriz por un vector



Descomposición Mixta

Se aplica una descomposición funcional macro con una descomposición de datos por cada función.

Descomposición
Dependencia entre tareas

- La descomposición puede generar dependencias entre las tareas que se representan mediante un grafo acíclico dirigido:
  - Los nodos corresponden a tareas. Pueden tener asociados un peso relacionado con el costo de resolver esa tarea (Por ejemplo: tiempo)
  - Las aristas/arcos indican dependencia de datos, relación de comunicación o sincronización.

- Cuando las tareas tratan con conjuntos de datos independientes, hay poca o ninguna dependencia y no hay necesidad de comunicación de datos o resultados, el problema se conoce como **Embarrassingly parallel**. La solución es paralela por naturaleza.

Descomposición
Dependencia entre tareas

- Un mismo problema puede descomponerse de diferentes maneras y generar grafos distintos.

- El grafo mas adecuado dependerá de características como:

Grado máximo de concurrencia

Máximo número de tareas ejecutadas en paralelo.

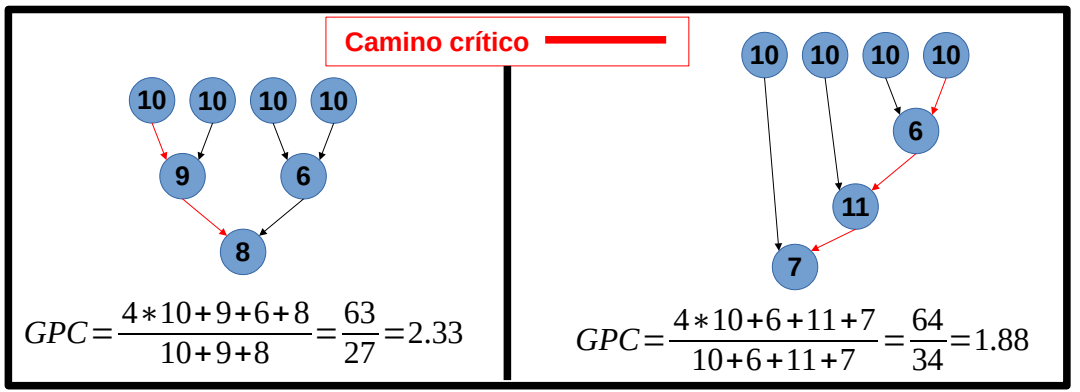
Camino crítico

Camino dirigido más largo entre un nodo inicial (que no recibe aristas) y un nodo final (del que no salen aristas).
Un camino crítico más corto favorece a un mayor grado de concurrencia.

Grado promedio de concurrencia (GPC)

Número promedio de tareas ejecutadas en paralelo.

$$GPC = \frac{Peso\ total}{Longitud\ del\ camino\ crítico}$$



Descomposición
Características de las tareas

Generación
¿Cómo y cuantas se generan?

Estáticas:

Se pueden identificar a priori

Dinámicas:

No se conocen a priori.

El volumen de trabajo no depende del tamaño de los datos sino de características propias de los mismos.

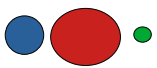
Comportamiento

Uniforme:

Tareas iguales

No uniforme:

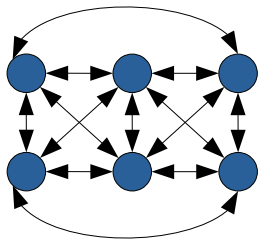
Tareas diferentes



Volumen asociado por tarea
(Depende de la granularidad)

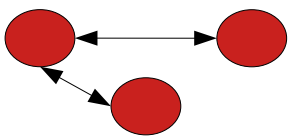
Granularidad fina:

Poco cómputo por tarea
Mucha dependencia
Mucha comunicación



Granularidad gruesa:

Mucho cómputo por tarea
Poca dependencia
Poca comunicación



Descomposición

¿Como descomponer un problema en tareas?

- Ananth Grama[1] propone algunas estrategias:
  - Descomposición recursiva
  - Descomposición basada en los datos
  - Descomposición exploratoria
  - Descomposición especulativa
  - Descomposición híbrida

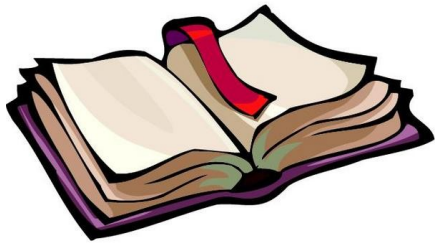
Un mismo problema puede aceptar más de una estrategia de descomposición.

Sin embargo, algunas pueden ser mas adecuadas que otras.

[1]Libro: An Introduction to Parallel Computing - 2003

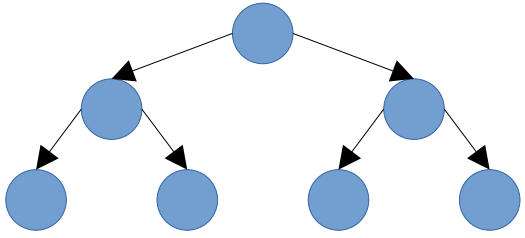
Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación

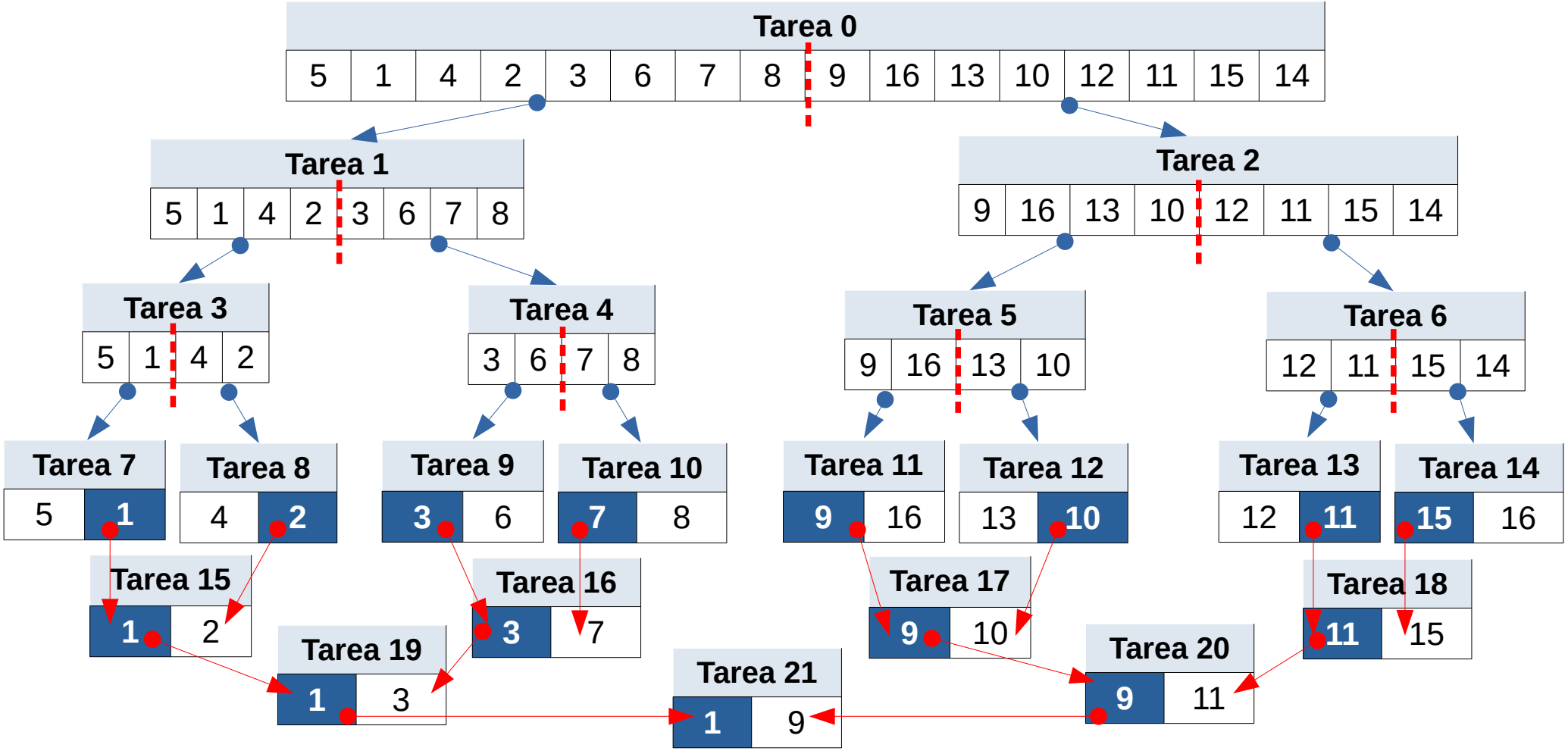


Descomposición recursiva

- La descomposición recursiva se utiliza en problemas que siguen una estrategia “divide y vencerás”.
- El problema se descompone en un par de subproblemas idénticos independientes.
- Recursivamente, cada uno de los subproblemas se vuelve a subdividir de a pares hasta alcanzar una cierta granularidad.
- Finalmente se combinan los resultados.
- Algunos ejemplos:
  - Ordenación (Quicksort, Mergesort)
  - Reducción (dado N valores aplicar una función para obtener un único valor):
    - Búsqueda del valor máximo o mínimo en una lista
    - Sumar los elementos de una lista
    - Factorial

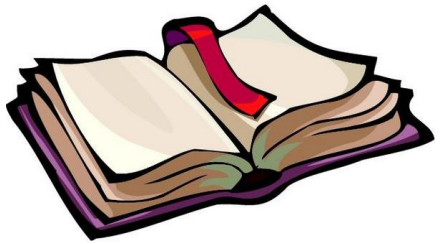


Descomposición recursiva
Búsqueda del mínimo en un arreglo



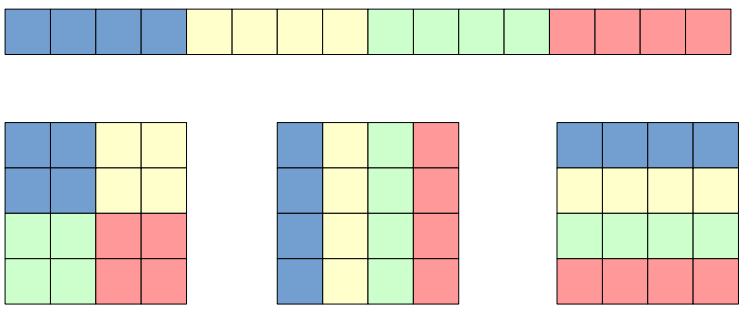
Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



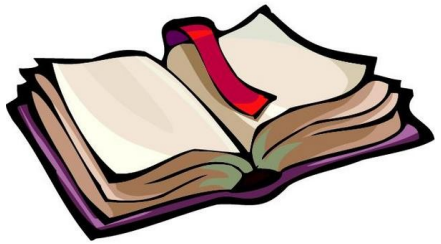
Descomposición basada en datos

- La descomposición basada en datos identifica el conjunto de datos sobre los cuales se realizarán los cálculos.
- Luego divide los datos entre tareas.
- Existen varias alternativas:
  - Descomposición basada en datos de **salida**
  - Descomposición basada en datos de **entrada**
  - Descomposición basada en datos de **entrada y salida**
  - Descomposición basada en datos **intermedios**



Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



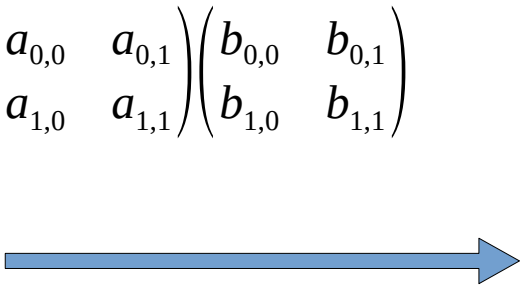
Descomposición basada en datos de salida

- Si los cálculos de los elementos de salida (conjunto de resultados) de un programa paralelo son independientes, la descomposición natural es una tarea por resultado.

- **Por ejemplo:** La **multiplicación de** dos **matrices** cuadradas de $2x2$ elementos $C=AB$.
  - Dado que el cálculo de cada elemento de la matriz resultado C es independiente del cálculo de los restantes elementos, una posible **descomposición basada en datos de salida** genera cuatro tareas **independientes**:

$$\begin{pmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \end{pmatrix} = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \begin{pmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{pmatrix}$$

$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

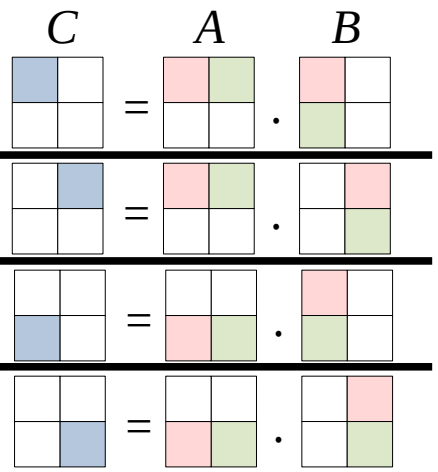


$Tarea_0 \rightarrow c_{0,0} = a_{0,0} \cdot b_{0,0} + a_{0,1} \cdot b_{1,0}$

$Tarea_1 \rightarrow c_{0,1} = a_{0,0} \cdot b_{0,1} + a_{0,1} \cdot b_{1,1}$

$Tarea_2 \rightarrow c_{1,0} = a_{1,0} \cdot b_{0,0} + a_{1,1} \cdot b_{1,0}$

$Tarea_3 \rightarrow c_{1,1} = a_{1,0} \cdot b_{0,1} + a_{1,1} \cdot b_{1,1}$



Embarrassingly parallel !!!

Descomposición basada en datos de salida

- Sin embargo, la descomposición anterior no es única:

Descomposición 1

$$Tarea_0 \rightarrow c_{0,0} = a_{0,0} \cdot b_{0,0} + a_{0,1} \cdot b_{1,0}$$
$$Tarea_1 \rightarrow c_{0,1} = a_{0,0} \cdot b_{0,1} + a_{0,1} \cdot b_{1,1}$$
$$Tarea_2 \rightarrow c_{1,0} = a_{1,0} \cdot b_{0,0} + a_{1,1} \cdot b_{1,0}$$
$$Tarea_3 \rightarrow c_{1,1} = a_{1,0} \cdot b_{0,1} + a_{1,1} \cdot b_{1,1}$$

Descomposición 2

$$Tarea_0 \rightarrow c_{0,0} = a_{0,0} \cdot b_{0,0}$$
$$Tarea_1 \rightarrow c_{0,0} = c_{0,0} + a_{0,1} \cdot b_{1,0}$$
$$Tarea_2 \rightarrow c_{0,1} = a_{0,0} \cdot b_{0,1}$$
$$Tarea_3 \rightarrow c_{0,1} = c_{0,1} + a_{0,1} \cdot b_{1,1}$$
$$Tarea_4 \rightarrow c_{1,0} = a_{1,0} \cdot b_{0,0}$$
$$Tarea_5 \rightarrow c_{1,0} = c_{1,0} + a_{1,1} \cdot b_{1,0}$$
$$Tarea_6 \rightarrow c_{1,1} = a_{1,0} \cdot b_{0,1}$$
$$Tarea_7 \rightarrow c_{1,1} = c_{1,1} + a_{1,1} \cdot b_{1,1}$$

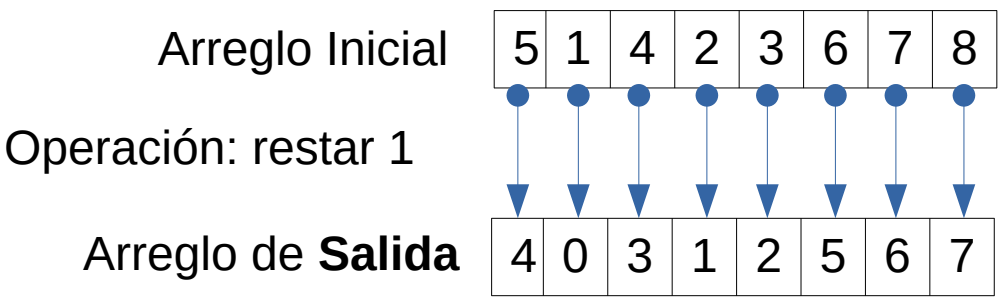
Descomposición 3

$$Tarea_0 \rightarrow c_{0,0} = a_{0,1} \cdot b_{1,0}$$
$$Tarea_1 \rightarrow c_{0,0} = c_{0,0} + a_{0,0} \cdot b_{0,0}$$
$$Tarea_2 \rightarrow c_{0,1} = a_{0,1} \cdot b_{1,1}$$
$$Tarea_3 \rightarrow c_{0,1} = c_{0,1} + a_{0,0} \cdot b_{0,1}$$
$$Tarea_4 \rightarrow c_{1,0} = a_{1,1} \cdot b_{1,0}$$
$$Tarea_5 \rightarrow c_{1,0} = c_{1,0} + a_{1,0} \cdot b_{0,0}$$
$$Tarea_6 \rightarrow c_{1,1} = a_{1,1} \cdot b_{1,1}$$
$$Tarea_7 \rightarrow c_{1,1} = c_{1,1} + a_{1,0} \cdot b_{0,1}$$

Descomposición basada en datos de salida
Función de Mapeo

- Otro ejemplo donde se puede aplicar este tipo de descomposición es la función de **Mapeo** (MAP), que consiste en aplicar la misma operación a cada elemento de una estructura de datos.

- **Por ejemplo:** Decrementar el valor de los elementos de un arreglo. El cálculo de cada celda del arreglo de salida es independiente del cálculo de los restantes elementos. Una posible descomposición basada en **datos de salida**:

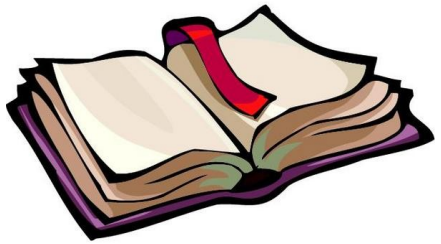


$$Tarea_0 \rightarrow \text{Arreglo de Salida}[0] = \text{Arreglo Inicial}[0] - 1$$
$$Tarea_1 \rightarrow \text{Arreglo de Salida}[1] = \text{Arreglo Inicial}[1] - 1$$
$$Tarea_2 \rightarrow \text{Arreglo de Salida}[2] = \text{Arreglo Inicial}[2] - 1$$
$$Tarea_3 \rightarrow \text{Arreglo de Salida}[3] = \text{Arreglo Inicial}[3] - 1$$
$$Tarea_4 \rightarrow \text{Arreglo de Salida}[4] = \text{Arreglo Inicial}[4] - 1$$
$$Tarea_5 \rightarrow \text{Arreglo de Salida}[5] = \text{Arreglo Inicial}[5] - 1$$
$$Tarea_6 \rightarrow \text{Arreglo de Salida}[6] = \text{Arreglo Inicial}[6] - 1$$
$$Tarea_7 \rightarrow \text{Arreglo de Salida}[7] = \text{Arreglo Inicial}[7] - 1$$

Embarrassingly parallel !!!

Agenda

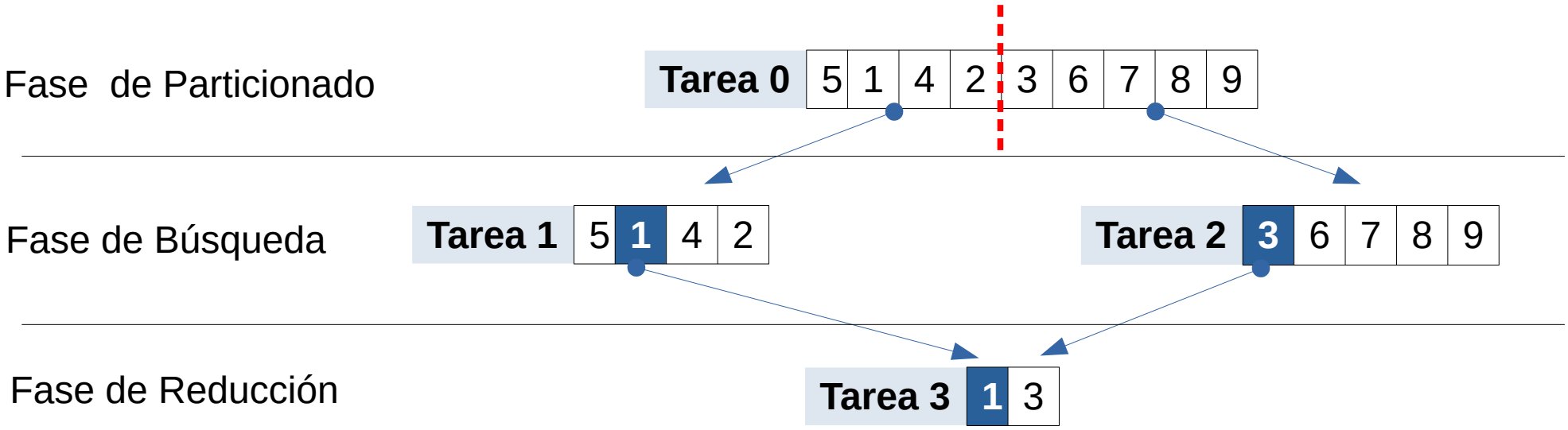
I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



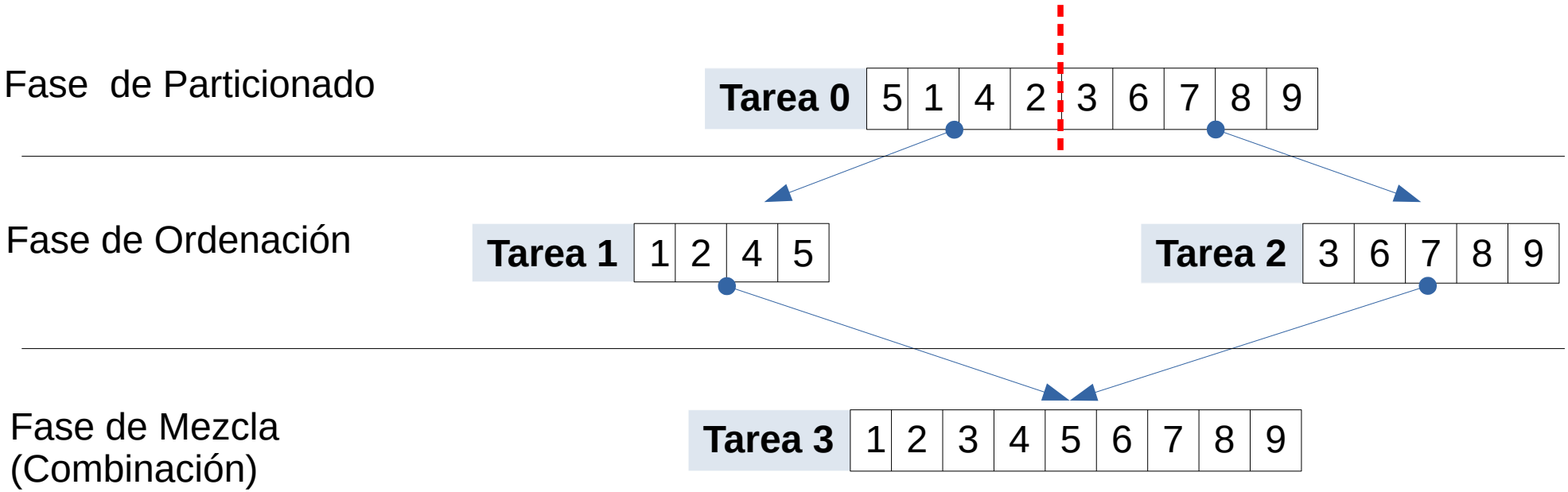
Descomposición basada en datos de entrada

- Es la descomposición basada en datos más común.
- Cada **tarea** se asocia con una **partición** de los **datos de entrada**.
- Se utiliza cuando no se conocen a priori cuáles o cuántos son los resultados:
  - Búsqueda del valor máximo o mínimo en una lista
  - Ordenación (Quicksort, Mergesort)
  - Procesamiento de imágenes: Buscar un patrón, analizar similitudes en imágenes, aplicar filtros etc.
  - Función de mapeo
  - Otros
- Generalmente, se requiere un **paso** posterior para **recuperar los resultados**.

Descomposición basada en datos de entrada
Búsqueda del mínimo en un arreglo

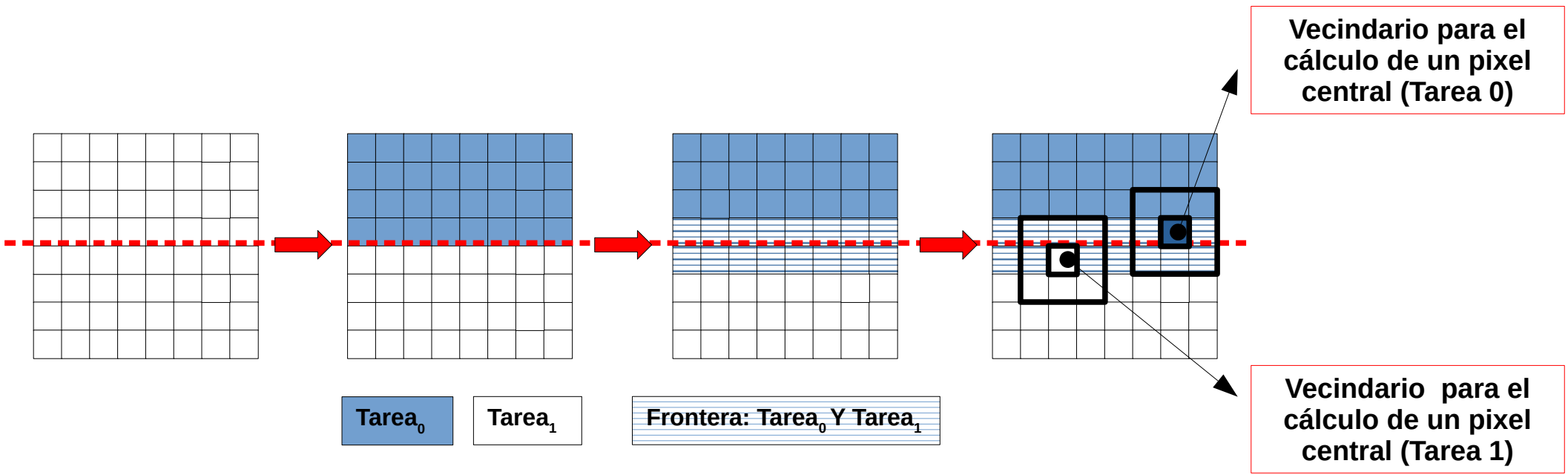


Descomposición basada en datos de entrada
Ordenación por mezcla de un arreglo



Descomposición basada en datos de entrada Problema de la frontera

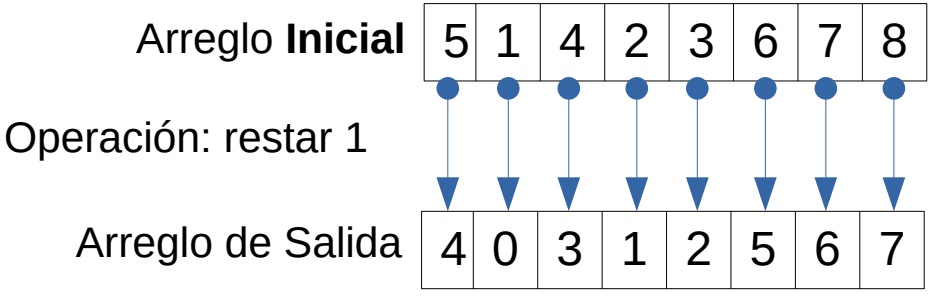
- Un caso particular en la descomposición basada en datos de entrada es el **problema de la frontera. Las tareas comparten áreas de datos**
- Caso típico de los algoritmos de **procesamiento de imágenes** donde aplicar una operación a un pixel requiere información de los píxeles vecinos.



Descomposición basada en datos de entrada Función de Mapeo

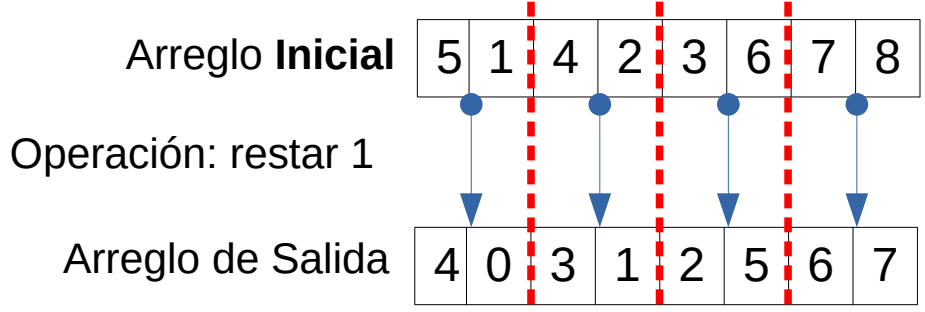
- La función de **Mapeo** también puede descomponerse a partir de los **datos de entrada**.

Descomposición 1



$Tarea_0 \rightarrow \text{Arreglo Inicial}[0]-1$
$Tarea_1 \rightarrow \text{Arreglo Inicial}[1]-1$
$Tarea_2 \rightarrow \text{Arreglo Inicial}[2]-1$
$Tarea_3 \rightarrow \text{Arreglo Inicial}[3]-1$
$Tarea_4 \rightarrow \text{Arreglo Inicial}[4]-1$
$Tarea_5 \rightarrow \text{Arreglo Inicial}[5]-1$
$Tarea_6 \rightarrow \text{Arreglo Inicial}[6]-1$
$Tarea_7 \rightarrow \text{Arreglo Inicial}[7]-1$

Descomposición 2

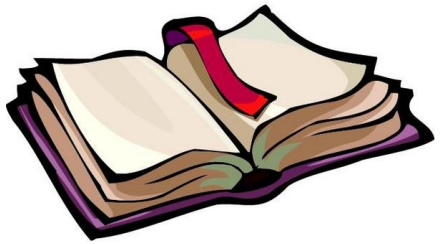


$Tarea_0 \rightarrow \text{Arreglo Inicial}[0]-1$
$Tarea_0 \rightarrow \text{Arreglo Inicial}[1]-1$
$Tarea_1 \rightarrow \text{Arreglo Inicial}[2]-1$
$Tarea_1 \rightarrow \text{Arreglo Inicial}[3]-1$
$Tarea_2 \rightarrow \text{Arreglo Inicial}[4]-1$
$Tarea_2 \rightarrow \text{Arreglo Inicial}[5]-1$
$Tarea_3 \rightarrow \text{Arreglo Inicial}[6]-1$
$Tarea_3 \rightarrow \text{Arreglo Inicial}[7]-1$

Embarrassingly parallel !!!

Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Descomposición basada en datos de entrada/salida

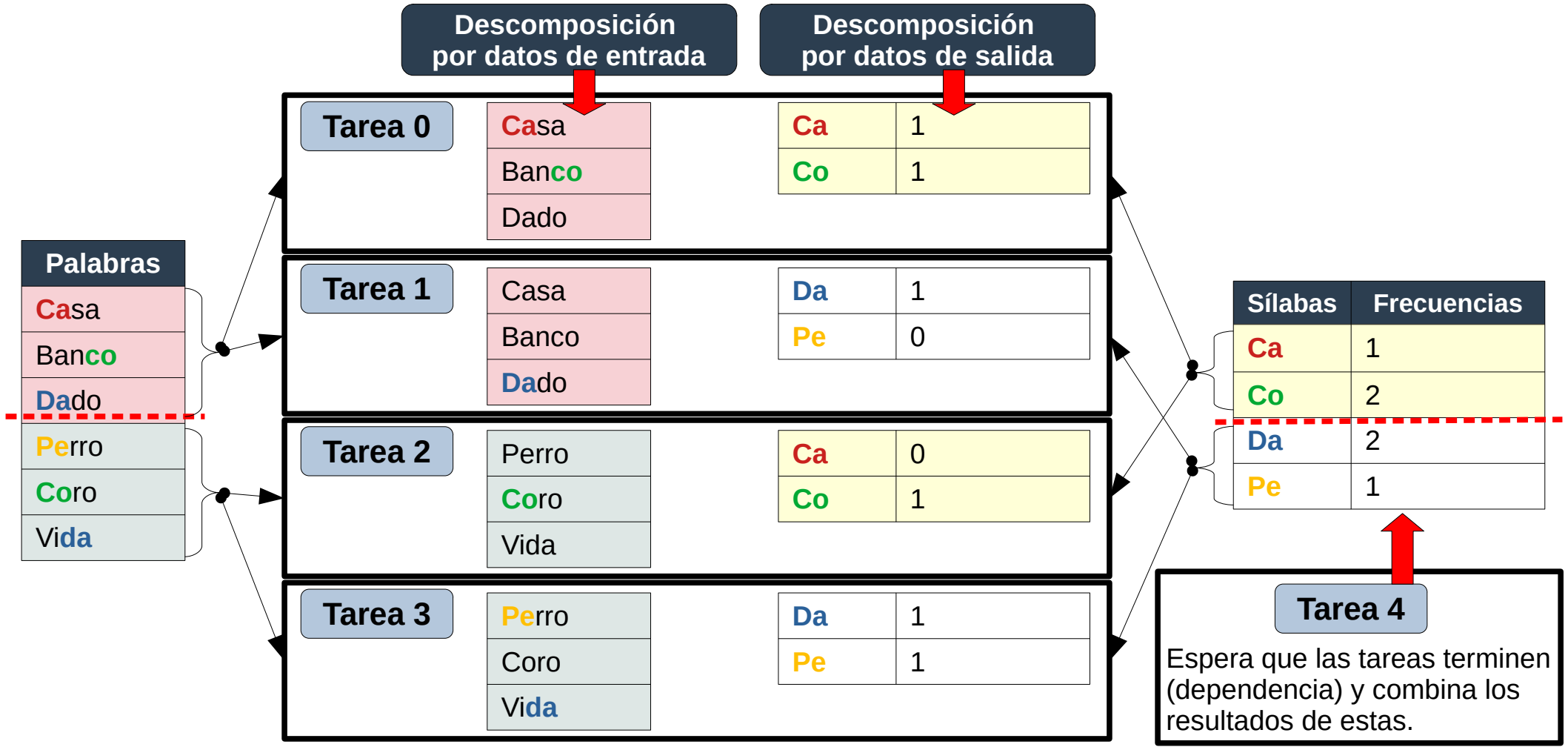
- Es posible combinar la estrategia basada en datos de entrada con la de salida.
- **Por ejemplo:** buscar la frecuencia con que se repiten ciertas sílabas en un listado de palabras (no importa mayúsculas y minúsculas):
  - **Entrada:** una lista de palabras.
  - **Salida:** un listado de sílabas con sus frecuencias.

Palabras
Casa
Banco
Dado
Perro
Coro
Vida

Sílabas	Frecuencias
Ca	1
Co	2
Da	2
Pe	1

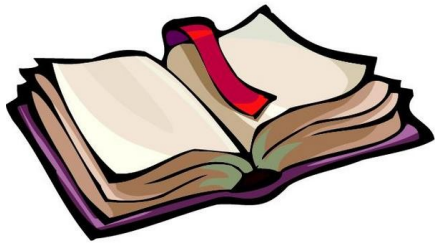
- Cada tarea consiste en buscar las frecuencias de un subconjunto de sílabas (conjunto de salida) en un subconjunto de palabras (conjunto de entrada).

Descomposición basada en datos de entrada/salida



Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Descomposición basada en datos intermedios

- Los algoritmos requieren una sucesión de transformaciones sobre los datos.
- El problema se descompone en **etapas** sucesivas que a su vez generan dependencias entre tareas.

$$\begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix}\begin{pmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{pmatrix}=\begin{pmatrix} C1_{0,0} & C1_{0,1} \\ C1_{1,0} & C1_{1,1} \end{pmatrix}+\begin{pmatrix} C2_{0,0} & C2_{0,1} \\ C2_{1,0} & C2_{1,1} \end{pmatrix}$$

Etapa 1

$Tarea_0 \rightarrow C1_{0,0}=a_{0,0}\cdot b_{0,0}$
$Tarea_1 \rightarrow C2_{0,0}=a_{0,1}\cdot b_{1,0}$
$Tarea_2 \rightarrow C1_{0,1}=a_{0,0}\cdot b_{0,1}$
$Tarea_3 \rightarrow C2_{0,1}=a_{0,1}\cdot b_{1,1}$
$Tarea_4 \rightarrow C1_{1,0}=a_{1,0}\cdot b_{0,0}$
$Tarea_5 \rightarrow C2_{1,0}=a_{1,1}\cdot b_{1,0}$
$Tarea_6 \rightarrow C1_{1,1}=a_{1,0}\cdot b_{0,1}$
$Tarea_7 \rightarrow C2_{1,1}=a_{1,1}\cdot b_{1,1}$

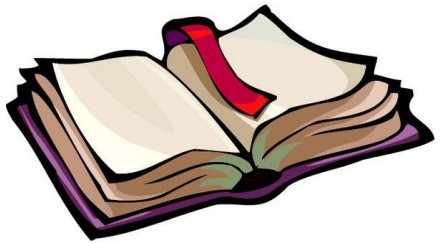
Dependencias

Etapa 2

$Tarea_8 \rightarrow c_{0,0}=C1_{0,0}+C2_{0,0}$
$Tarea_9 \rightarrow c_{0,1}=C1_{0,1}+C2_{0,1}$
$Tarea_{10} \rightarrow c_{1,0}=C1_{1,0}+a_{1,0}$
$Tarea_{11} \rightarrow c_{1,1}=C1_{1,1}+C2_{1,1}$

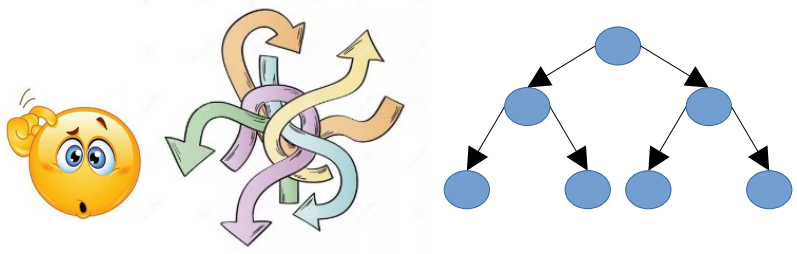
Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Descomposición exploratoria

- Se aplica a problemas que evolucionan durante la ejecución explorando un espacio de soluciones posibles.
- La descomposición progresa con la ejecución y genera **tareas de forma dinámica**.
- Son problemas de optimización discreta y combinatoria:
  - TSP
  - Puzzle-15
  - N-reinas
  - Problema de asignación cuadrática (QAP)
  - Redes neuronales evolutivas, programación entera 0/1, prueba de teoremas… etc.



- A los algoritmos que resuelven este tipo de problemas de les conoce como algoritmos **Branch and Bound**.
- Consiste en una enumeración de soluciones candidatas, mediante la búsqueda en un espacio de estados, que generalmente se estructuran en forma de un **árbol** que representa subconjuntos del conjunto de soluciones.

Descomposición exploratoria

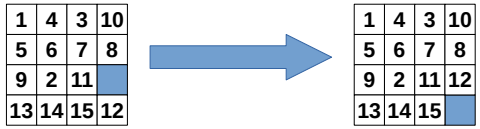
- El espacio de búsqueda se descompone en partes de menor tamaño y se asigna a tareas diferentes.
- Suelen resolverse con técnicas de búsqueda en árboles:
  - El algoritmo explora las ramas (**branch**) del árbol. Una rama se compara con los límites (**bound**) estimados superior e inferior de la solución óptima, y se descarta si no puede producir una solución mejor que la mejor encontrada hasta ahora por el algoritmo.
- El algoritmo depende de una estimación eficiente de los límites superior e inferior de las ramas del espacio de búsqueda. Si no hay límites disponibles, el algoritmo degenera en una búsqueda exhaustiva.
- Varios criterios para finalizar:
  - Se encuentra la primera solución
  - Se encuentra una solución óptima
  - Se encuentra una solución sub-óptima
  - Se encuentran todas las soluciones posibles

Suelen ser superlineales !!!

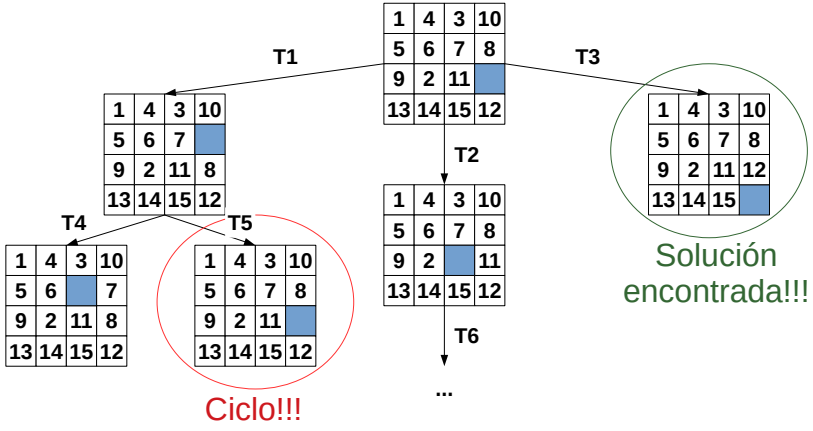
Tareas no finalizadas pueden terminar de inmediato cuando se alcance una solución deseada.
La carga de trabajo puede ser muy distinta entre la solución secuencial y la solución paralela.

Descomposición exploratoria
Ejemplo: Puzzle 15

- Se tiene un tablero inicial con 15 fichas numeradas y un espacio vacío, se quiere llegar a un tablero final realizando la menor cantidad de movimientos.

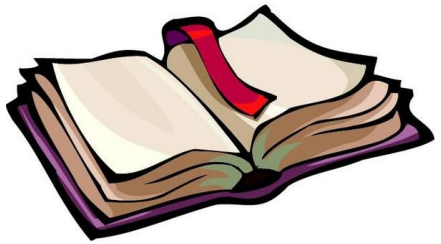


- El espacio vacío puede ser ocupado sólo por la ficha superior, inferior o las laterales.
- Cada posibilidad de movimiento genera una nueva tarea (estado) independiente.



Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Descomposición especulativa

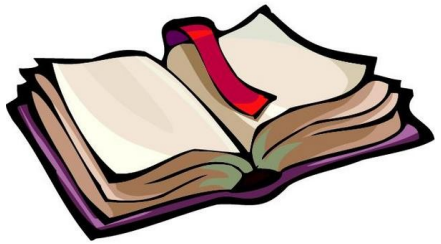
- Cuando las dependencias entre tareas no se conocen a priori es difícil asegurar la descomposición en tareas totalmente independientes.

- La **descomposición especulativa** se utiliza en problemas donde el próximo paso es una acción entre varias posibles, que sólo puede determinarse cuando las tareas precedentes concluyan:
  - Se asume cierto resultado de las tareas precedentes y se ejecutan los pasos posteriores.
  - Si la predicción es acertada se habrá ganado tiempo.
  - Si la predicción es errónea el trabajo realizado se desperdicia.
  - Debe existir un compromiso entre concurrencia alcanzable y el porcentaje de tareas que hay que abortar y reiniciar.

- Ejemplos:
  - Simulación discreta (incendios, inundaciones etc),
  - Movimiento de robots en un espacio limitado.

Agenda

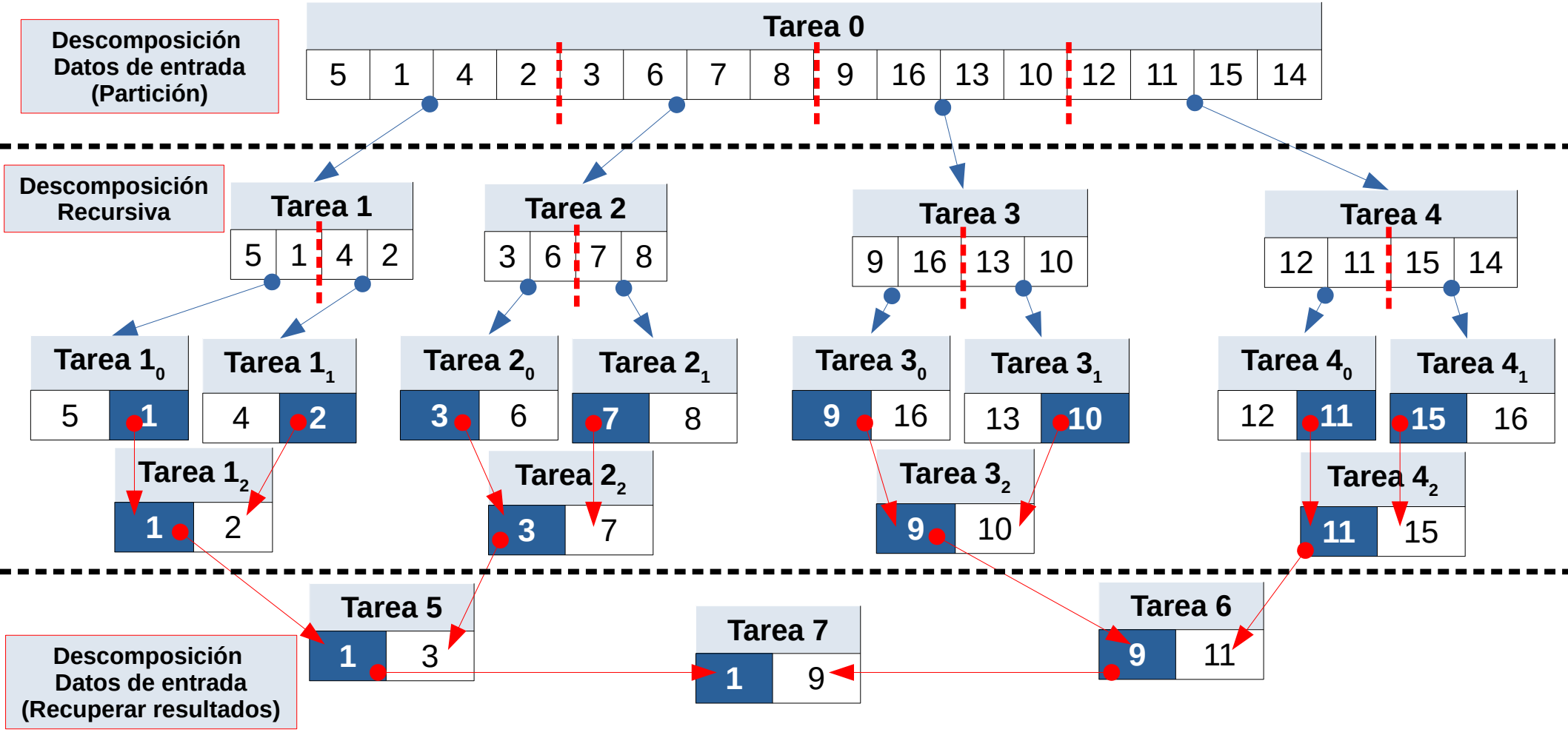
I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Descomposición Híbrida

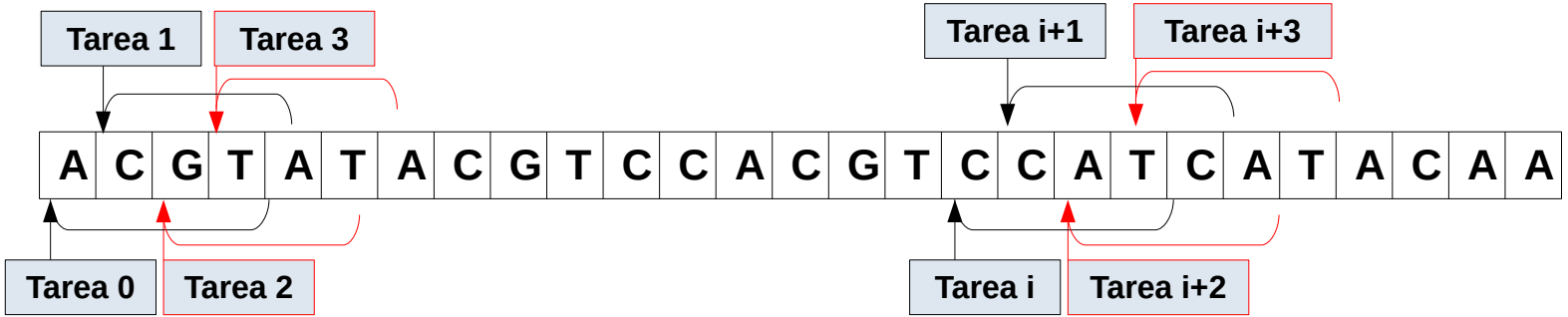
- Las técnicas de descomposición pueden combinarse.
- La descomposición puede estructurarse en múltiples etapas y en cada una aplicar una estrategia de descomposición diferente.
- **Por ejemplo:** encontrar el valor mínimo en un conjunto de números. Podemos aplicar una descomposición híbrida en dos etapas:
  1) **Descomposición por datos de entrada:** divide la entrada en **K partes** iguales y cada tarea consiste en obtener el mínimo de una parte asignada (etapa 2). Al final, se deben recuperar y reducir los resultados de cada tarea.
  2) **Descomposición recursiva:** las tareas en cada una de las K partes se dividen recursivamente en subtareas que obtienen el mínimo de esa parte.

Descomposición Híbrida
Búsqueda del mínimo en un arreglo

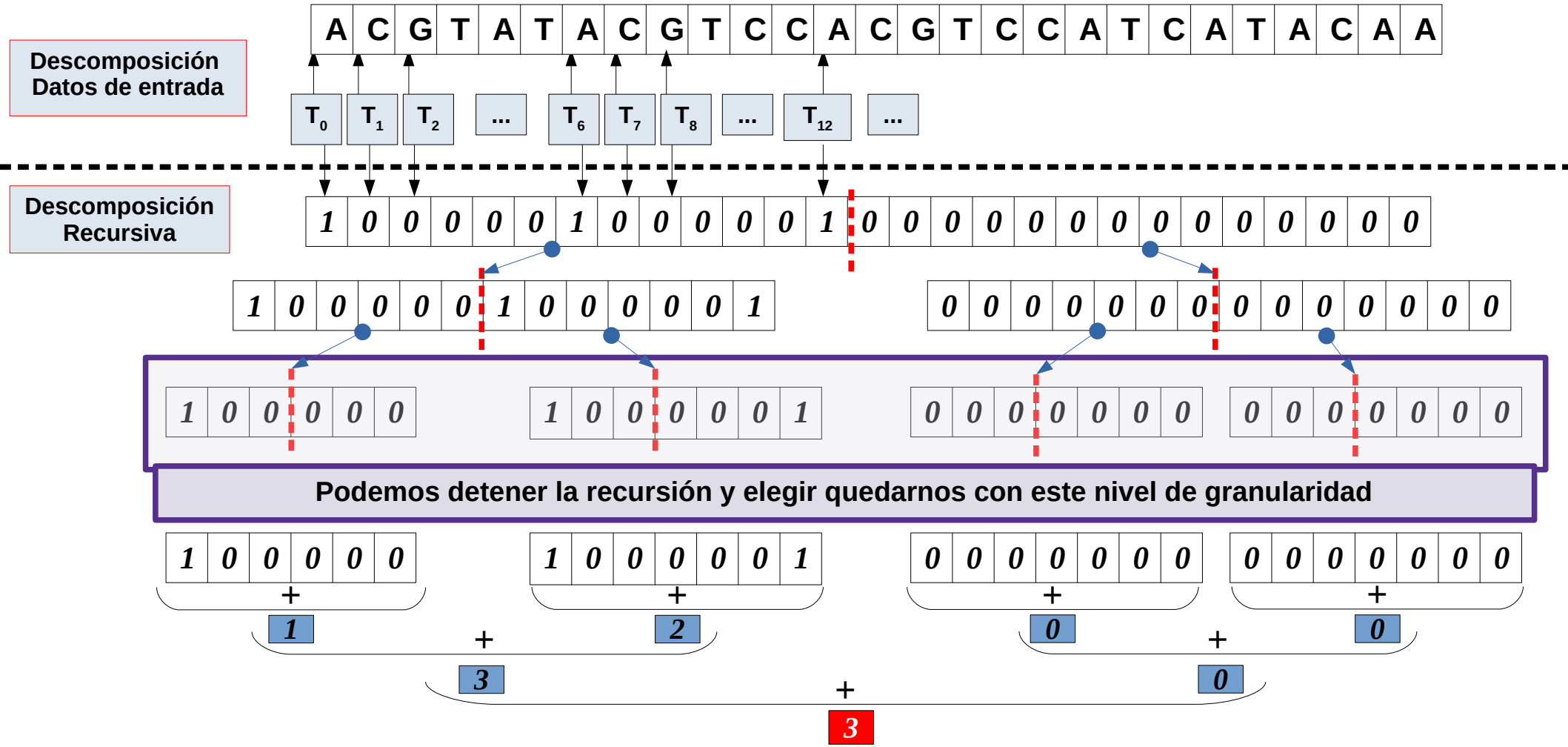


Descomposición Híbrida
Búsqueda de un patrón

- Buscar el número de veces que aparece el patrón “ACGT” en una cadena de ADN representada como un arreglo que contiene caracteres A, C, G y T:
  1) **Descomposición por datos de entrada:** cada tarea consiste en buscar el patrón en una posición. Existe superposición entre tareas. Cada tarea escribe en un arreglo de salida:
     - *1* si encontró el patrón
     - *0* si no encontró el patrón.
  2) **Descomposición recursiva:** se realiza un paso de reducción del arreglo de salida sumando todos los elementos.

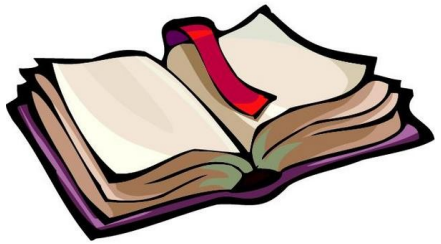


Descomposición Híbrida
Búsqueda de un patrón



Agenda

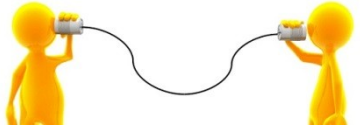
I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Comunicación

- Luego de identificar las tareas, el próximo paso es determinar los patrones de comunicación entre ellas y definir estructuras y algoritmos de comunicación apropiados.

- Una posible clasificación de los patrones de comunicación es la siguiente:
  - Global/Local
  - Estructurado/No estructurado
  - Estático/Dinámico
  - Síncrono/Asíncrono
  - Solo Lectura/Lectura-Escritura
  - Unidireccional/Bidireccional

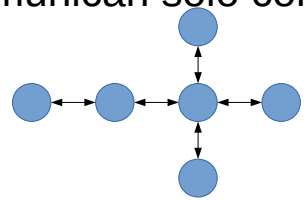


- Tener en cuenta que la comunicación es una característica inherente a los algoritmos paralelos (ausente en algoritmos secuenciales), es fuente de overhead y debe ser minimizada.

Comunicación
Patrones de comunicación

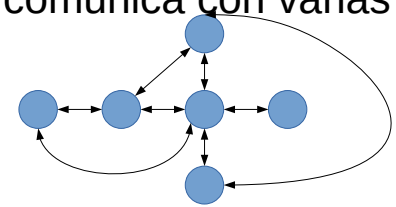
Local

Las tareas se comunican sólo con tareas vecinas.



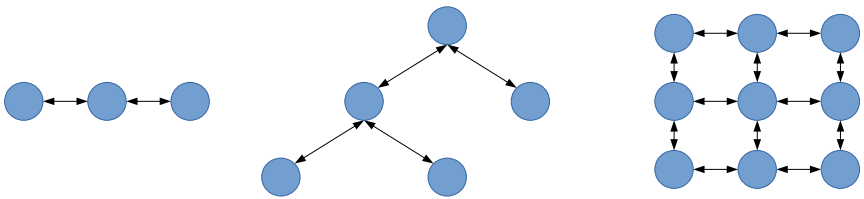
Global

Las tareas se comunica con varias tareas.



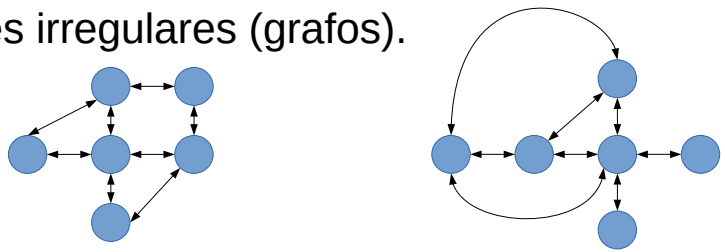
Estructurado

Patrones regulares (lista, árbol, grilla).



No estructurado

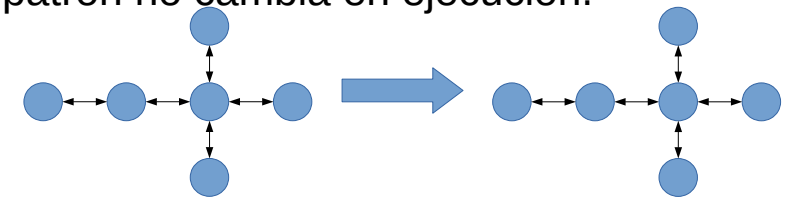
Patrones irregulares (grafos).



Comunicación

Estático

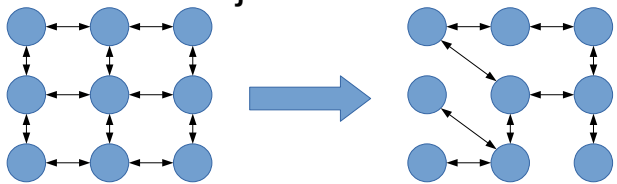
El patrón no cambia en ejecución.



Instante i Instante i+1

Dinámico

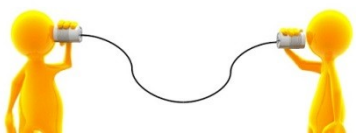
El patrón cambia en ejecución.



Instante i: estructurado Instante i+1: No estructurado

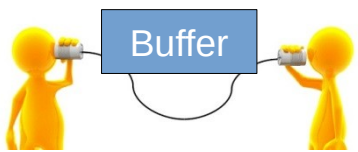
Síncrono

Comunicación entre emisor y receptor coordinada (PMS).



Asíncrono

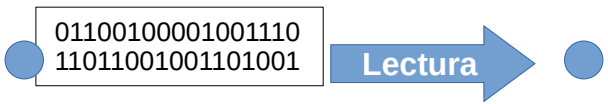
Comunicación entre emisor y receptor sin coordinación (PMA).



Comunicación

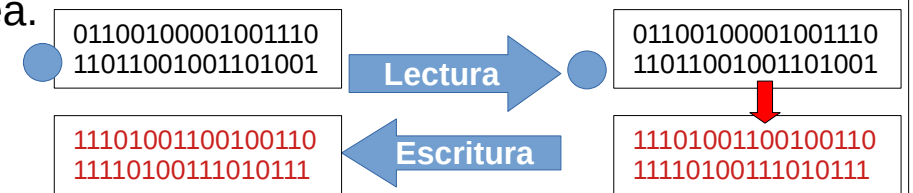
Solo lectura

Una tarea sólo lee datos asociados a otra tarea.



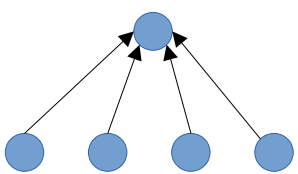
Lectura-Escritura

Una tarea lee y escribe datos asociados a otra tarea.



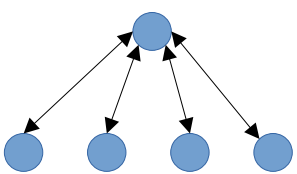
Unidireccional

La comunicación se desarrolla en único sentido.



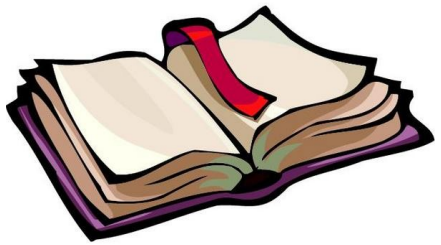
Bidireccional

La comunicación se desarrolla en ambos sentidos.



Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



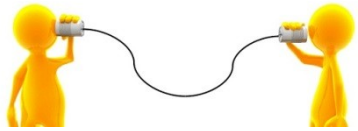
Aglomeración

- El algoritmo que surge de las etapas previas es aún abstracto, es decir NO está especializado para la ejecución eficiente en una máquina paralela particular.
- Puede ser ineficiente si existen más tareas que unidades de procesamiento disponibles.
- El objetivo de esta etapa es obtener un algoritmo que se ejecute en forma eficiente sobre una máquina paralela real. Por lo tanto, pueden descartarse posibilidades detectadas anteriormente.
- En la etapa aglomeración se combinan las tareas obtenidas en una etapa previa en otras de mayor tamaño con el objetivo de:
  - Reducir el overhead de comunicación y optimizar la localidad de los datos
  - Conservar la escalabilidad
  - Reducir el costo de la Ingeniería de Software

Aglomeración
Overhead en las comunicaciones

- Un tema crítico, que influye en el rendimiento de una aplicación paralela, es el costo de las comunicaciones.

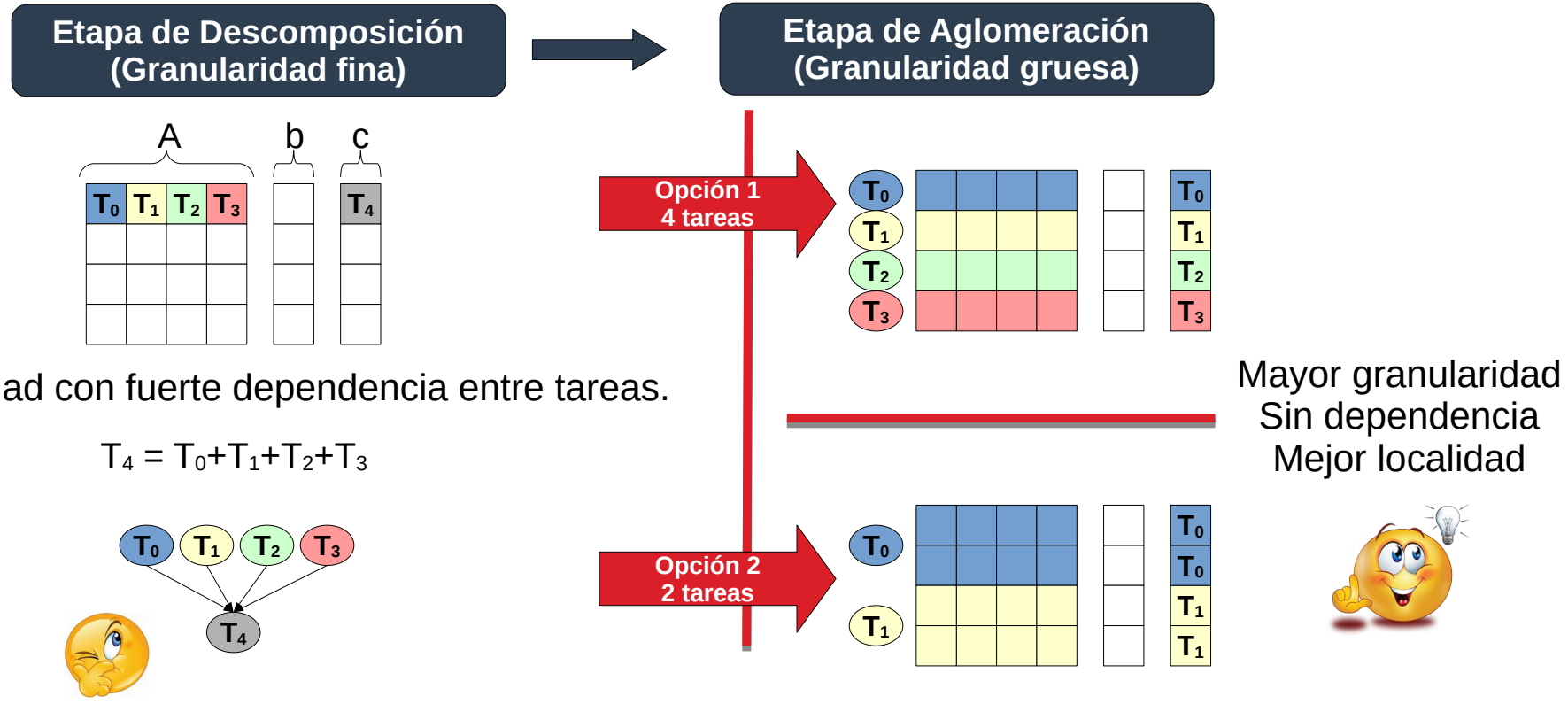
- Fuentes de overhead en la comunicación pueden ser:
  - El costo del startup de las comunicaciones
  - El costo proporcional de la cantidad de datos a transferir
  - Esperas inactivas:
    - Restricciones impuestas por el grafo de dependencias.
    - Problemas que hacen que algunas tareas se terminen antes que otras (balance de carga)



- Una forma de reducir este overhead es incrementar la granularidad asociando más cómputo por tarea.

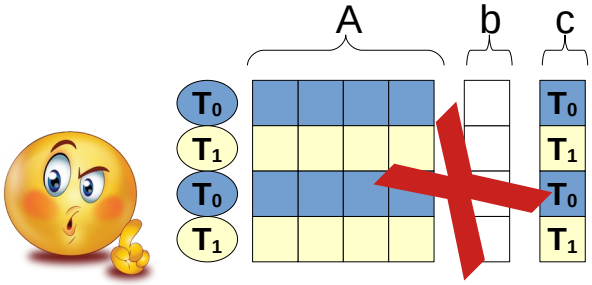
Aglomeración
Overhead en las comunicaciones y localidad

Incrementando la granularidad al multiplicar una matriz por un vector ($c=Ab$)



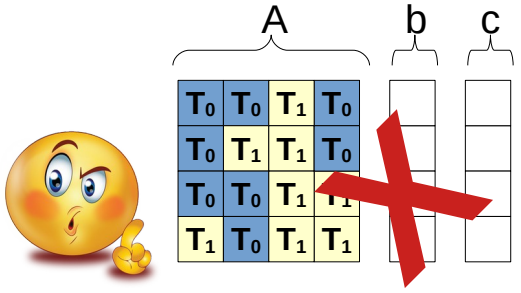
Aglomeración
Overhead en las comunicaciones y localidad

- Otras opciones pero no adecuadas:



Esta distribución atenta contra la localidad de los datos.

Si consideramos un algoritmo implementado en un modelo de memoria compartida, es probable que la matriz A se encuentre ordenada por filas en memoria. A medida que el tamaño de A aumenta, la cantidad de fallos de caché se incrementa e impacta en el rendimiento.



Esta distribución atenta contra la localidad de los datos y tiene una fuerte dependencia entre las tareas.

Aglomeración
Overhead en las comunicaciones - Replicar cómputo

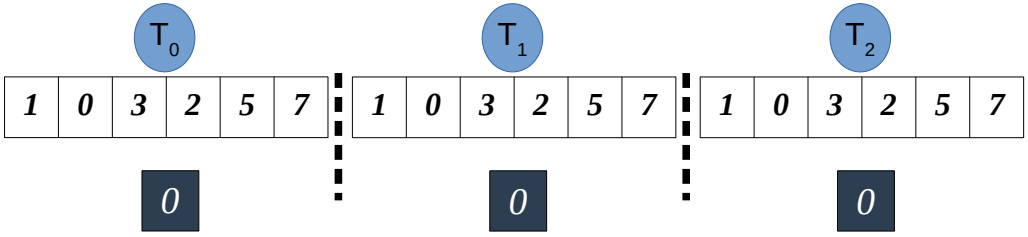
- Inicialmente se busca no replicar cómputo ni datos. Sin embargo, una forma de reducir el costo de ciertas comunicaciones es replicar el cómputo:
  - Cuando el costo de comunicar datos es mucho mayor que el de replicar el cómputo.
  - **Ejemplo:** obtener el valor mínimo de una lista. Todas las tareas poseen la misma lista.

Opción 1
Comunicar/Sincronizar

Cada tarea consiste en determinar sobre que parte de la lista debe buscar y obtiene el mínimo local.
Luego todas las tareas deben sincronizar para obtener el mínimo global y comunicarlo.

Opción 2
Replicar cómputo

Replicar el cómputo si el costo de las sincronizaciones es muy alto.



Aglomeración
Conservar la escalabilidad

- Un error común en esta etapa es tomar decisiones de diseño que limitan innecesariamente la escalabilidad del algoritmo.
- Debemos combinar las tareas de forma tal que podamos ejecutar nuestro programa soportando cambios en el número de unidades de procesamiento.
- **Ejemplo:** resolver la expresión de matrices cuadradas $AB + CD$. Si sólo tenemos 2 unidades de procesamiento:

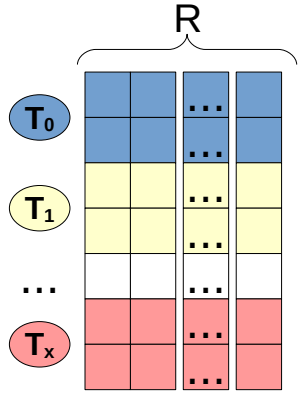
Una solución que genere dos tareas y las asigne así:

Tarea 0	*Tarea 1*
AB	**+** ***CD***

No es escalable!!!

No podrán obtenerse beneficios cuando el número de unidades de procesamiento crece.

Una posible **solución escalable** combina tareas de acuerdo al número de filas de la matriz resultado:

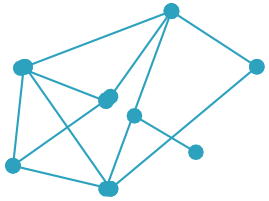


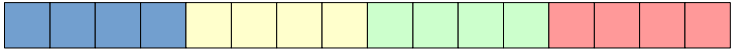
$$R = AB + CD$$

Aglomeración
Costos en la Ingeniería de Software

- Se suele diseñar un software paralelo pensando que es un sistema aislado. Esto no siempre es así ya que puede formar parte de un gran sistema.

- A la hora de tomar decisiones de diseño debemos tener en cuenta dos aspecto de la ingeniería de software:
  - Evitar excesivos cambios en el código ante eventuales cambios futuros.
  - La organización de los datos en el sistema visto como un todo:
    - El mejor algoritmo para nuestro diseño necesita una estructura de datos en 2 dimensiones pero una fase anterior requiere la estructura en 3 dimensiones.
    - Debemos cambiar uno u otro algoritmo o agregar una fase de reestructuración.

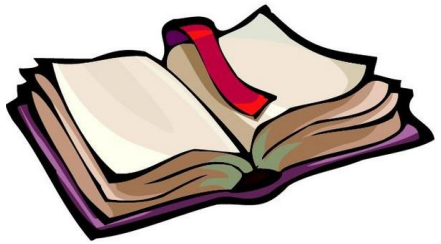




Recordar que buscamos mejorar el rendimiento de un algoritmo!!!
Diseño vs. Rendimiento a veces son objetivos contrapuestos

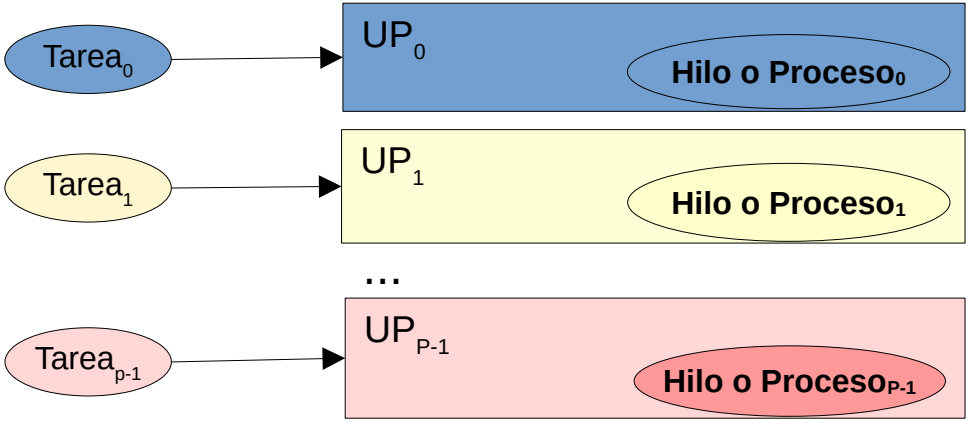
Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Mapeo

- En esta etapa se asignan las tareas a unidades de procesamiento con el objetivo de:
  - **Maximizar el uso de las unidades de procesamiento:** Ubicar tareas que pueden ejecutarse concurrentemente en distintas unidades de procesamiento
  - **Minimizar la comunicación entre unidades de procesamiento:** Ubicar tareas que se comunican con frecuencia de manera de aprovechar la localidad
- Luego, dependiendo del modelo de programación elegido, se crea un **proceso** (o hilo) por cada unidad de procesamiento que se encargará de resolver la tarea asignada.



Aclaración:
Dado que asignamos unívocamente un proceso o hilo a una unidad de procesamiento, de acá en más nos referiremos a unidad de procesamiento o proceso de manera indistinta.

Mapeo

- Un mapeo adecuado debe asegurar:
  - Un correcto balance de carga entre unidades de procesamiento (volumen de trabajo uniforme entre tareas).
  - Evitar esperas ociosas (Idling).
- En arquitecturas de memoria compartida, el sistema operativo se encarga de mapear automáticamente, y de manera adecuada, cada proceso o hilo a una unidad de procesamiento independiente. Sin embargo, esta asignación puede no ser la mejor y por cuestiones de rendimiento el usuario podría elegir otro tipo de asignación (afinidad).
- El mapeo se puede especificar estáticamente o determinar en tiempo de ejecución (dinámicamente) mediante algoritmos de balance de carga.

TENER EN CUENTA!!!: El problema de mapeo es NP- Completo 

No existe un algoritmo de tiempo polinomial tratable computacionalmente que lo resuelva

Mapeo

- Ananth Grama[1] propone dos estrategias:

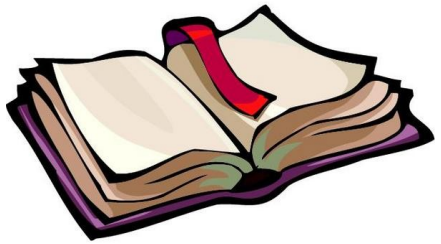
  - **Mapeo Estático:** las tareas se distribuyen entre las unidades de procesamiento antes de la ejecución.
  - **Mapeo Dinámico:** las tareas se distribuyen entre las unidades de procesamiento durante la ejecución.

- El mapeo depende de como se generan las tareas en las etapas previas:
  - **Tareas generadas estáticamente:** cualquier mapeo es válido siendo el mapeo estático el mas habitual. Un mapeo dinámico puede usarse cuando el volumen de datos por unidad de procesamiento es muy grande y esta unidad no es capaz de soportarlo. Si embargo, el mapeo dinámico tendría mayor costo de comunicación.
  - **Tareas generadas dinámicamente:** Si el número de tareas se desconoce es más efectivo el mapeo dinámico. Un mapeo estático podría desbalancear la carga o generar esperas ociosas.

[1]Libro: An Introduction to Parallel Computing - 2003

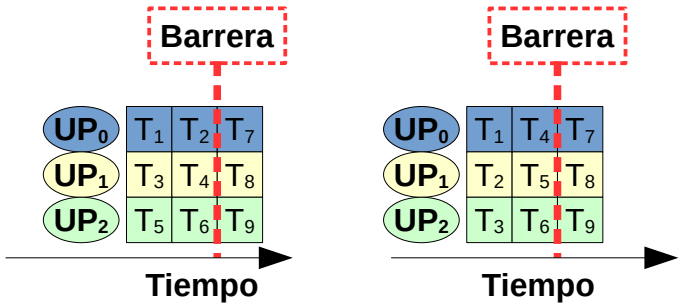
Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



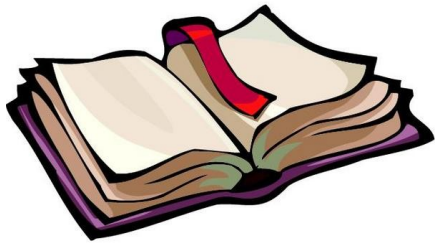
Mapeo estático

- Existen tres estrategias de mapeo estático:
  - Basada en partición de datos
  - Basada en partición de tareas
  - Jerárquico
- La elección de una buena estrategia de mapeo estático depende de varios factores:
  - Conocimiento de las tareas
  - Tamaño de los datos asociados a cada tarea
  - Las características de interacción entre tareas
  - El paradigma de programación paralela
- El mismo número de tareas pueden dar mapeos diferentes:



Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación

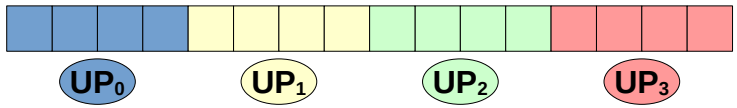


Mapeo estático
Basado en partición de datos

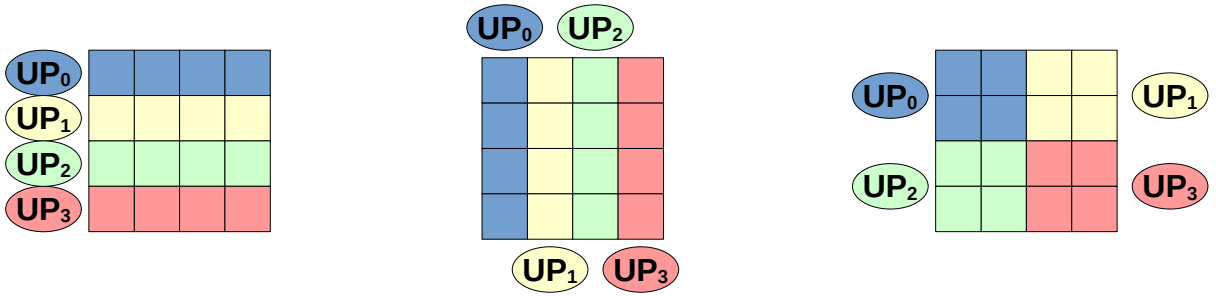
- Se derivada de una descomposición basada en datos.
- Existen dos formas de representar datos en un algoritmo:
  - Arreglos
  - Grafos
- De esto se derivan las siguientes estrategias de mapeo estático basadas en partición de datos:
  - Arreglos:
    - Por Bloques
    - Cíclico
  - Grafos

Mapeo estático
Basado en partición de datos – Arreglos por bloques

- En **arreglos unidimensionales** se utiliza una distribución por bloques distribuyendo porciones del arreglo entre las unidades de procesamiento.



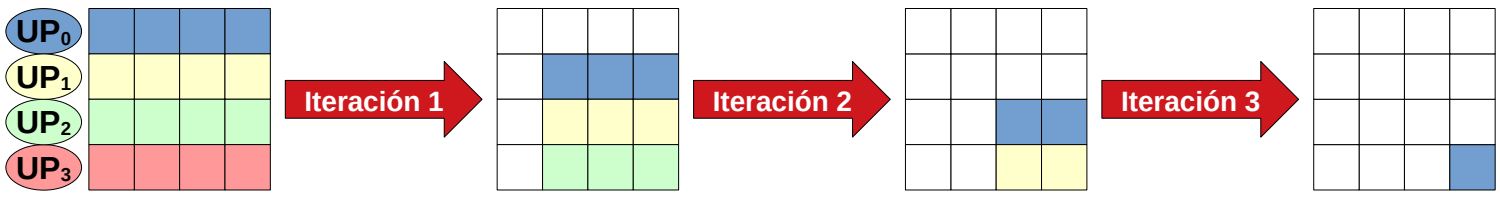
- En **arreglos multidimensionales** se utiliza una distribución en bloques de manera que puedan aprovechar la localidad de datos.



Mapeo estático
Basado en partición de datos – Arreglo cíclicos

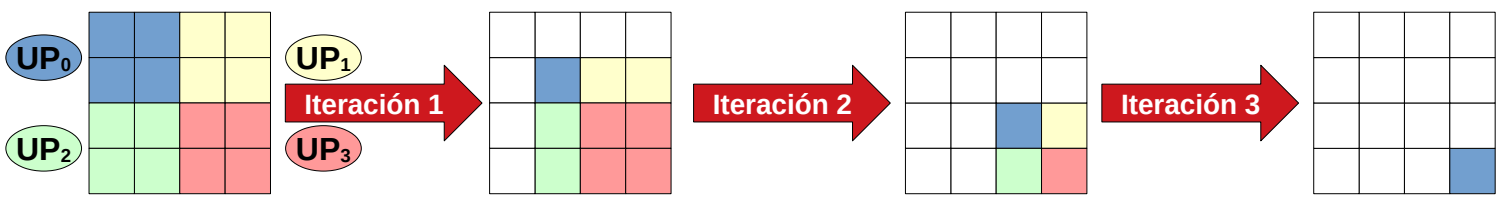
- El mapeo cíclico se utiliza cuando el volumen de trabajo se modifica en tiempo de ejecución.
- **Ejemplo**: factorización LU.

Solución 1



La solución presenta **desbalance de carga:** En la iteración 2 sólo dos unidades de procesamiento trabajan.

Solución 2

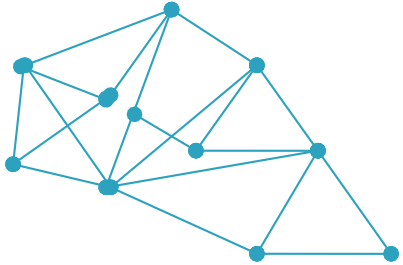


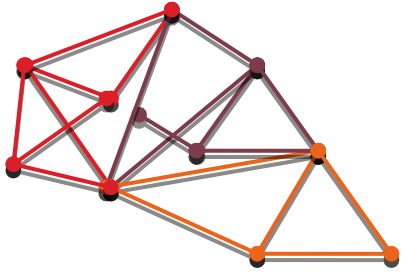
Otra distribución mejora el balance de carga: Mejora la iteración 2, todas las unidades de procesamiento trabajan.

Mapeo estático
Basado en partición de datos – Grafos

- Existen algoritmos que operan sobre estructuras de datos dispersas (grafos) que no son regulares como los arreglos.
- Generalmente asociados a fenómenos físicos.
- Mapear estáticamente estas estructuras tiende al desbalance de carga.
- La idea es particionar el grafo de manera de distribuir los vértices a procesar de manera "balanceada" entre las unidades de procesamiento.

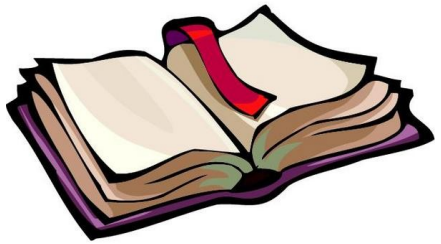
Encontrar una partición óptima (balanceada) es un problema NP- Completo.
Se pueden utilizar algoritmos heurísticos que obtienen una partición subóptima.





Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Mapeo estático
Basado en partición de tareas

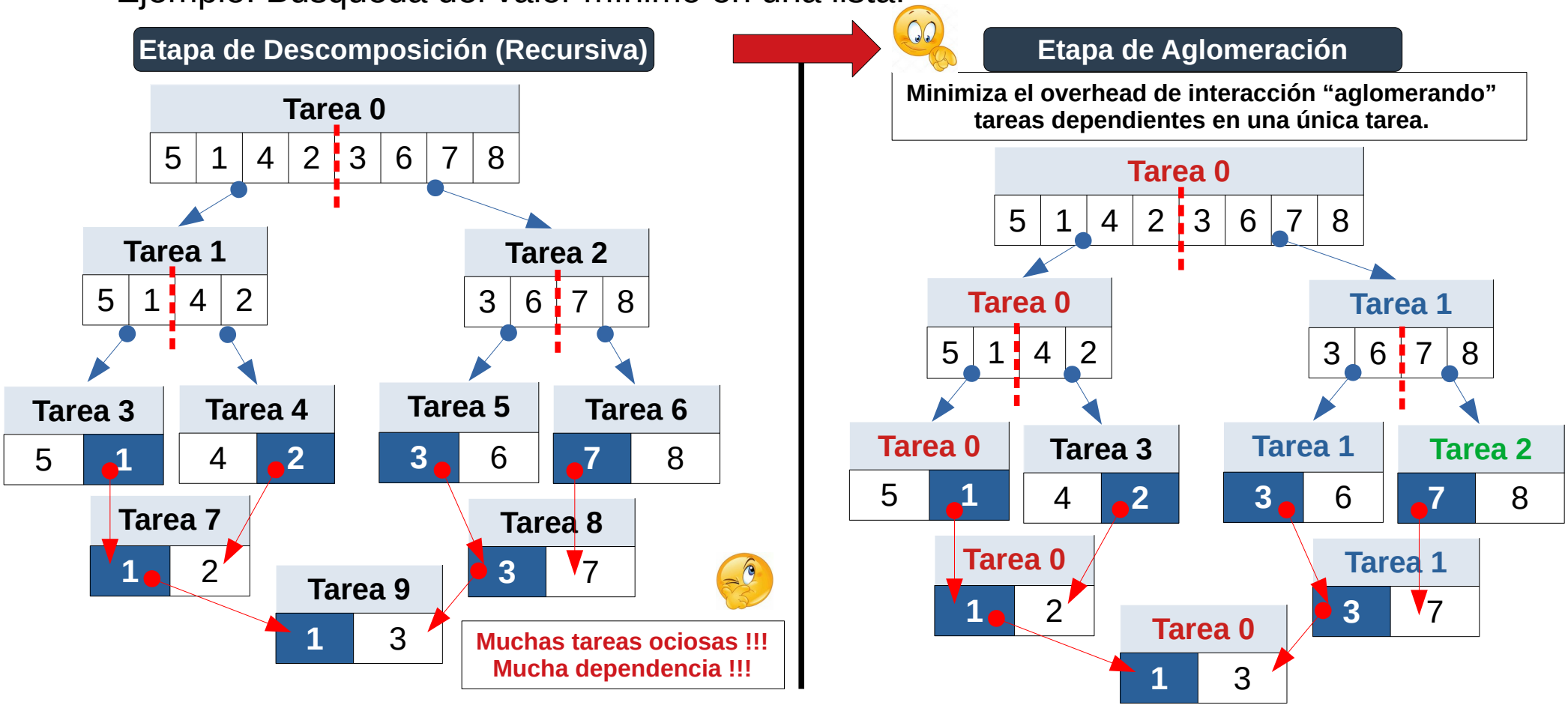
- Se puede utilizar esta técnica cuando el cálculo puede expresarse naturalmente en forma de un grafo con dependencias estáticas de tareas de tamaño conocido.

Anteriormente GRAFO=DATOS ahora GRAFO=TAREAS

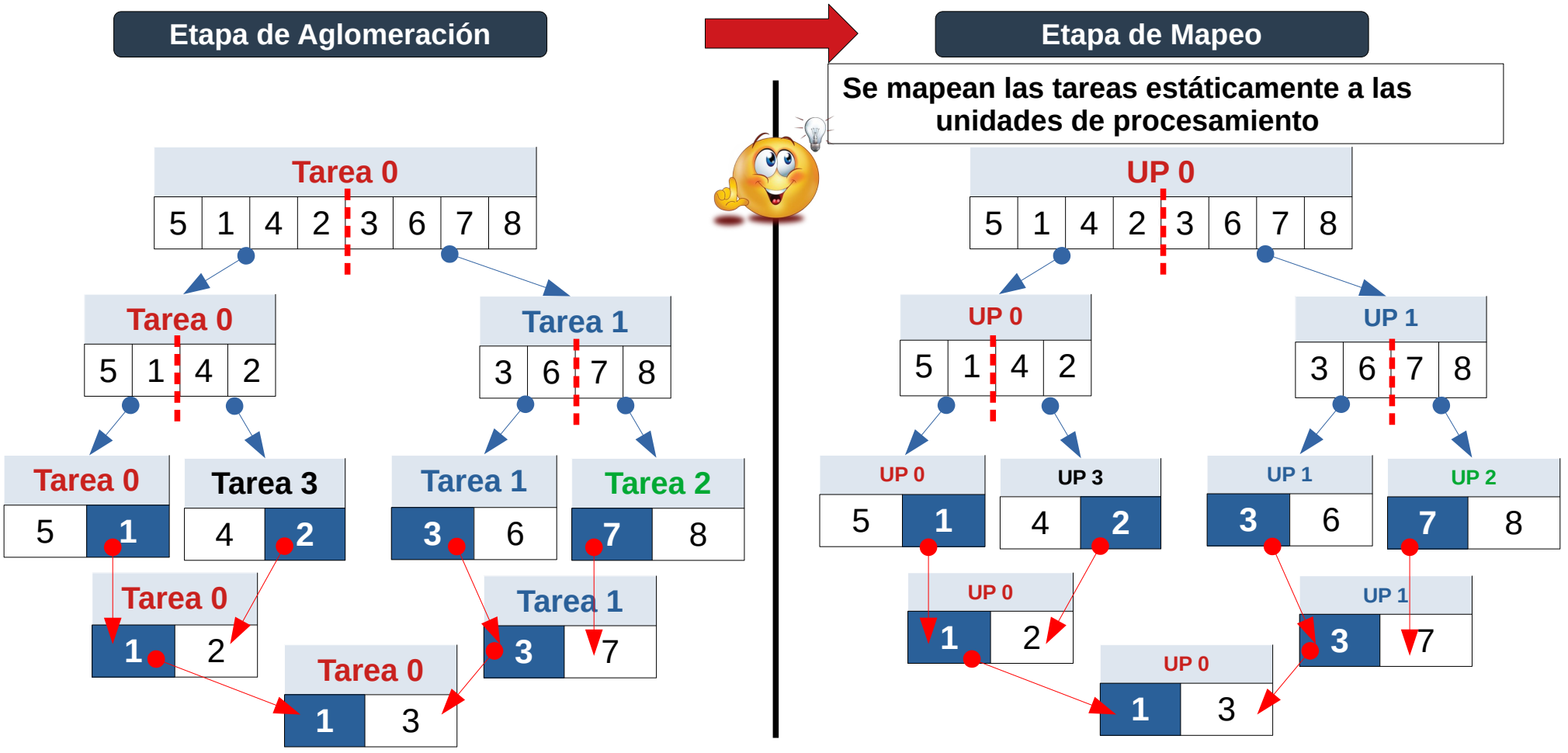
- Ejemplos:
  - Grafo de dependencias de tareas con forma de árbol binario derivado de una descomposición recursiva de tareas (búsqueda del valor mínimo en una lista DATOS=ARREGLO)
  - Multiplicación de una matriz dispersa (la mayor parte de los elementos son ceros) por un vector

Mapeo estático
Basado en partición de tareas

- Ejemplo: Búsqueda del valor mínimo en una lista.

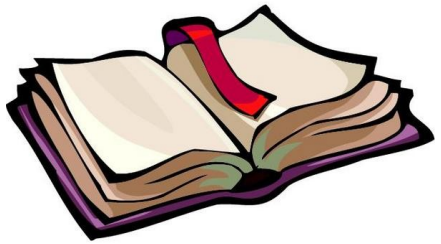


Mapeo estático
Basado en partición de tareas



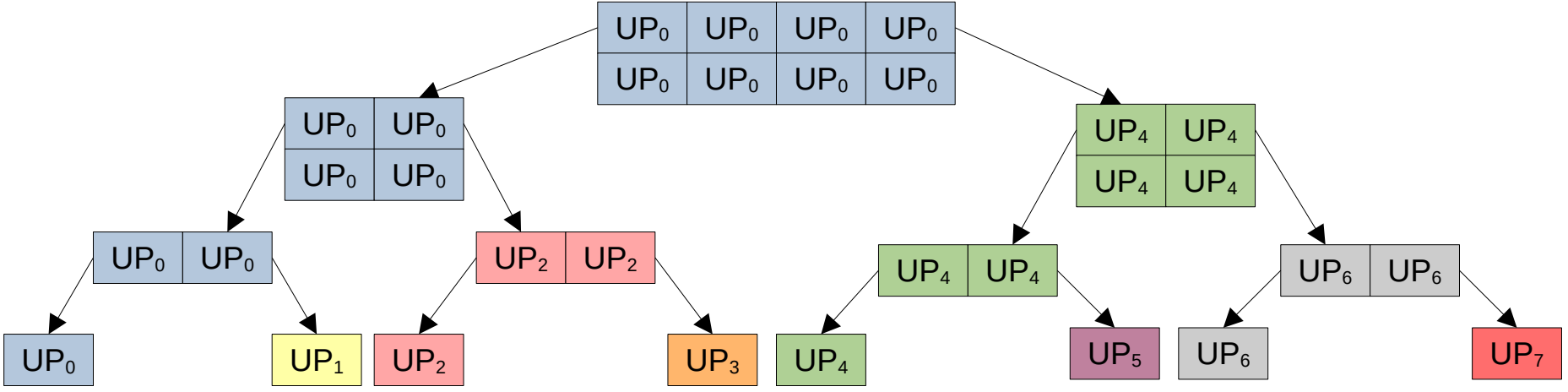
Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Mapeo estático Jerárquico

- Un mapeo basado sólo en un grafo de dependencias de tareas puede tender al desbalance de carga, esperas ociosas o concurrencia inadecuada.
- En un grafo con dependencias de tareas en forma de árbol binario:

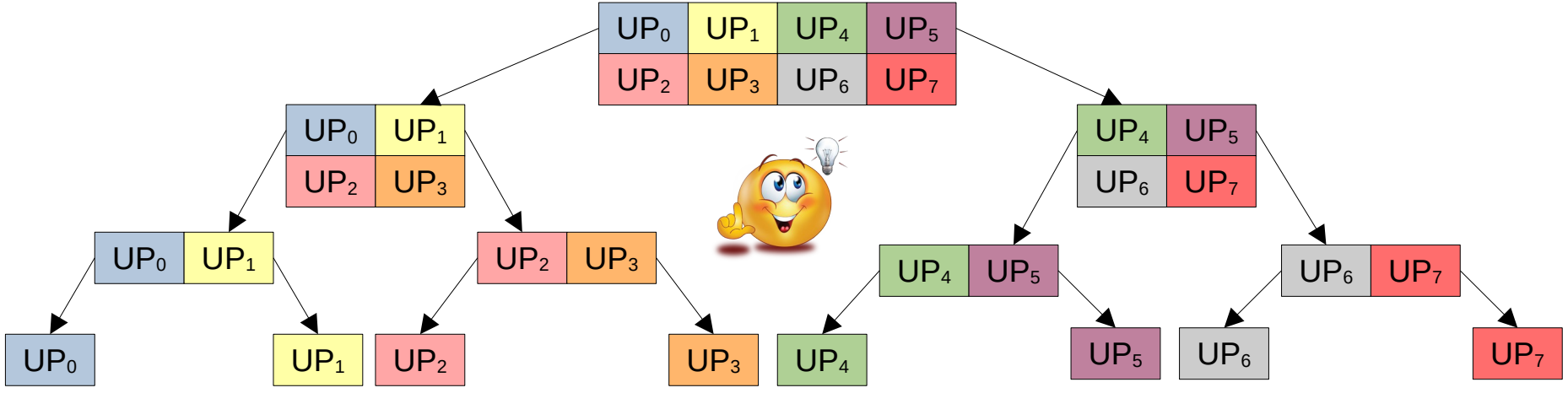


En las primeras etapas de cómputo hay muchas unidades de procesamiento ociosas!!!

Mapeo estático Jerárquico

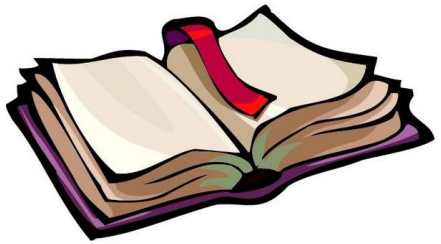
- Si una tarea tiene un volumen de trabajo suficientemente grande, y el problema lo admite, es posible distribuir el trabajo entre varias tareas con una descomposición de datos en niveles superiores.

- Un ejemplo donde se puede utilizar este mapeo es en problemas de factorización de matrices dispersas (valores 0, de forma irregular, en la mayoría de sus posiciones).



Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación

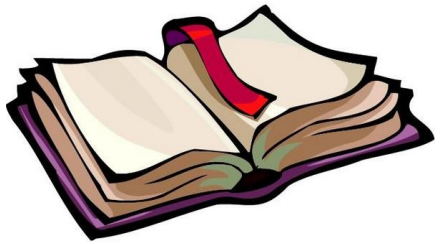


Mapeo dinámico

- Se utiliza mapeo dinámico en situaciones donde:
  - El mapeo estático lleva a una distribución de trabajo desbalanceada entre unidades de procesamiento (Por ejemplo: al utilizar arquitecturas heterogéneas)
  - El grafo de dependencias de tareas es por naturaleza dinámico
- Un mapeo dinámico puede generar overhead pero se compensa el balance de carga.
- Como la razón principal para usar mapeo dinámico es el balance de carga entre unidades de procesamiento, se le suele llamar balance de carga dinámico.
- Las estrategias clasifican en:
  - Centralizadas
  - Distribuidas

Agenda

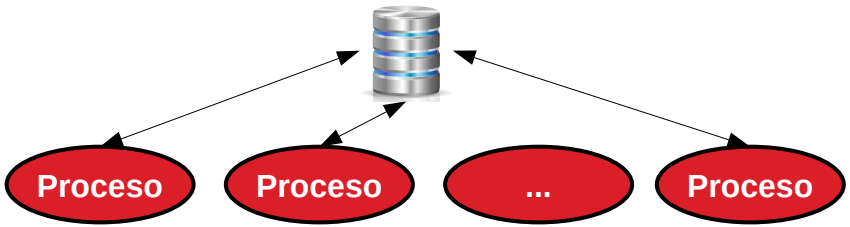
I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Mapeo dinámico Centralizado

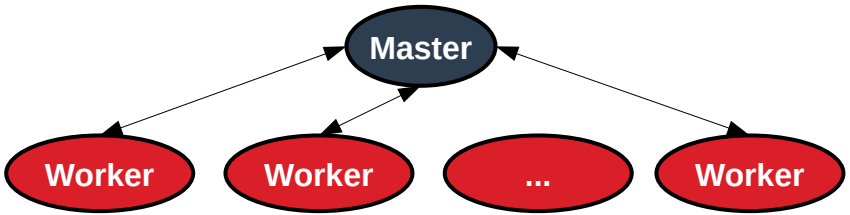
- Todas las tareas a realizar se mantienen en una de dos posibles formas:

Estructura de datos centralizada



Un proceso ocioso obtiene tareas de la estructura de datos centralizada.
Un proceso puede generar una nueva tarea y agregarla a la estructura de datos centralizada.

Uno o varios Procesos Master



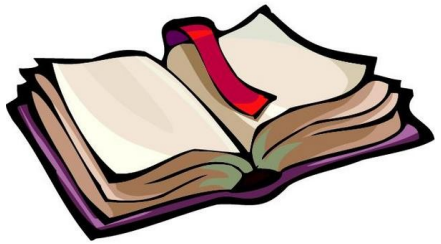
Worker (o Slave):
Procesos que obtienen del Master el trabajo a realizar.

Un proceso ocioso obtiene tareas solicitándolas al Master.
Un proceso puede generar una nueva tarea e informar al Master.

- Esta estrategia puede tener **problemas de escalabilidad** a medida que se incrementa el número de procesos a utilizar: un gran número de accesos a la estructura centralizada o al proceso Master genera **cuellos de botella**.

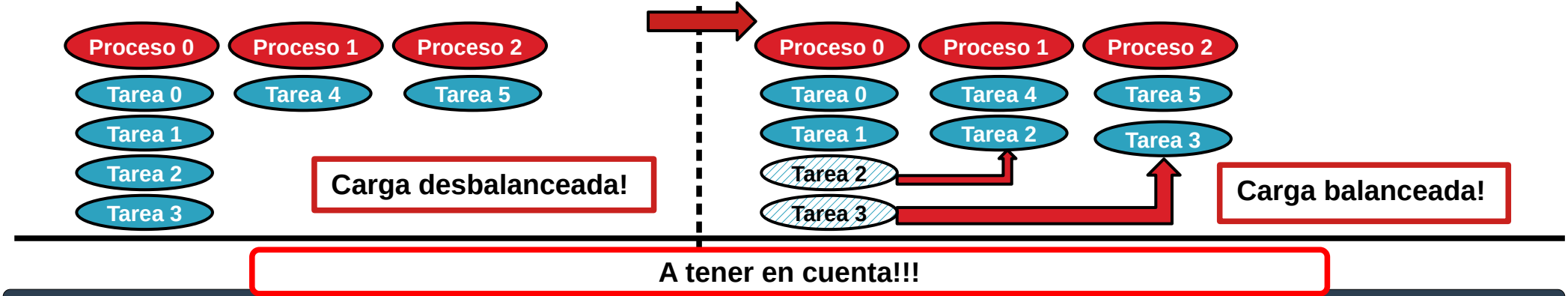
Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Mapeo dinámico Distribuido

- En un esquema **dinámico distribuido** las tareas se distribuyen entre las unidades de procesamiento:
  - Los procesos intercambian tareas para balancear la carga.
  - Cualquier proceso puede enviar o recibir tareas evitando los cuellos de botella.



A tener en cuenta!!!

¿Cuánto trabajo se transfiere?

Transferir Mucho trabajo: puede implicar mucha comunicación.
Transferir Poco trabajo: puede hacer que algunos procesos pasen mucho tiempo sin trabajar.

¿Cómo se transfiere el trabajo?

Directo: Un proceso decide a quien enviar trabajo.
Por demanda: Un proceso solicita a otro que le envíe trabajo.
Work Stealing: Un proceso "roba" trabajo a otro proceso.

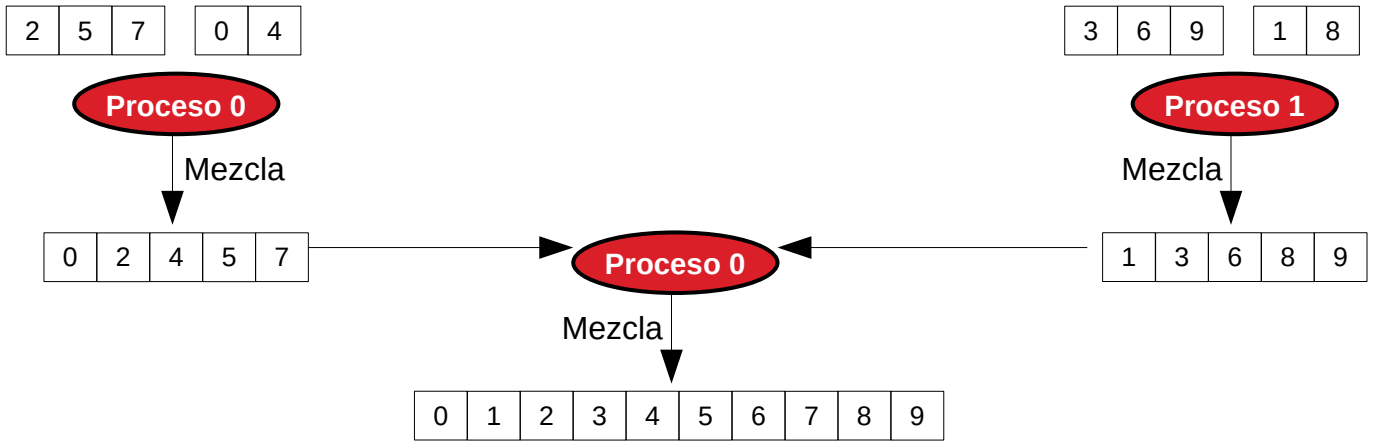
¿Cuando se transfiere el trabajo?

La solicitud de trabajo podría llegar cuando el proceso está trabajando, esto implica una demora en el proceso que requiere ese trabajo.

Mapeo dinámico Distribuido

- En esta etapa hay que tener en cuenta el tipo de aplicación que ejecutamos y cuál es el efecto de ciertos aspectos relacionados al hardware.

- Suponer un programa paralelo sortByMerge ejecutando sobre un cluster de multicores. Después de cierto tiempo de ejecución, quedan dos procesos encargados de mezclar.

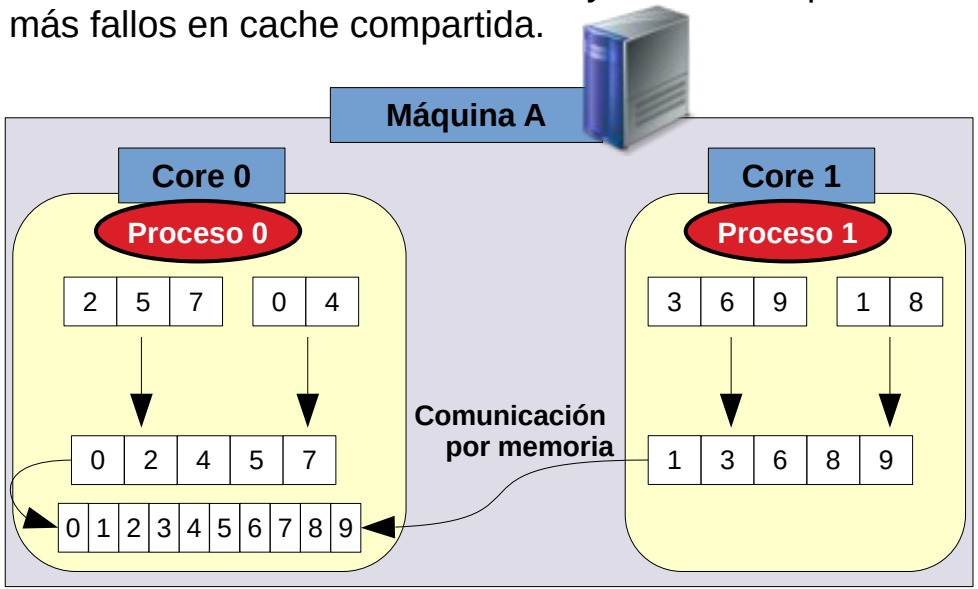


Mapeo dinámico Distribuido

- Dos alternativas de mapeo:

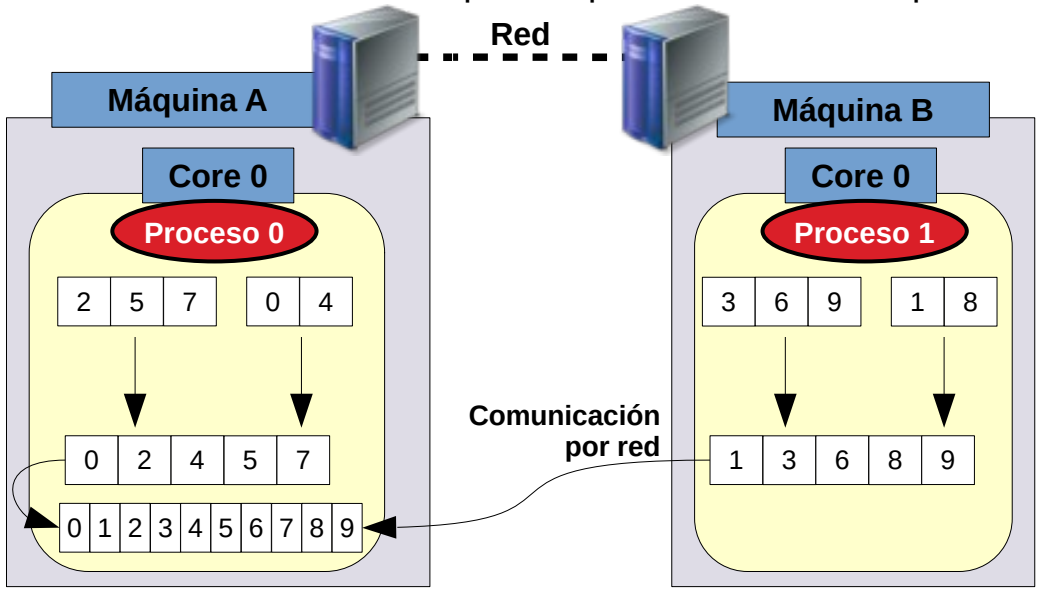
Dos procesos en una máquina

Cada uno en un core diferente. Mayor localidad pero más fallos en cache compartida.



Un proceso por máquina

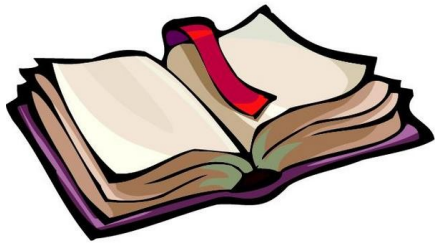
Menos fallos de cache compartida pero comunicación por red.



La mejor solución dependerá del tamaño del problema y que fuente de overhead (fallos de caché o comunicación) tiene menor peso.

Agenda

I. Introducción al diseño de algoritmos paralelos
II. Etapas de diseño
    i. Descomposición o Particionado
        i. Recursiva
        ii. Basada en datos (Salida | Entrada | Entrada/Salida | Intermedios)
        iii. Exploratoria
        iv. Especulativa
        v. Híbrida
    ii. Comunicación
    iii. Aglomeración
    iv. Mapeo
        i. Estático (Partición de datos | Partición de Tareas | Jerárquico)
        ii. Dinámico (Centralizado | Distribuido)
    v. Del diseño a la implementación



Del diseño a la implementación

Del diseño a la implementación

- Los programadores no habituados a programar algoritmos paralelos tienden a desarrollar al estilo secuencial: implementan un algoritmo completo o una gran parte de este.
- El principal problema surge ante la aparición de un error, la depuración de un programa paralelo es extremadamente difícil:
  - Interleavings: escenarios difíciles o imposibles de replicar (**el algoritmo a veces anda!!!**)
  - Errores de comunicación y sincronización
- Una buena práctica es implementar y evaluar el algoritmo paralelo en etapas muy simples:
  - Evaluar que se crearon todos los hilos/procesos (**hello ID!!!**)
  - Evaluar que las comunicaciones funcionan (se recibe lo que se tiene que recibir)
  - Evaluar como responde el algoritmo a medida incrementamos el número de procesos/hilos