



Tema 2

Modelos de programación paralela adaptados a la arquitectura

Nicolás Calvo Cruz
Dpto. de Arquitectura y Tecnología de los Computadores
@ncalvocruz
ncalvocruz@ugr.es



Objetivos

- Aprender a **encontrar** puntos para paralelizar un algoritmo
- Aprender a **descomponer** un algoritmo en tareas
- Aprender a identificar y respetar **dependencias** entre tareas
- **Agrupar** las tareas que se pueden realizar en paralelo
- Aplicar revisiones de **diseño**





Motivación

- 1 Abordar **problemas** cada vez más **grandes**.
- 2 Obtener (mejores) soluciones en un **tiempo razonable**.
- 3 Ilustrar diferentes **enfoques** de la bibliografía para **paralelizar** algoritmos tradicionales.

Índice

1. Introducción
2. ¿Cómo encontrar concurrencia?
 - a. Encontrar tareas
 - b. Agrupar tareas
 - c. Buscar patrones de comunicación
3. Patrones de los principales algoritmos paralelos
4. Estructuras de algoritmos más comunes
5. Qué hacer con las estructuras de datos
6. Ejemplos prácticos de algoritmos para arquitecturas de memoria compartida

1. Introducción





1. Introducción

Los pasos básicos para paralelizar un algoritmo son:

1. **Encontrar** la **conurrencia** del algoritmo de partida.
2. Decidir la **estructura** del algoritmo.
3. Dar **soporte a las estructuras de datos** necesarias para la implementación.
4. **Implementación** según la metodología deseada.

Buscar Conurrencia

Elegir Estructura del Algoritmo

Dar soporte a las Estructuras de Datos

Implementación



1. Introducción

Los pasos a seguir y cómo desarrollarlos, dependen del **Dominio de problema**.



2. Cómo encontrar concurrentencia

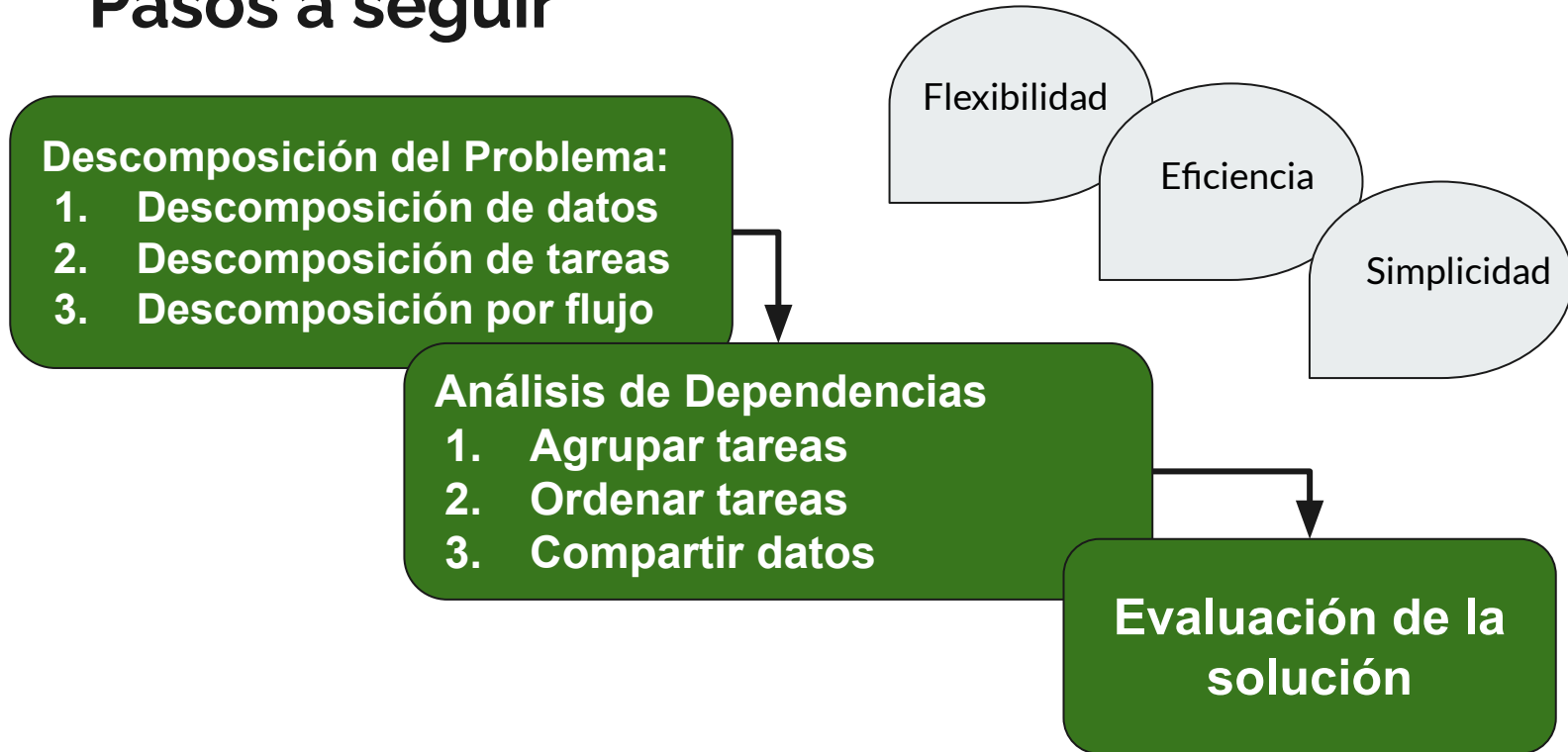
2. Cómo encontrar concurrencia

Primeras cuestiones

- ¿Es **paralelizar** la opción adecuada? (P. Ej., Búsqueda Binaria)
- Control del **Dominio del Problema**
- Identificación de las **partes más costosas** computacionalmente (**Profiling**)
- Pasos a seguir para encontrar concurrencia
 - **Descomposición del problema**
 - **Análisis de dependencias**
 - **Evaluación del diseño elegido**

2. Cómo encontrar concurrencia

Pasos a seguir



2. Cómo encontrar concurrencia

1. Descomposición del problema según sus datos



2. Cómo encontrar concurrencia

1. Descomposición del problema según sus datos

- La descomposición de datos **requiere** que el algoritmo use **una estructura de datos divisible** entre unidades de ejecución (hilos/procesos) concurrentes.
- Paralelizar en este caso es hacer que **cada unidad de ejecución** trabaje con una **parte** de los datos.
- Debe haber, al menos, tantas porciones como unidades de ejecución.
- Este enfoque sólo es eficiente si las **operaciones sobre la estructura de datos** son relativamente **independientes**.

2. Cómo encontrar concurrencia

1. Descomposición del problema según sus datos

- Este enfoque es adecuado si:
 - La **mayor parte** computacional del algoritmo se dedica a **gestionar la estructura** de datos (matrices, vectores, grafos...)
 - Las **operaciones en cada parte** son **iguales** y se pueden hacer **independientemente**.
- **A veces**, la descomposición de datos, genera una descomposición de tareas.

2. Cómo encontrar concurrencia

1. Descomposición del problema según sus tareas



2. Cómo encontrar concurrencia

1. Descomposición del problema según sus tareas

- El algoritmo se ve como un flujo de instrucciones que podrían ejecutarse **simultáneamente** y que condicionarán el proceso general del algoritmo.
- La pregunta es: ¿qué operaciones se pueden realizar de forma **simultánea e independiente**?
- En un equipo, esta fase se aborda de forma individual para obtener **varias opciones posibles** y elegir la mejor.

2. Cómo encontrar concurrencia

1. Descomposición del problema según sus tareas

- Las tareas podrían ser:
 - **Llamadas a función** (functional decomposition)
 - **Iteraciones de bucles** (loop-split)
 - **Tareas que reparten datos** a las diferentes unidades de ejecución y los actualizan
- Debe cumplirse:
 - **Que compense** crear y destruir (**gestionar**) esa tarea
 - Que sean **tareas que se puedan adaptar** al crecimiento de las unidades de ejecución, siendo siempre mayor o igual a ese número (**escalabilidad**)

2. Cómo encontrar concurrencia

1. Descomposición por flujo de datos



2. Cómo encontrar concurrencia

1. Descomposición por flujo de datos

- No siempre se destaca, pero como condiciona el siguiente paso (la elección de la estructura del algoritmo), hay que tenerlo en cuenta en esta primera fase también.
- Es la opción adecuada cuando el flujo de los datos es el que ordena los grupos de tareas estableciendo un orden entre ellas.
- Es el enfoque que se sigue en todos los procesadores comerciales en el pipeline, gestionando la paralelización de instrucciones.
- Es el enfoque menos utilizado, aunque hay problemas que lo requieren, como los algoritmos de compresión.

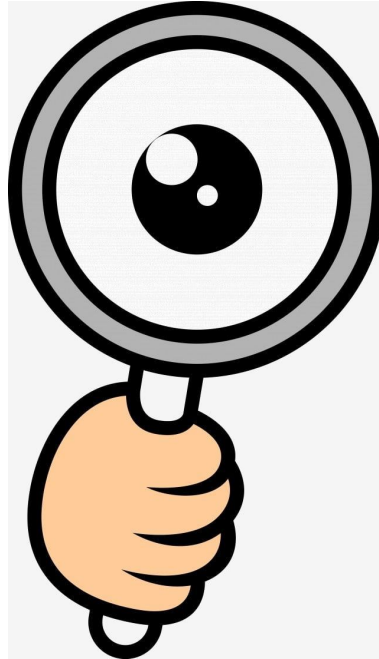
2. Cómo encontrar concurrencia

1. Descomposición del problema: Consideraciones

- Flexibilidad: la partición de tareas o datos debe **poder adaptarse** a un número variable de unidades de ejecución.
- Eficiencia: al decidir sobre partición de tareas y datos, hay que **pensar en la máquina** escogida, la **granularidad**, y el **rendimiento** que queremos obtener.
- Simplicidad: el **punto de partida** debe ser el **más sencillo posible**, así las tareas deben ser las más básicas y la partición de datos la más pequeña posible para cada hebra o proceso.

2. Cómo encontrar concurrencia

1. Descomposición del problema: Ejemplos



2. Cómo encontrar concurrencia

1. Descomposición del problema: Ejemplos Imágenes médicas PET

- La *Positron Emission Tomography* (PET) produce **imágenes** que muestran cómo se propaga una sustancia radioactiva por el paciente **para diagnosticar enfermedades**.
- Las imágenes son de baja calidad, y el cuerpo del paciente falsea su intensidad.
- Para solucionarlo: Se modela un cuerpo humano con el que corregir las imágenes crudas. Cada zona del cuerpo emitirá una radiación y se simulará la trayectoria de cada partícula.
- Este proceso se puede **paralelizar con partición de datos o de tareas...**

2. Cómo encontrar concurrencia

1. Descomposición del problema: Ejemplos Imágenes médicas PET

- Paralelización de datos:
 - Se reparte el cuerpo en partes de forma que la simulación de las trayectorias que atraviesan cada porción se asignan a una unidad de ejecución.
- Paralelización de tareas:
 - Las unidades de ejecución realizan la simulación de trayectorias completas de partículas, que son independientes entre sí.
- ¿Pros y contras de cada una?

2. Cómo encontrar concurrencia

1. Descomposición del problema: Ejemplos Imágenes médicas PET

- Paralelización de datos: La estructura de datos principal es el **modelo del cuerpo**.
 - Debe **partirse en segmentos**.
 - Los segmentos se pueden agrupar según el número de bloques deseado.
 - No hay dependencias de datos porque la estructura es de sólo lectura.
- La partición de tareas asociadas a la partición de datos será la simulación de las trayectorias que se emitan en cada uno de los segmentos creados. **Si llega al límite del segmento, ha de pasarse** al siguiente:
 - En **Memoria Compartida** pasar una trayectoria de una zona a otra implicará sincronización.
 - En **Memoria Distribuida** habrá que trasladar trayectorias antes y durante...

2. Cómo encontrar concurrencia

1. Descomposición del problema: Ejemplos

Imágenes médicas PET

- Paralelización de tareas:
 - Se selecciona un punto del cuerpo y se golpea con una partícula radioactiva.
 - Se sigue la trayectoria de dicha partícula.
 - Este proceso se repite con miles de partículas.
- Esta descomposición genera un **alto grado de paralelismo**, lo que es bueno para su implementación en cualquier máquina paralela:
 - En **Memoria Compartida** se almacena en memoria el modelo del cuerpo, y los hilos acceden a él **sin sincronización**.
 - En **Memoria Distribuida**, el coste de **hacer llegar** a todos los nodos el modelo del cuerpo puede ser problemático.

2. Cómo encontrar concurrencia

1. Descomposición del problema: Ejemplos

Producto de matrices

$$\begin{matrix} & A & & * & & B & & = & & C \\ \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} & & * & & \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} & & = & & \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} \end{matrix}$$

$$c_{1,1} = (a_{1,1} * b_{1,1} + a_{1,2} * b_{2,1} + a_{1,3} * b_{3,1})$$

(...)

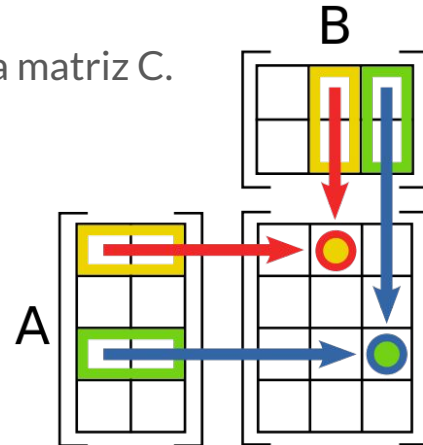
$$c_{3,3} = (a_{3,1} * b_{1,3} + a_{3,2} * b_{2,3} + a_{3,3} * b_{3,3})$$

2. Cómo encontrar concurrencia

1. Descomposición del problema: Ejemplos

Producto de matrices

- Paralelización de datos:
 - Se parte la matriz A en bloques de filas y la matriz B en bloques de columnas
- Paralelización de tareas:
 - Cada tarea será generar un elemento completo de la matriz C.
- ¿Pros y contras?

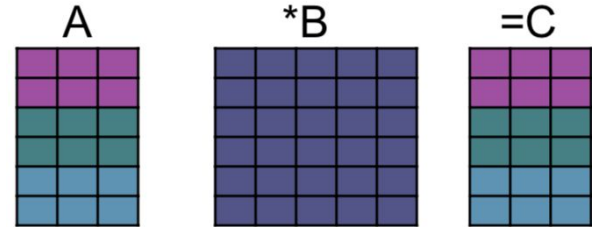


2. Cómo encontrar concurrencia

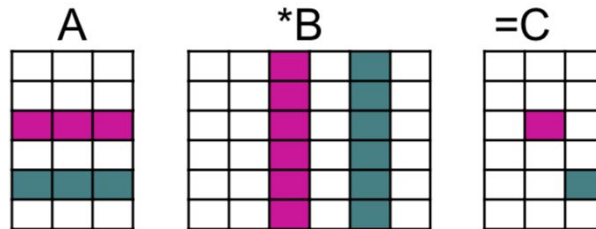
1. Descomposición del problema: Ejemplos

Producto de matrices

- Existen diversas alternativas de reparto, como:
 - Descomponer las matrices A y C en grupos de filas:



- Descomponer la matriz C y, a partir de esta división, partir A y B:



2. Cómo encontrar concurrencia

Producto de matrices

Cada componente se usa dos veces en cada Tile (2x2)

Tiling:

N

| | | | |
|-----|-----|-----|--|
| 0,0 | 0,1 | 0,2 | |
| 1,0 | 1,1 | 1,2 | |
| 2,0 | 2,1 | 2,2 | |
| 3,0 | 3,1 | 2,3 | |

M

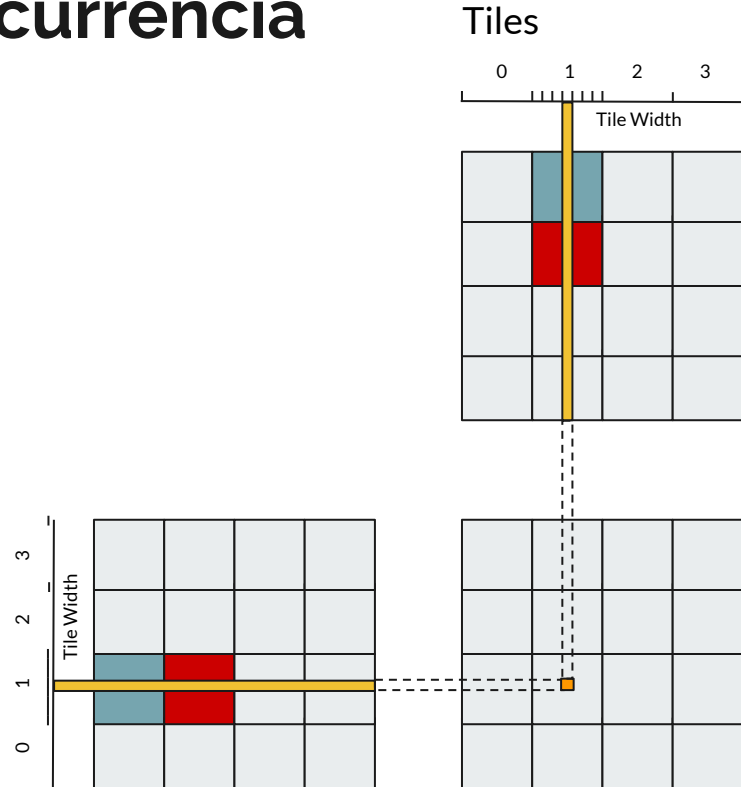
| | | | |
|-----|-----|-----|-----|
| 0,0 | 0,1 | 0,2 | 0,3 |
| 1,0 | 1,1 | 1,2 | 1,3 |
| 2,0 | 2,1 | 2,2 | 2,3 |
| 3,0 | 3,1 | 3,2 | 3,3 |

| R(0,0) | R(1,0) | R(0,1) | R(1,1) |
|-----------------|-----------------|-----------------|-----------------|
| $M(0,0)*N(0,0)$ | $M(1,0)*N(0,0)$ | $M(0,0)*N(0,1)$ | $M(1,0)*N(0,1)$ |
| $M(0,1)*N(1,0)$ | $M(1,1)*N(1,0)$ | $M(0,1)*N(1,1)$ | $M(1,1)*N(1,1)$ |
| $M(0,2)*N(2,0)$ | $M(1,2)*N(2,0)$ | $M(0,2)*N(2,1)$ | $M(1,2)*N(2,1)$ |
| $M(0,3)*N(3,0)$ | $M(1,3)*N(3,0)$ | $M(0,3)*N(3,1)$ | $M(1,3)*N(3,1)$ |

2. Cómo encontrar concurrencia

Producto de matrices

- Escalamos el problema y suponemos que cada cuadrado es una matriz de 4×4 , así estas matrices M y N serán de 16×16 .
- Se van calculando *tiles* parciales que luego se van sumando a los siguientes conforme aumenta el tile por el que recorremos la matriz
- ¿Partición por datos?
- ¿Partición por tareas?



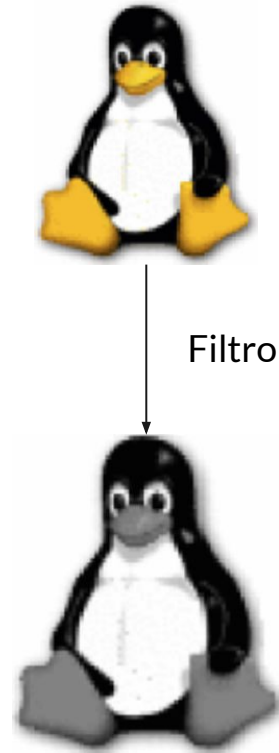
2. Cómo encontrar concurrencia

Procesamiento de imágenes

- Aplicación de un filtro a una imagen:

```
do j = 1 , Numero_Pixels_en_j  
    do i = 1, Numero_Pixels_en_i  
        Color = Imagen (i,j)  
        Imagen (i,j) = f(i,j, Color )  
    enddo  
enddo
```

- La aplicación de un filtro a un pixel no depende de los vecinos, es un **problema embarazosamente paralelo**.



2. Cómo encontrar concurrencia Procesamiento de imágenes

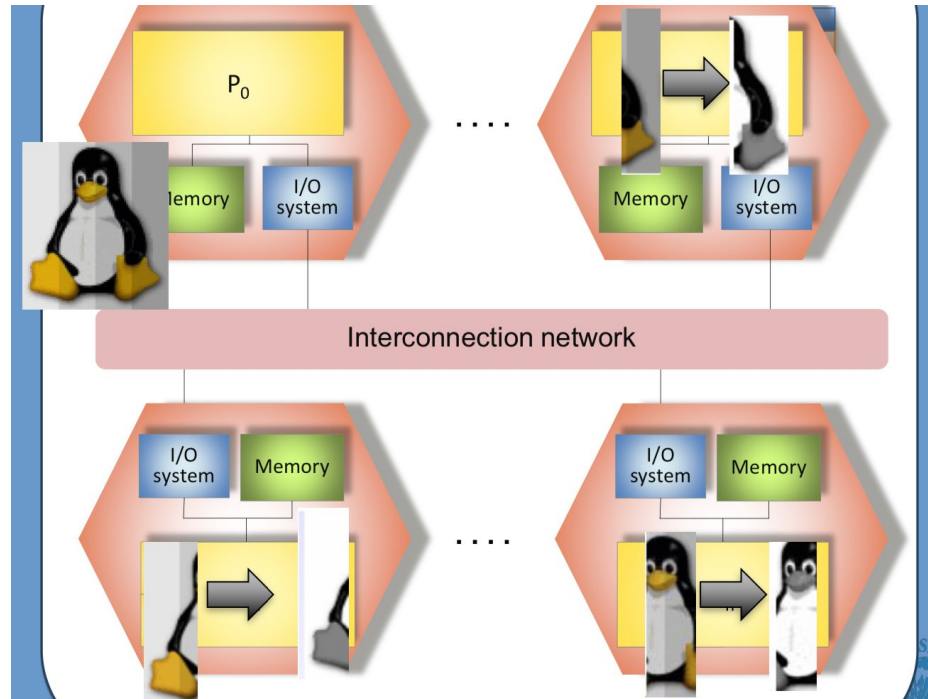
- Descomposición equitativa de la imagen (datos):

```
j1 = Mi_Primer Columna
j2 = Mi_Ultima_Columna
do j = j1 , j2
    do i = 1, Numero_Pixels_en_i
        Color = Imagen (i,j)
        Imagen (i,j) = f(i,j, Color )
    enddo
enddo
```



2. Cómo encontrar concurrencia

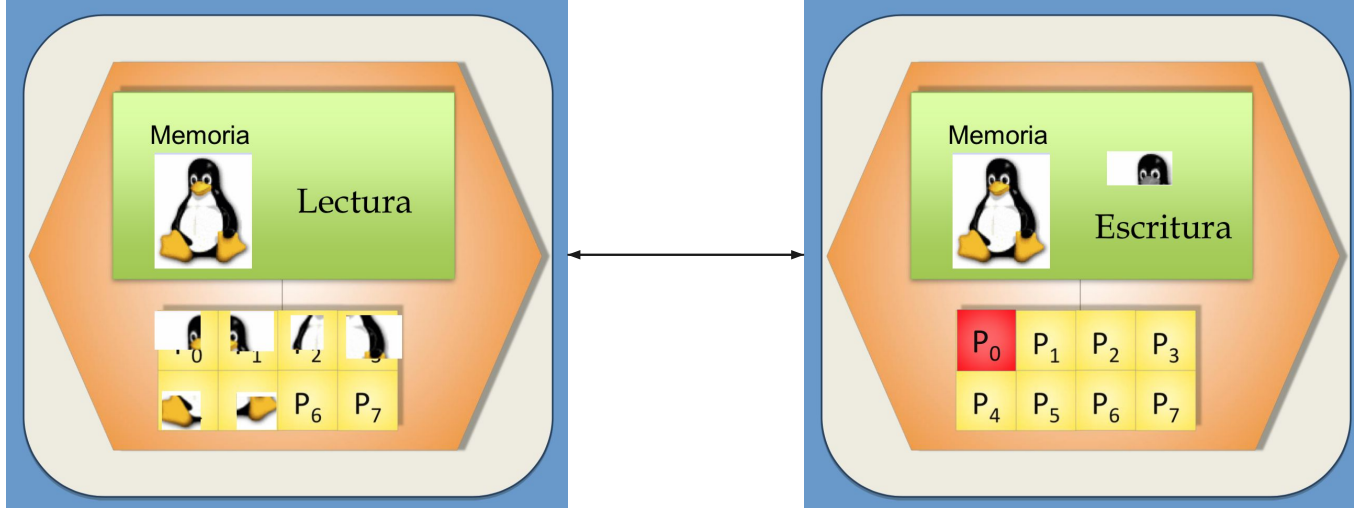
Procesamiento de imágenes



2. Cómo encontrar concurrencia

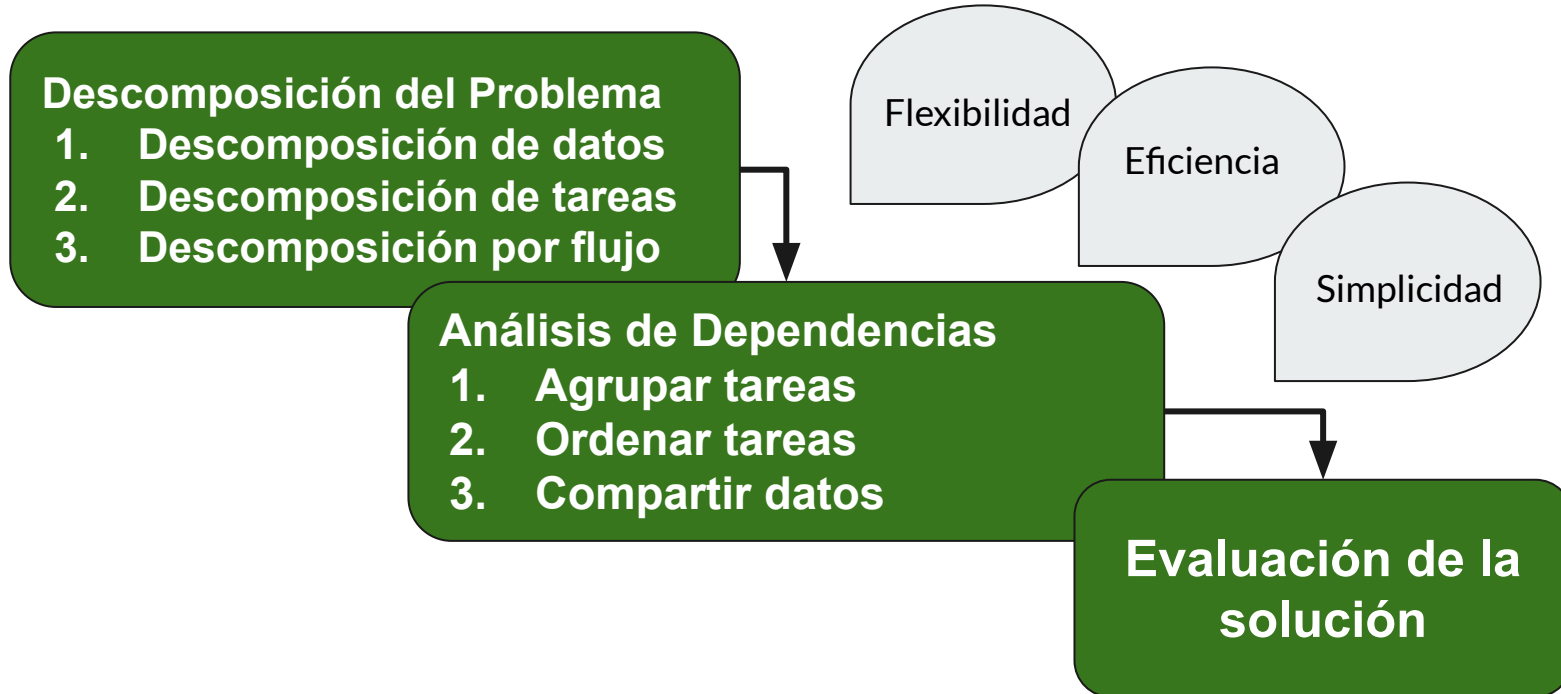
Procesamiento de imágenes

- Descomposición de tareas:
 - Cada tarea es el cálculo de un píxel resultado.



2. Cómo encontrar concurrencia

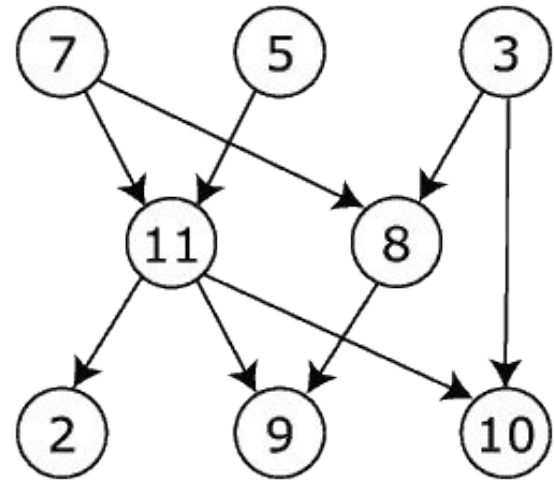
Pasos a seguir (Recordatorio)



2. Cómo encontrar concurrencia

2. Análisis de dependencias

- En esta fase hay que gestionar **dependencias** entre las partes de la **estructura de datos** que hayamos dividido o entre **las tareas** encontradas.
- Los pasos son:
 - a. **Agrupar** intentando minimizar dependencias
 - b. **Ordenar** los grupos de tareas estableciendo prioridades
 - c. **Buscar patrones** de comunicación



2. Cómo encontrar concurrencia


2. Análisis de dependencias: agrupar tareas

- ❑ La agrupación de tareas permite identificar grupos que pueden ejecutarse simultáneamente y establecer un orden parcial según restricciones comunes.

La agrupación puede ser:

- Jerarquía
- Restricciones comunes

- ❑ Todas las tareas dentro de un grupo tienen que ser independientes entre sí



2. Cómo encontrar conurrencia: 2. Análisis de dependencias: Producto de matrices (Agrupar tareas)

Tarea: Generar un valor de la matriz resultado,
Generar el sumatorio de los resultados parciales¿?

Agrupación: Todas pueden ser independientes y
ejecutarse simultáneamente

Simplicidad / Granularidad: Podría crecer
generando más de un valor por cada proceso o
hebra

Eficiencia: Se podría ajustar al número de unidades
de ejecución siendo igual al número de tiles. Se
podría hacer una partición donde se reutilizara
cada dato más de una vez

2. Cómo encontrar conurrencia: 2. Análisis de dependencias: Procesado de imágenes (Agrupar tareas)

Tarea: Calcular el píxel resultante de aplicar un filtro.

Agrupación: Todas pueden ser independientes y ejecutarse simultáneamente

Simplicidad / Granularidad: Podría crecer cómo se planifique el acceso a la imagen.

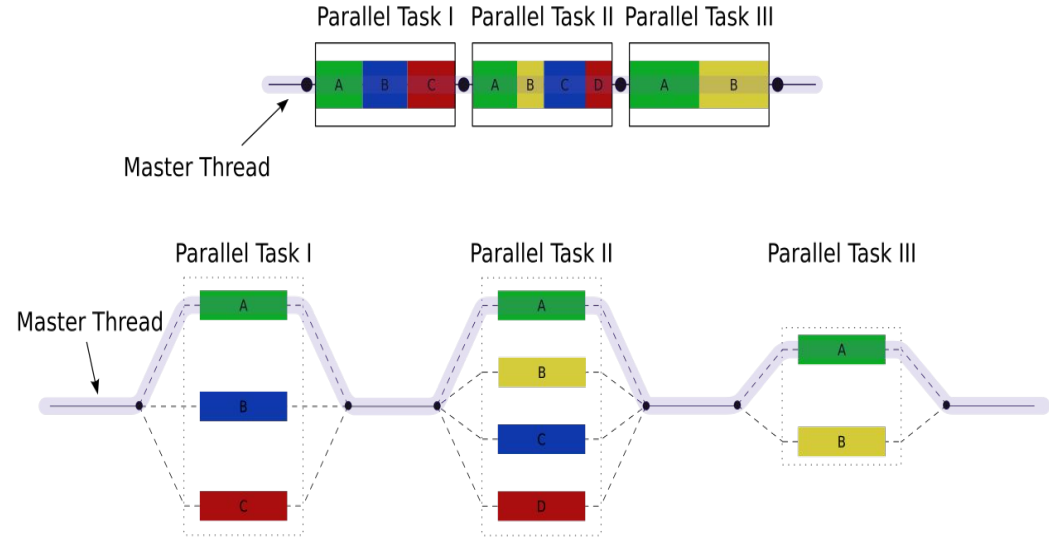
Eficiencia: Se podría ajustar al número de unidades de ejecución generando tantos grupos de tareas como unidades de ejecución.



2. Cómo encontrar concurrencia:

2. Análisis de dependencias (Ordenar tareas)

- Lo único a tener en cuenta en la ordenación de las tareas es **respetar las dependencias** y poder identificar posibles **puntos de sincronización** obligatoria.



2. Cómo encontrar concurrencia:

2. Análisis de dependencias: patrones de comunicación

- Esta fase consiste en identificar **qué datos necesita cada una de nuestras tareas.**
- Tengamos una paralelización por datos o por tareas, **todos los casos necesitarán un conjunto de datos con los que operar** y en eso debemos centrarnos.
- Los datos que cada tarea maneja se denominan *task-local* o *local-data*.

2. Cómo encontrar concurrencia:

2. Análisis de dependencias: patrones de comunicación

- Situaciones posibles:
 - Un conjunto de **tareas** que comparten el **mismo conjunto de datos**
 - Un conjunto de tareas que actualizan la **misma estructura de datos en colaboración**.
 - La misma situación anterior, pero con **dependencias entre los diferentes sectores** por vecindades, etc.

2. Cómo encontrar concurrencia:

2. Análisis de dependencias: patrones de comunicación

- Pasos a seguir:
 - Analizar qué datos o qué porción de datos necesita cada tarea y **proponer la partición**.
 - **Identificar las operaciones** que cada tarea realizará sobre sus datos y **clasificar las estructuras de datos según**:
 - Read only: Son de sólo lectura
 - Local: Será una porción de datos que sólo actualiza una tarea y no hay acceso compartido.
 - Read-Write: Si más de una tarea lee y escribe en la estructura, hay que establecer sincronización en los accesos de escritura

2. Cómo encontrar concurrencia:

2. Análisis de dependencias: patrones de comunicación

- Tipos de estructuras *Read-Write*:
 - Accumulative: Para realizar reducciones o acumulaciones de resultados parciales. En este caso, cada tarea posee una copia local de la estructura y opera con ella, hasta que hay que hacer la recopilación final en una sola estructura global.
 - Multiple-Read/Single-Write: Todas las tareas leen la estructura pero cada una escribe solo en su parte.

2. Cómo encontrar concurrencia:

3. Evaluación de la solución

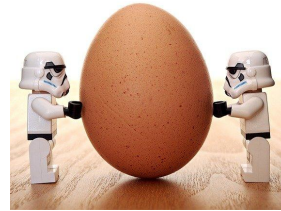
- En este paso se **revisarán las decisiones tomadas** buscando posibles problemas que no se hayan tenido en cuenta, como intercambio de datos, datos en los bordes que se tendrán que duplicar, etc.
- Los puntos a abordar son:
 - **Asegurarse** de que cada **tarea** se puede realizar **de forma simultánea**.
 - **Que hemos identificado** correctamente **los datos** que pueden ser **locales** a cada tarea.
 - **Que se satisfacen** todas las **dependencias** temporales establecidas.

2. Cómo encontrar concurrencia:

3. Evaluación de la solución

- Siguiendo con las comprobaciones:
 - Que el enfoque de **partición** es adecuado para la máquina donde se ejecutará.
 - Que hemos respetado las reglas de **simplicidad, eficiencia y flexibilidad**.
 - Y ya es hora de plantearse otras cosas como:
 - ¿Las **tareas** son **regulares**? ¿realmente necesitamos **sincronización**?
 - ¿Podríamos **agrupar** tareas de otra forma?

Trabajo para la próxima semana



Cada grupo debe escoger un problema computacionalmente costoso, explicarlo, y analizar cómo se podría paralelizar con una división por datos y otra por tareas.

Pensad en las estructuras de datos necesarias y de qué tipo sería su acceso.

=> Un miembro del grupo presentará su trabajo la próxima semana. Habrá que entregar tanto la presentación como una transcripción de lo que se dice (incluyendo las fuentes consultadas).

Importante: Poneos de acuerdo entre grupos para no escoger el mismo problema.



Gracias.

