# Tema 6: Introducción a los Algoritmos Paralelos

El alumno clasificará los elementos a considerar en el diseño y análisis de algoritmos paralelos vs algoritmos seriales para su programación.

# 6.0 Antecedentes

- “Si uno es bueno, entonces 10 o 10’000 debe ser mejor”
- “Si un hombre construye una alberca en 10 días, dos hombres la construyen en 5” …
- y 4? y 8?

# 6.0 Antecedentes

- Ideas como ésta, dieron origen al procesamiento paralelo.
- Los científicos y teóricos de la computación, han aceptado al paralelismo como una manera novedosa de solucionar problemas de programación


GERENTE DE PROYECTOS
Aquella persona que piensa que 9 mujeres juntas pueden hacer un bebé en 1 mes

# 6.0 Antecedentes

- El principal uso de la programación paralela se centra en:

Desarrollo de nuevos modelos de investigación

Inteligencia Artificial

Aplicaciones en la medicina

Simulaciones

# 6.0.1 Proceso

- La mayoría de los autores acepta que un programa en ejecución es un proceso.
- Un proceso es cualquier secuencia de operaciones que están ejecutándose en memoria activa, que realiza una o varias acciones sobre ciertos datos
- Permite describir el comportamiento de un objeto mediante las acciones que es determinado a realizar.

# 6.0.1.2 Tipos de Procesos

- Procesos disjuntos
  - ✓Se ejecutan en diferentes bloques de un programa
  - ✓No tienen posibilidad de acceder entre sí (no se comunican)
- Procesos cooperativos.
  - ✓ Su objetivo es cooperar para la realización de una tarea
  - ✓ Comparten recursos en común
  - ✓ Requieren alguna forma de sincronización.

# 6.0.2 Hilos

Al ejecutarse un programa secuencial, sigue una sola “hebra de control”, inicia con una operación atómica y avanza a través del proceso conforme las operaciones atómicas se van ejecutando.

# 6.0.3 Programación…

- **Programación concurrente, paralela y distribuida.**

Un programa concurrente, paralelo o distribuido se especifica por dos o más procesos que se ejecutan simultáneamente en un tiempo determinado, y que cooperan entre sí para realizar una tarea.

La ejecución de un programa concurrente, paralelo o distribuido resulta en el seguimiento de múltiples hebras de control que se comunican entre sí

# 6.0.3.1 Programa concurrente

- Un programa concurrente es aquel formado por varios procesos que se ejecutan en un solo procesador siguiendo un esquema de paralelismo temporal.

# 6.0.3.2 Programa paralelo

- Un programa paralelo es aquel formado por varios procesos que se ejecutan en varios procesadores conectados entre sí mediante una red de comunicación.
- Siguiendo un esquema de paralelismo espacial, el cual se expresa físicamente en varios procesadores que operan simultáneamente.

# 6.0.3.3 Programa Distribuido

- Un programa distribuido comparte la definición de uno paralelo, la diferencia radica en que la red de comunicación se extiende a dispositivos ubicados en diferentes puntos geográficos

# 6.0.4 Ventajas

- La razón de existir de la programación paralela es encontrar una mejora con respecto a la programación secuencial; sus principales ventajas son las siguientes:
  - Aumento De Velocidad
  - Eficiencia en el procesamiento
  - Simplicidad en la expresión de modelos
  - Aplicaciones
  - Tamaño y costo

# 6.0.5 Desventajas

- Algunas de las desventajas, han inhibido el uso de sistemas paralelos ya que en algunas ocasiones el paralelismo se puede complicar en extremo.

- Las principales desventajas son las siguientes:

# 6.0.5 Desventajas

- Dificultad
  - Principalmente en la comunicación y sincronización.
  - Resultados esperados.
- Eficiencia en la programación
- Problemas de HW y SW
  - Relación costo beneficio
  - Existen muchos algoritmos que no son paralelizables

# 6.0.6 Organización de la memoria

Durante la ejecución de un programa paralelo, los procesadores comparten recursos se comunican entre sí.

Por su interacción se pueden clasificar en fuertemente acoplados y débilmente acoplados.

Existen dos  tipos de mecanismos disponibles para la organización de la memoria

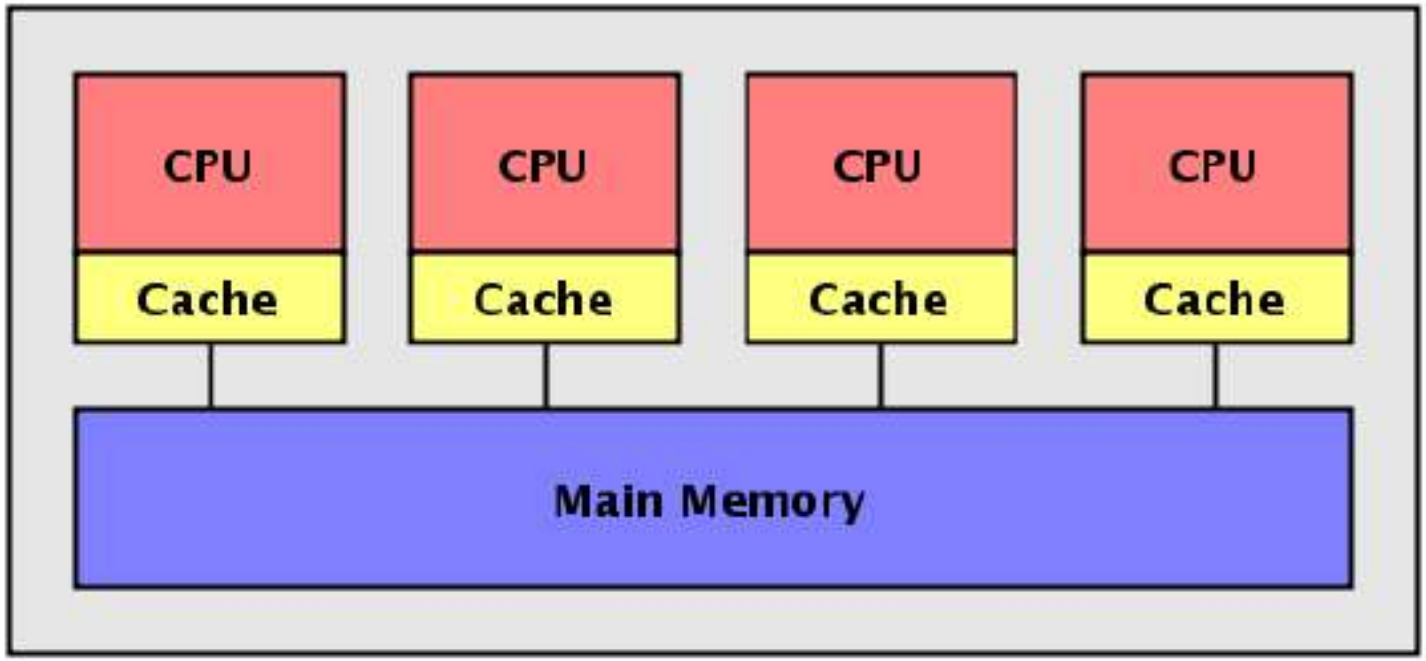
# 6.0.6.1 Memoria compartida

Permite el acceso de cualquier procesador del sistema a cualquier localidad de memoria común.

La red de conexión es controlada por un sistema de HW que no es visible al programador

Cada dirección de memoria es única e idéntica para cada procesador.

# 6.0.6.1 Memoria compartida



# 6.0.6.2 Memoria Distribuida

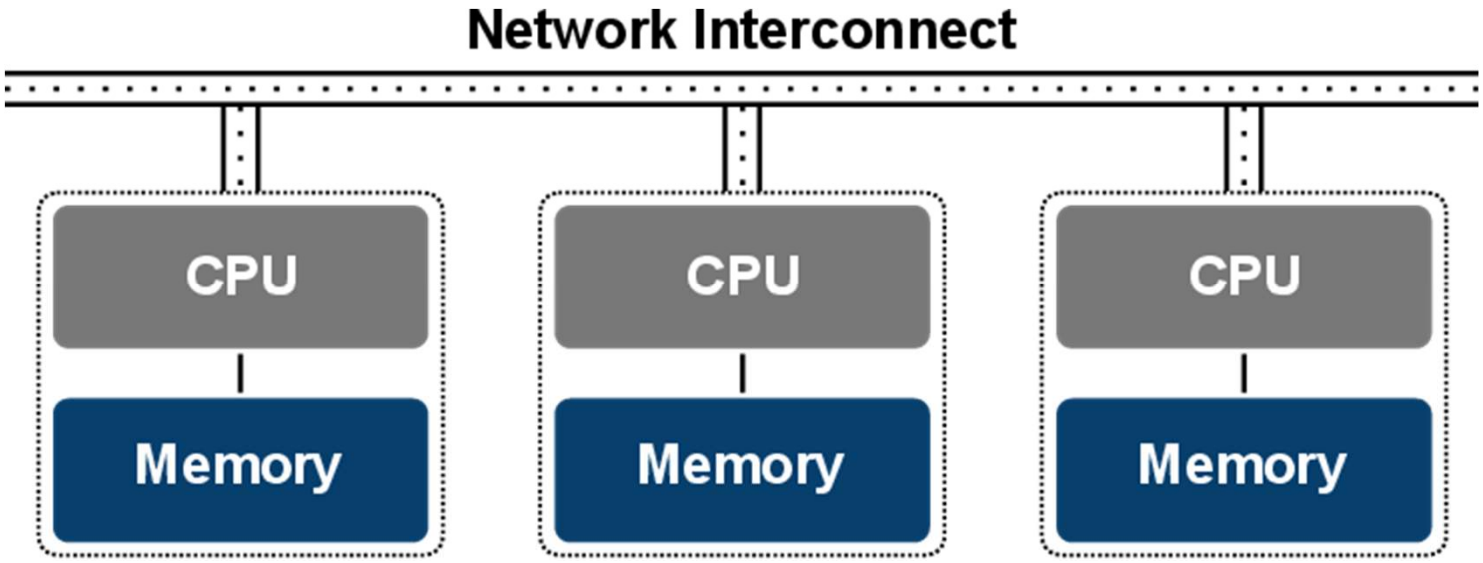
- Memoria distribuida.

Cada procesador utiliza su propia memoria privada comunicándose con otros procesadores mediante una red.

La red se forma basada en alguna topología.

Cada procesador reconoce la diferencia entre su memoria local y la memoria remota de otro procesador

# 6.0.6.2 Memoria Distribuida



# Ejercicio # ?

# Formas de comunicación

| Memoria Compartida | Memoria Distribuida |
| --- | --- |
| • Semáforos<br>• Regiones críticas<br>• Monitores | • Canales<br>• Llamadas a procedimientos remotos |

# Granularidad

- Es un indicador de la cantidad de trabajo que cada procesador puede realizar de manera independiente.
- Una aplicación de granularidad gruesa se divide en partes lógicas de procesos de secuencias independientes con poca sincronización o comunicación.

# Granularidad

- En una aplicación de granularidad fina, los procesadores ejecutan unas cuantas instrucciones y aumenta la cantidad de comunicaciones entre procesadores.
- La relación de tiempo de procesamiento entre tiempo de ejecución es baja

# Granularidad

- La finalidad de la granularidad es determinar la mejor forma de dividir el problema en conjuntos de tareas, de tal manera que el tiempo de ejecución de cada conjunto sea mínimo.
- Si la granularidad es gruesa, el nivel de paralelismo disminuye, si la granularidad es muy fina, puede existir una pérdida de tiempo de ejecución por un aumento en los tiempos de comunicación entre procesadores

# Balance de carga

- El balance de carga se refiere a la manera de repartir el total de las tareas a realizar entre el total de los procesadores disponibles.
- El balance de carga se divide en 2 grandes rubros.
  - ✓Estático
  - ✓Dinámico

# Clasificación de Flynn

- **SISD**
- **SIMD**
- **MISD**
- **MIMD**

**TAREA:** Describir cada uno e indicar 2 ejemplos de problemas que corresponden a esa categoría

# 6.1 Clasificación de Paralelismo

- Paralelismo Algorítmico

Se basa en la paralelización de tareas cuasi-independientes que ejecutan secciones del algoritmo.

Se relaciona con el modelo de flujo de datos ya que el procesamiento se conduce a partir de la disponibilidad de los datos ignorando el orden de las instrucciones

# 6.1 Clasificación de Paralelismo

- Paralelismo Geométrico

Se basa en la paralelización de tareas independientes.

Consiste en dividir los datos del problema de una manera simétrica considerando una distribución uniforme, siendo cada proceso responsable de una región espacial de datos.

# 6.1 Clasificación de Paralelismo

- Paralelismo “Farm” (Manager-Workers)

Se forma por tareas totalmente independientes e idénticas  que procesan datos en cualquier orden.

Aplica a problemas cuya solución puede descomponerse en pequeñas partes independientes entre sí.

Cada parte puede ejecutarse aisladamente.

# 6.1 Clasificación de Paralelismo

- Debido a la independencia de los elementos que trabajan (procesadores) para obtener la solución global se requiere de un coordinador que se encarga de distribuir el trabajo.
- Provee un mecanismo de balance dinámico basado en la demanda de trabajo, esto es, que las tareas se pasan a los procesadores según se requieren

# 6.2 Métricas de Desempeño

- En el desempeño de los sistemas paralelos, en términos de recursos, el tiempo ha sido su mayor ventaja.
- En la práctica, se ha demostrado una mejora considerable en la ejecución de un programa paralelo respecto a su contraparte secuencial
- Algunas de las métricas más importantes son las siguientes:

# 6.2 Métricas de Desempeño

- Tiempos de procesamiento y de comunicación

El principal propósito de un programa paralelo es disminuir el tiempo de ejecución.

Este tiempo puede subdividirse entre el tiempo de procesamiento y el tiempo de comunicación.

Una aplicación paralela trivial es aquella que no requiere comunicación entre procesadores, su tiempo de procesamiento sería lineal.

# 6.2 Métricas de Desempeño

- Speedup

Es una relación del tiempo de ejecución de un programa ejecutándose en un solo procesador sobre el tiempo de ejecución del mismo programa ejecutándose en n procesadores

$$S(n) = \frac{T(1)}{T(n)}$$

# 6.2 Métricas de Desempeño

El Speedup solo considera aspectos temporales, no toma en cuenta otros como la topología elegida, el balance de carga, la granularidad, etc.

# 6.2 Métricas de Desempeño

- Eficiencia

La medida de eficiencia se asocia a la idea que n trabajadores deben hacer el trabajo en una fracción 1/n del tiempo que le lleva a un solo trabajador.

$$E(n) = \frac{T(1)}{n \cdot T(n)} = \frac{S(n)}{n}$$

El valor de la eficiencia refleja ésta al aprovechar los recursos de HW del sistema, por lo tanto siempre será menor a 1

# 6.2 Métricas de Desempeño

- Fracción Serial

Trata de tomar en cuenta diversos factores y no solo el tiempo:

$$f = \frac{\frac{1}{S} - \frac{1}{n}}{1 - \frac{1}{n}}$$

# 6.3 Niveles de Paralelismo

- Paralelismo a nivel de instrucción.

Tareas independientes se pueden ejecutar en diferentes procesadores.

Las tareas que no son independientes pueden tener 3 tipos de dependencias

**Dependencia de Flujo:** Cuando una tarea forzosamente requiere datos de otra para continuar

**Anti-dependencia:** Es el proceso inverso, cuando una tarea I1 utiliza un registro o variable que se obtiene en I2

# 6.3 Niveles de Paralelismo

Dependencia de salida: Si dos tareas diferentes utilizan el mismo registro o valor para almacenar un resultado.

| | | |
|---|---|---|
| $I_1$: $R_1 \leftarrow R_2+R_3$ | $I_1$: $R_1 \leftarrow R_2+R_3$ | $I_1$: $R_1 \leftarrow R_2+R_3$ |
| $I_2$: $R_5 \leftarrow R_1+R_4$ | $I_2$: $R_2 \leftarrow R_4+R_5$ | $I_2$: $R_1 \leftarrow R_4+R_5$ |
| flow dependency | anti dependency | output dependency |

# 6.3 Niveles de Paralelismo

- Paralelismo de datos

Los elementos de la estructura de datos son distribuido equitativamente entre los procesadores.

Cada procesador, realiza la operación de sus elementos asignados.

# 6.3 Niveles de Paralelismo

- Paralelismo a nivel de ciclo (Loop-Parallelism)

Si no hay dependencias entre las iteraciones de un loop, las operaciones se pueden realizar en orden arbitrario y en paralelo por diferentes procesadores

# 6.3 Niveles de Paralelismo

- Paralelismo Funcional

Implica que un programa contenga diversos tipos de instrucciones y que unos conjuntos determinados se puedan paralelizar con otros.

Este tipo de paralelismo implica una calendarización para resolver posibles dependencias entre conjuntos de instrucciones.

# 6.4 Diseño de programas paralelos

- El diseño y el desarrollo de programas paralelos se ha caracterizado por ser un proceso manual, el programador es responsable de identificar y de implementar el paralelismo.
- Se han creado herramientas para facilitar al programador la creación de programas paralelos.
- La herramienta más común es utilizar un compilador o un pre-procesador paralelo.

# 6.4 Diseño de programas paralelos

- Un compilador paralelo puede trabajar de dos maneras

a) Completamente automática

Analiza el código fuente e identifica las "oportunidades" de paralelismo.

El análisis incluye la identificación de elementos que pueden interferir en la paralelización

El paralelismo automático de loops

# 6.4 Diseño de programas paralelos

b) Dirigido por el programador

Utilizando instrucciones específicas para hacer paralelas ciertas regiones del programa.

En ambos casos se deberá identificar el tipo de memoria a utilizar y la forma de comunicación y sincronización de elementos

# ¿Cómo se hace?

- Identificar si el problema es paralelizable o no.
- Identificar los puntos de acceso del programa
  - Dónde se hacen las operaciones esenciales
  - Dónde están las secciones paralelizables
- Encontrar posibles cuellos de botella
  - ¿Existen procesos más tardados que otros?
  - ¿Existen procesos dependientes?
  - ¿Es posible reestructurar estos procesos?
- Eliminar dependencia de datos

# 6.4 Diseño de programas paralelos

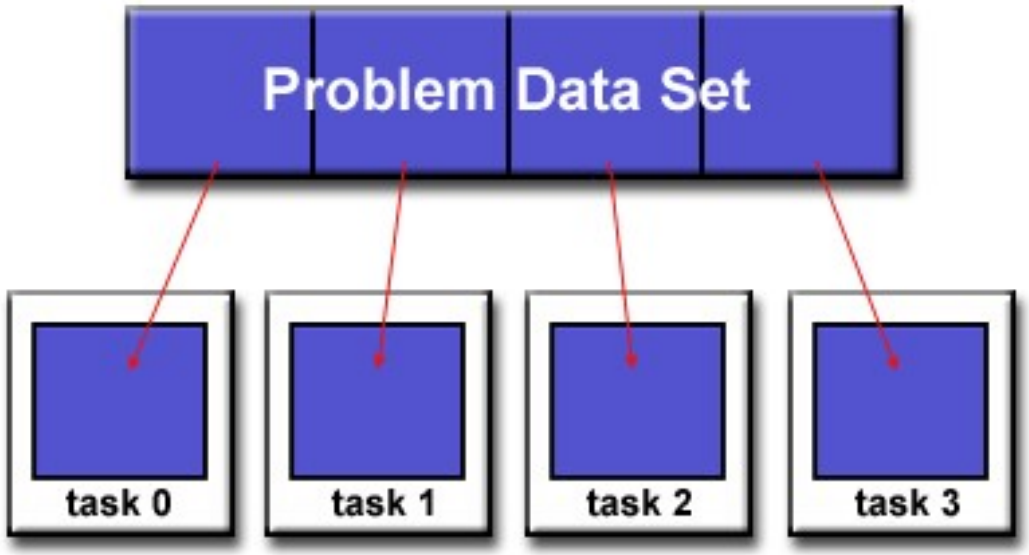
- Ian Foster propone el modelo PCAM para la realización de programas paralelos
- Partición
- Comunicación
- Aglomeración
- Mapeo

# Partición

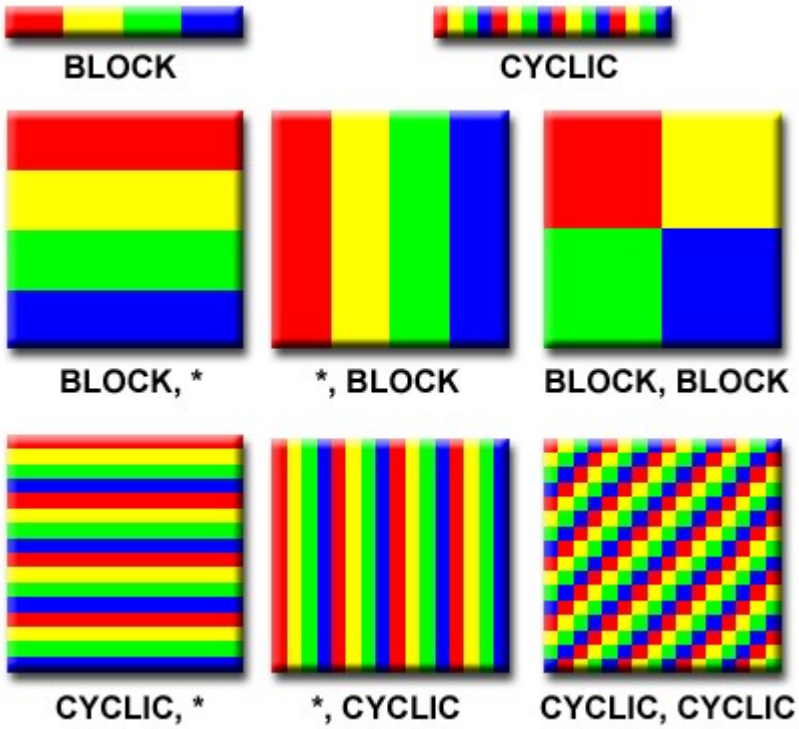
- El programa o algoritmo que se realiza y los datos que va a operar se descomponen en tareas.
- En esta etapa se ignoran aspectos como el numero de procesadores, la organización de la memoria, etc.
- Únicamente se especifica en encontrar la mejor manera de dividir el problema en tareas y en encontrar las “oportunidades de paralelismo”

# Partición

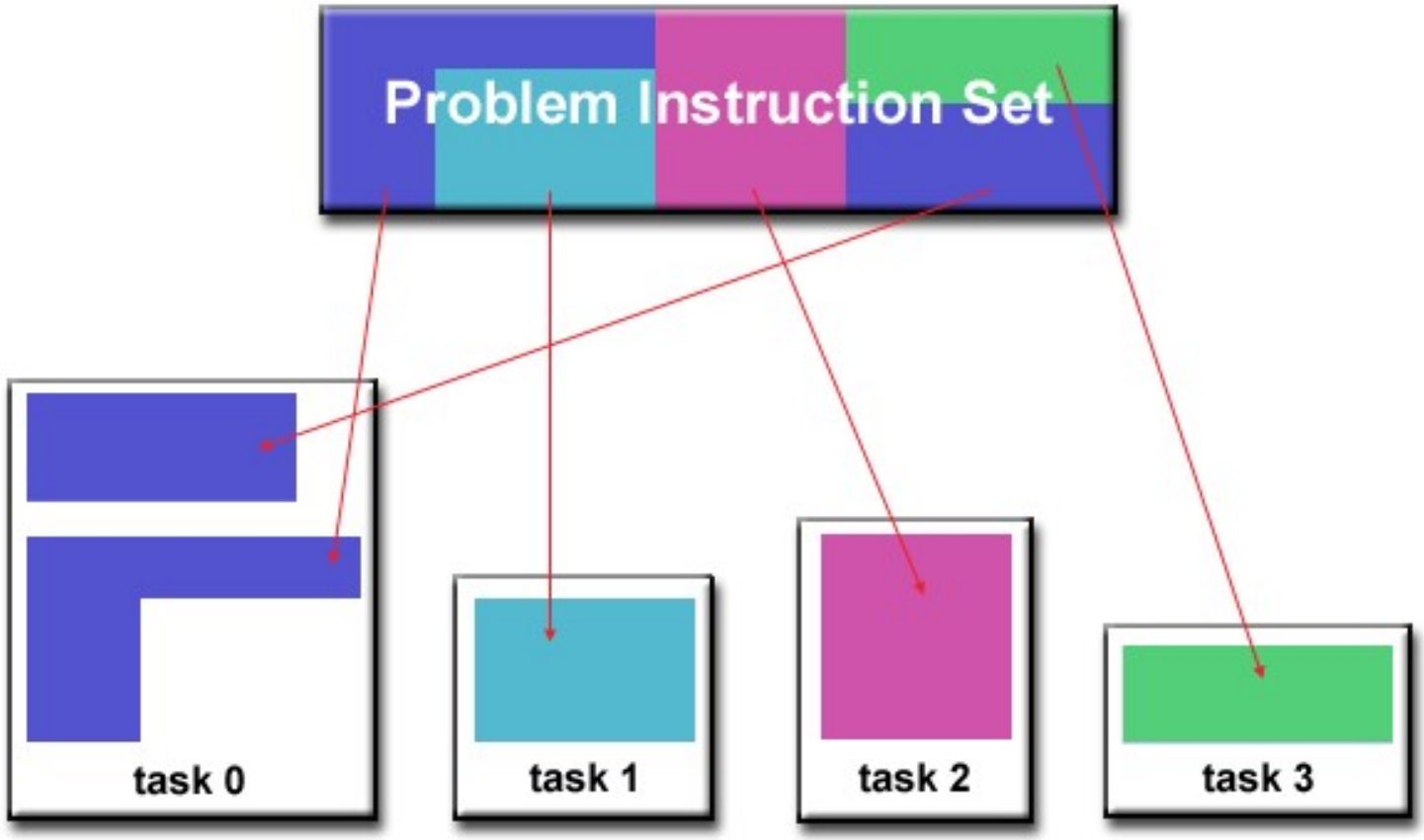
- Partir el problema en sub-problemas también se conoce como descomposición



# Partición


BLOCK
CYCLIC
BLOCK, *
*, BLOCK
BLOCK, BLOCK
CYCLIC, *
*, CYCLIC
CYCLIC, CYCLIC

# Partición



# Comunicación

- Se especifica en identificar la comunicación requerida para coordinar la ejecución de tareas, así como la comunicación apropiada para las estrcuturas y algoritmos
- Identificar la relación de orden de las tareas y las dependencias de datos que existan entre unas y otras

# Comunicación

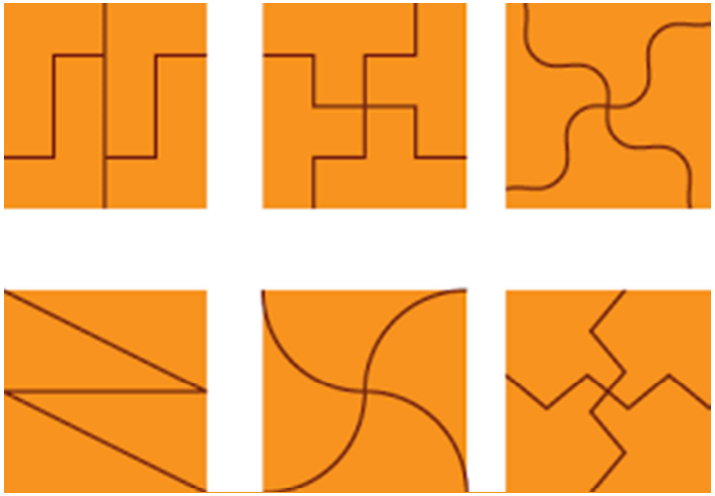
- Los tipos de comunicación que puede haber entre procesos.
  - Local/Global
  - Estructurada (tipo de topología)
  - Estática / dinámica
  - Síncrona /asíncrona.

# Aglomeración

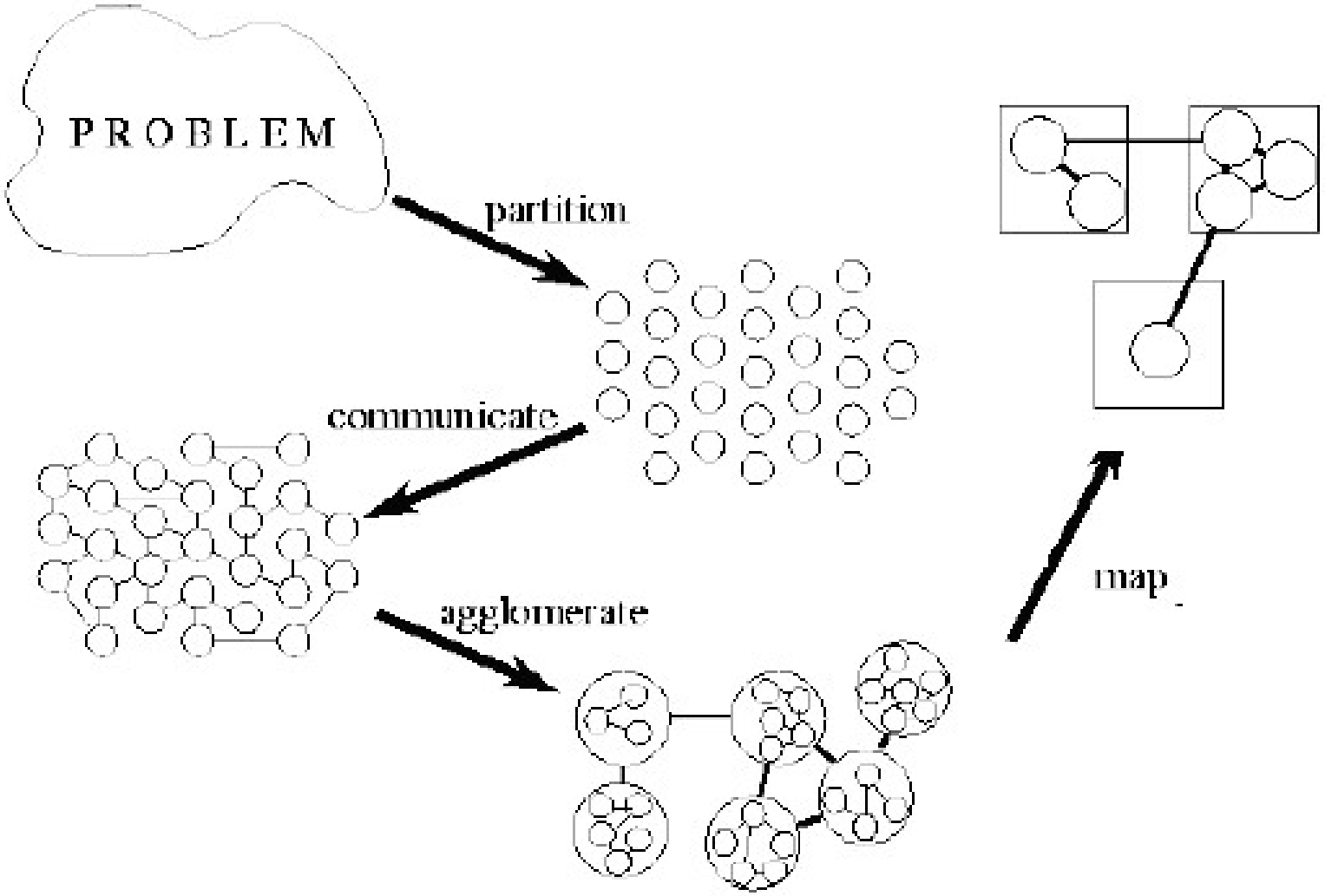
- Una vez definidas las dependencias de datos, de comunicación y del orden de las tareas, se busca hacer el proceso inverso a la partición, es decir, volver a juntar aquellas tareas que se pueden ejecutar de manera simultánea sin afectar al sistema en general

# Mapeo

- Asignar los bloques de tareas a los procesadores (o a los hilos) que conforman el sistema paralelo.



# PCAM…



# 6.5 Lenguajes de Programación Parlaleos

Un lenguaje paralelo puede ser descrito por su capacidad de expresar las siguientes características:

- Paralelismo
- Secuencialidad
- Comunicación y sincronización
- No determinismo

# 6.5.1 Paralelismo

- Ejecución paralela de procesos
- Cada lenguaje de programación requiere una o varias “instrucciones paralelas” las cuales representan la activación simultánea de un conjunto de instrucciones que generan procesos disjuntos
- La instrucción paralela finaliza exitosamente cuando todos los procesos terminan con éxito.

# 6.5.2 Secuencialidad

- Dentro de la programación paralela es necesario representar la secuencialidad de forma expresa. con el fin de contrastar su acción con la ejecución paralela

# 6.5.3 Comunicación y Sincronización

- Cuando se realiza la comunicación entre procesos, normalmente lo que se busca es modificar variables de un proceso a través de expresiones de otro.
- P1 envía a P2
- P2 recibe e identifica que el emisor es P1
- el tipo de var en P2 debe ser el mismo que la expresión realizada en P1
- Mientras se hace el proceso tanto p1 como p2 establecen mecanismos para no corromper las variables afectadas

# 6.5.4 No determinismo

- En la programación paralela, un proceso tiene la capacidad de intercambiar mensajes con otro u otros procesos.
- El orden en el que se reciben los mensajes es totalmente arbitrario.
- En ocasiones es necesario dar una prioridad a ciertos procesos de control, por lo que se implementan variables booleanas conocidas como comandos custodiados

# 6.6 Paralelismo y Orientación a Objetos

- La programación paralela orientada a objetos se basa en la idea de objetos que representan entidades que componen el dominio del problema, cooperan entre sí intercambiando información mediante mensajes.
- Se representa mediante mecanismos de sincronización y comunicación entre procesos paralelos como el paso de mensajes.

# 6.6 Paralelismo y Orientación a Objetos

- Cada objeto posee una existencia simultánea tanto en tiempo como en espacio (inter-objeto)
- Se permite a los objetos manejar varias acciones o mensajes concurrentemente (intra-objeto)

# 6.6 Paralelismo y Orientación a Objetos

- Tareas Paralelas

Un objeto se trata de una estructura que ejecuta el programa como una colección de tareas paralelas controladas de manera explícita.

Bloques paralelos

Primitivas de espera

Mecanismos de suspensión

Serialización

# 6.6 Paralelismo y Orientación a Objetos

- Paralelismo de datos

Un objeto realiza en paralelo tareas estructuradas respecto a los datos que manipulan.

Frecuentemente, el compilador (paralelo) se encarga de realizar los ajustes necesarios para organizar las operaciones sobre conjuntos de datos definidos

# 6.6 Paralelismo y Orientación a Objetos

- Objetos comunicantes

El modelo más aceptado en la OOPP es el modelo de objetos comunicantes.

Cada objeto representa una unidad de procesamiento autónoma que se comunica activamente con los demás.

Se identifican objetos emisores y receptores

# 6.6 Paralelismo y Orientación a Objetos

Los objetos receptores son semejantes a los de la programación secuencial, constan de un conjunto de datos y métodos.

Los objetos emisores contienen un proceso que permite ejecutar sus funciones en paralelo con el resto del programa.

Un programa de objetos comunicantes se observa como un conjunto de objetos emisores que se ejecutan en diferentes procesadores con un conjunto de objetos receptores que son modificados.

# 6.6 Paralelismo y Orientación a Objetos

Descomposición de actividades.

Las unidades de programación se identifican claramente al descomponer un programa en un conjunto de objetos interactivos.

Transparencia en sincronización

La sincronización se establece en el diseño de los objetos y no se tiene que establecer a bajo nivel

# 6.6 Paralelismo y Orientación a Objetos

## Multigranularidad

Existe interacción entre objetos de diversas granularidades, pueden cohabitar y cooperar entre sí en el mismo sistema lo que permite la descomposición de un diseño complejo en varios niveles