# Tema 1

## ¿Qué es el paralelismo?

Combinar la potencia de cálculo de varios procesadores para aumentar el rendimiento

## Tipos de paralelismos

- Implícito: transparente al usuario (pipelining, multithreading…)
- Explícito:
    - Hardware
        - Múltiples procesadores
        - Múltiples memorias
        - Redes de interconexión
    - Software
        - SSOO paralelos
        - Programas orientados a la concurrencia

## Plataformas para procesamiento paralelo

- Organización lógica: Visión que tiene el programador de la máquina, desde el punto de vista del software del sistema
- Organización física: La arquitectura hardware real.
- Arq. Física, hasta cierto punto, independiente de la Arq. Lógica

## Organización lógica – Modelo de control

### Taxonomía de Flynn:

- SISD (Single Instruction Single Data)
- MISD (Multiple Instruction Single Data)
- SIMD (Single Instruction Multiple Data)
- MIMD (Multiple Instruction Multiple Data)

## Organización lógica – Modelo de comunicación

Dos maneras distintas de intercambiar información entre tareas paralelas:

- Paso de mensajes
- Espacio de memoria compartida

### Paso de mensajes (Multicomputadoras)

La comunicación se produce a través de mensajes entre el procesador emisor y el receptor

Estándares: MPI (Message Passing Interface), PVM (Parallel Virtual Machine).

### Espacio de memoria compartida (Multiprocesadores)

- UMA. Acceso a memoria uniforme
- NUMA. Acceso a memoria no uniforme

Estándares: Posix, OpenMP.

## Organización Física:

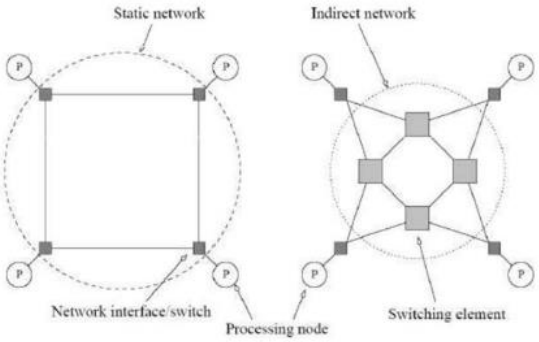
### Redes de interconexión (RICs):

Proporcionar conexión entre los distintos procesadores y memorias del sistema

### Tipos

Estática. Enlaces punto a punto (conectan nodos).

Dinámica. Formada por elementos de conmutación (conectan nodos o bancos de memoria)




estática dinámica

### Métricas de evaluación para RICs

- Diámetro (Cuanto más pequeño mejor)
- Conectividad (Cuanto más grande mejor)
- Ancho de bisección (Cuanto más grande mejor)
- Ancho de banda de bisección (Cuanto más grande mejor)
- Coste (Cuanto más pequeño mejor)

# Tema 2

## Proceso

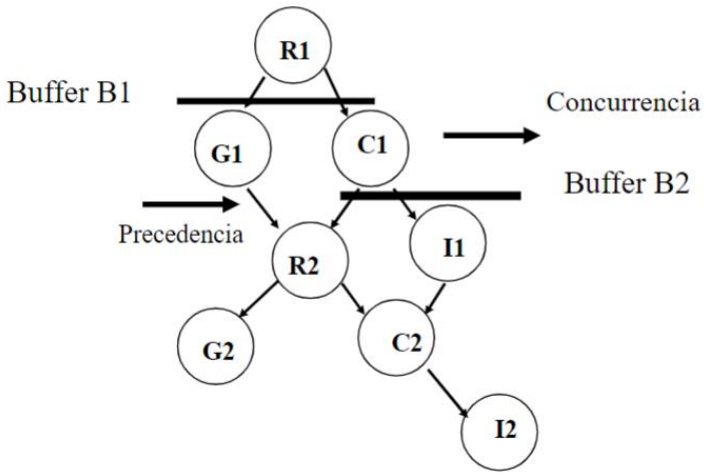
Unidad más pequeña de trabajo individualmente planificable por un SO. Si estamos hablando de hilos o threads ésta será la Unidad más pequeña planificable.

### Ventajas

- Ganancia de Velocidad
- Uso de dispositivos de E/S que tienen latencia
- Conveniencia para el Usuario
- Multiprocesamiento
- Computación distribuida

### Diagrama de precedencia

Descomposición y ejecución concurrente para un mismo resultado. Dado que el programa original es cíclico, cada parte también lo es.




### Puntos importantes para el diseño de un SO

- Multiprogramación con un solo procesador. Apariencia de ejecución simultánea.
- Multiprocesador. Verdadera ejecución simultánea.
- Procesamiento distribuido. Paso de mensajes.

*La concurrencia juega un papel fundamental*

*En un sistema monoprocesador intercalar la ejecución de procesos aumenta la eficacia.*

*En un sistema multiprocesador además del intercalado se permite el solapamiento.*

*La concurrencia abarca varios aspectos*

- Comunicación entre procesos
- Compartición de recursos
- Sincronización de procesos
- Reserva del procesador para los procesos

### Procesos dependientes de forma indirecta

Procesos que tienen algún objeto en común pero no son conscientes de ello

### Procesos dependientes de forma directa

Procesos que se comunican con otros procesos y que pueden ser diseñados para trabajar conjuntamente

## Problemas asociados a la concurrencia

Los procesos concurrentes suelen compartir datos y recursos

Si el acceso a estos datos o recursos no se controla se puede tener inconsistencia

La velocidad de ejecución de los procesos no se puede predecir, con lo que pueden existir problemas al compartir datos

### *Competencia entre procesos y recursos*

#### Inanición

- Tres procesos necesitan un recurso
- El SO da acceso alternativamente a P1 y P3
- P2 queda bloqueado indefinidamente

#### Interbloqueo

- Los dos procesos necesitan dos recursos y el sistema le asigna uno a cada uno
- No liberan el que tienen asignado hasta que se libere el otro

Secciones Críticas >>> Indeterminismo-Soluciones no coherentes

### *Problema de la sección crítica*

Cuando un proceso accede a un dato compartido decimos que éste es un recurso crítico y la parte de programa que lo usa se llama sección crítica

La ejecución de secciones críticas debe ser mutuamente exclusivas: en cualquier instante sólo un proceso puede ejecutar la sección crítica

Todos los procesos necesitan un “permiso”

### *Estructura:*

- Sección de Entrada
- Sección Crítica
- Sección de Salida

## Exclusión Mutua

### Requisitos

- Sólo un proceso debe tener permiso para entrar en la sección crítica por un recurso en un instante dado
- No puede permitirse el interbloqueo o la inanición
- Cuando ningún proceso está en su sección crítica, cualquier proceso que solicite entrar en la suya debe poder hacerlo sin dilación
- No se deben hacer suposiciones sobre la velocidad relativa de los procesos o el número de procesadores
- Un proceso permanece en su sección crítica solo por un tiempo finito

## Tipos de soluciones

### *Software*

### Decker

La variable compartida “turno” se inicializa a “i” o a “j” antes de ejecutar Pi

La sección crítica de Pi se ejecuta si: turno = i

Pi está en espera “activa” si Pj está dentro de su sección crítica. Exclusión mutua ok

Existe una alternativa estricta. El ritmo de ejecución depende del más lento

```
Process Pi:
repeat
  while(turno!=i){};
     SC
  turno:=j;
     SR
forever
```

Si un proceso falla dentro o fuera de su sección crítica el otro nunca podrá entrar a su sección crítica

### Solución no valida

Una variable booleana (0,1) por proceso: flag [0] y flag [1] inicializadas a FALSE

Pi indica que desea entrar en su zona crítica haciendo: flag [i]: = true

¿Exclusión mutua ok? Si uno falla fuera de la sección crítica el otro no está bloqueado bien

```
Process Pi:
repeat
  flag[i]:=true;
  while(flag[j]){};
     SC
  flag[i]:=false;
     SR
forever
```

### Peterson

Inicio: flag [0]: = flag [1]: = false turno: = i o j

El deseo de entrar en la zona crítica se indica con flag [i]: = true y la sección de salida con flag[i]: = false

Si P0 y P1 intentan simultáneamente entrar en su sección crítica se lo impide la variable “turno”

(Ejemplo Página 11 – Secciones Críticas y Exclusión Mutua.pdf)

```
Process Pi:
repeat
  flag[i]:=true;
  turno:=j;
  do {} while
  (flag[j]and turno=j);
     SC
  flag[i]:=false;
     SR
forever
```

### Lamport

Antes de ejecutar la sección crítica cada proceso recoge un número. Aquel que tenga el número más pequeño entra en la sección crítica

Si i < j: Pi entrará primero, si no Pj

Pi pone su número a 0 en la sección de salida

```
Process Pi:
repeat
  choosing[i]:=true;
  number[i]:=max(number[0]..number[n-1])+1;
  choosing[i]:=false;
  for j:=0 to n-1 do {
    while (choosing[j]) {};
    while (number[j]!=0
       and (number[j],j)<(number[i],i)){};
  }
  SC
  number[i]:=0;
  SR
forever
```

### Desventajas de las soluciones software

- Los procesos en espera activa consumen CPU
- Es preferible bloquear los procesos que están esperando a entrar en la sección crítica

*Hardware*

### Desactivación de interrupciones

```
Process Pi:
repeat
  disable interrupts
   critical section
  enable interrupts
   remainder section
forever
```

### Instrucciones particulares

Normal: El acceso a una misma dirección de memoria no se puede hacer mediante varios procesos

Extensiones: Disponibilidad de instrucciones máquina que ejecutan 2 acciones sobre la misma variable de manera atómica (indivisible)

La ejecución de esta instrucción es mutuamente exclusiva incluso si tenemos varios procesadores

Necesitamos algoritmos mas completos para satisfacer las 3 exigencias principales del problema de la sección crítica

### Ventajas

Aplicable a uno o varios procesos y a cualquier número de procesos

Algoritmo simple

Puede utilizarse con múltiples secciones críticas cada una de ellas con una variable distinta

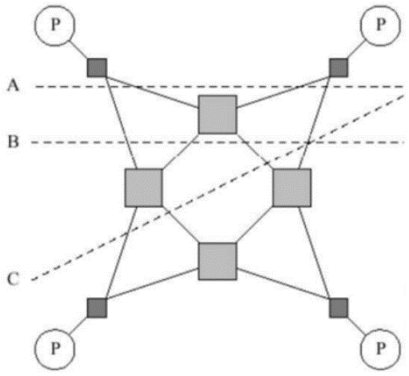
### Desventajas

Existe espera activa

Posible inanición

# Tema 3

## Modulo 1 – Modelos Sistemas/Programación Paralela




El ancho de bisección tiene en cuenta los nodos de procesamiento (p).

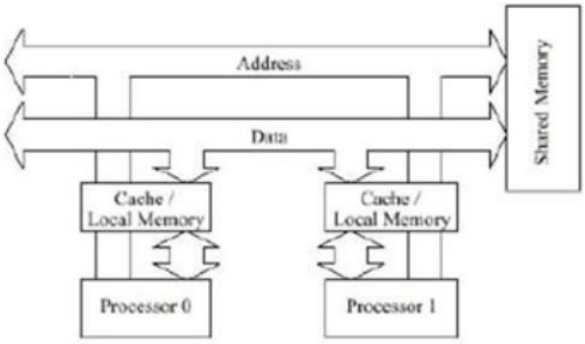
Por ejemplo, en este caso el ancho de bisección es de 4, independientemente de la zona de corte.

### Topologías de red: Bus

Medio compartido:

La información es difundida

- Diámetro: O (1)
- Conectividad: O (1)
- Ancho de bisección: O (1)
- Coste: O (p)
- Bloqueante

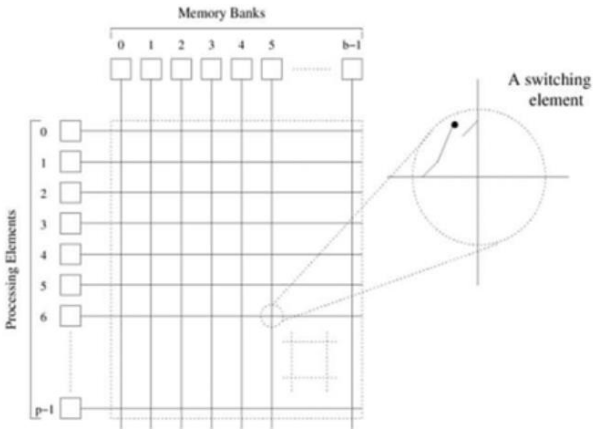



### Topologías de red: Red Matricial

Basada en conmutación

Soporta conexiones simultáneas

- Diámetro: O (1)
- Conectividad: O (1)
- Ancho de bisección: O (p)
- Coste: Ω (p)
- No bloqueante




### Topologías de red: Red multietapa

Caso particular: Red Omega (Ω)

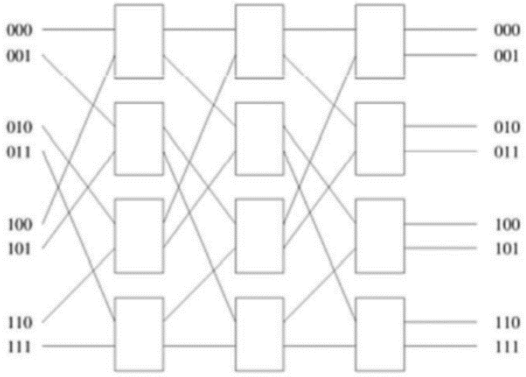
p procesadores ⇒ log p etapas ⇒ p/2 conmutadores por etapa

Caso intermedio entre bus y crossbar

Red omega completa de 8 entradas y 8 salidas

3 etapas y 4 conmutadores por etapa

Coste: O (p·log(p))



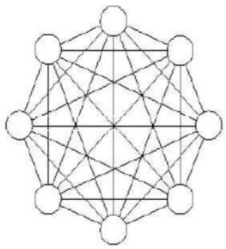
## Topologías de red: Completa y estrella

### *Completa*

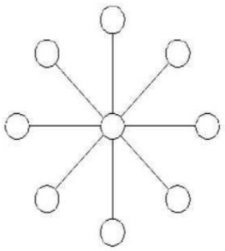
- No bloqueos
- Similar a red matricial

### *Estrella*

- Nodo central cuello de botella
- Similar a bus



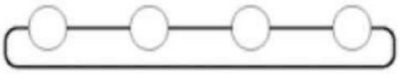
Red completamente conectada (8 nodos)



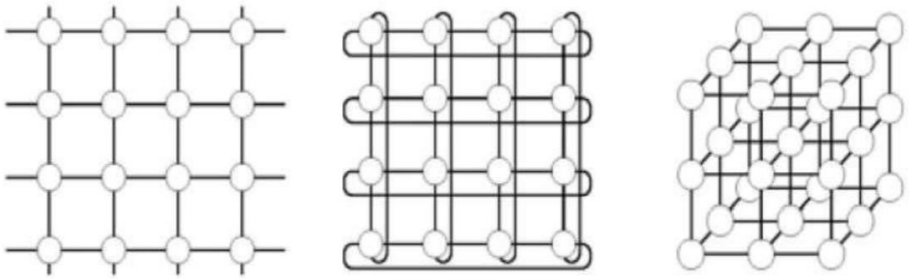
Red conectada en estrella (9 nodos)

## Topologías de red: Estructuras cartesianas

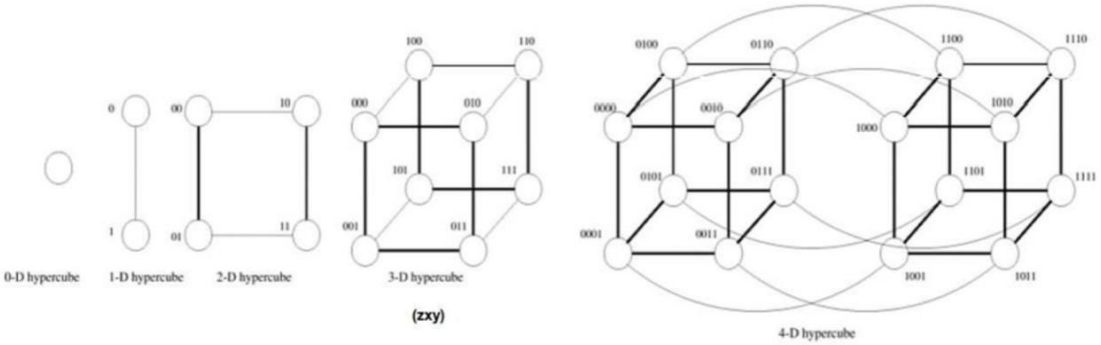
**Arrays lineal**



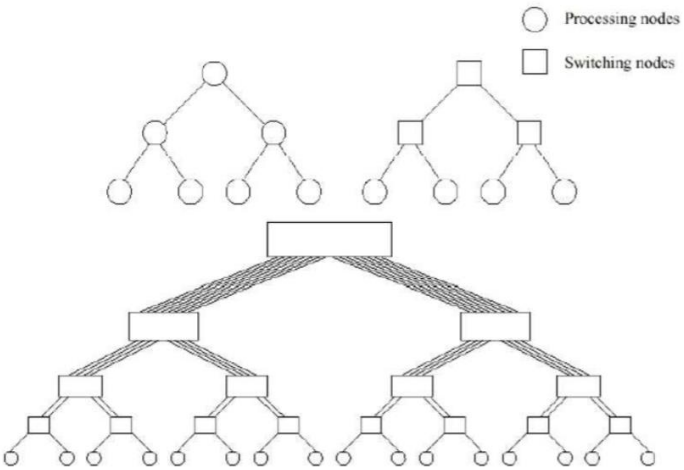
**Anillo**



## Topologías de red: Hipercubos



## Topologías de red: Árboles




## Costes de Comunicación en Sistemas Paralelos

El mayor coste en la ejecución de programas paralelos está asociado a la comunicación de información entre nodos de procesamiento (p)

Influyen:

- Modelo de programación semántica
- Topología de la red
- Tratamiento y enrutamiento de datos

## Costes de Comunicación en Sistemas Paralelos: Paso de Mensajes

El tiempo de comunicación de un mensaje:

- Tiempo de preparación del mensaje para la transmisión
- Tiempo que tarda el mensaje en atravesar la red hasta su destino

**Tiempo de inicio $t_s$**: Añadir cabecera, corrección de errores, ejecución del algoritmo de enrutamiento, conexión entre fuente y destino. Una vez por mensaje.

- m: tamaño, en palabras de red, del mensaje

**Tiempo de salto $t_h$**: Tiempo de desplazamiento entre dos nodos conectados directamente que tarda en enviarse la cabecera.

- l: número de enlaces que debe atravesar el mensaje

**Tiempo de transferencia de palabra $t_w$**: Inverso del ancho del canal de comunicación para atravesar un link

- Si el ancho de banda del canal es "*r*" palabras por segundo, la palabra tarda:

$$t_w = \frac{1}{r}$$

## “Store and forward” y “Cut-through”

Store and forward: manda un mensaje de un nodo a otro sólo después de haber recibido el mensaje completo.

$$t_{com} = t_s + (m \cdot t_w + t_h) \cdot l$$

$$t_{com} = t_s + m \cdot t_w \cdot l$$

Cut-through: manda un mensaje de nodo a otro por partes.

- Se establece el camino entre origen y destino: $l \cdot t_h$
- Se manda el mensaje atravesando el camino establecido: $m \cdot t_w$

$$t_{com} = t_s + m \cdot t_w + t_h \cdot l$$

$$t_{com} = t_s + m \cdot t_w$$

Actualmente, el tiempo de salto ($t_w$) es muy pequeño y puede ser ignorado.

## Modelo para redes NO congestionadas

$$t_{com} = t_s + m \cdot t_w$$

## Modelo para redes congestionadas

$$t_{com} = t_s + m \cdot t_w \cdot \frac{nº\ mensajes}{ancho\ bisección}$$

## Mecanismos de enrutamiento

### *Enrutamiento*

Algoritmo para determinar el camino que un mensaje tomará desde la fuente hasta el destino

### *Clasificación*

#### Determinista vs Adaptativo

Determinista: no tiene en cuenta la información de congestión de la red

Adaptativo: tiene en cuenta la congestión de la red

## Transformaciones en la topología

### *Mapeo entre redes*

Útil en los comienzos de la computación paralela, cuando los algoritmos dependían de las topologías

### *Métricas de calidad de las transformaciones*

- Congestión: máximo número de enlaces de la topología inicial mapeados en un único enlace de la topología final
- Dilatación: máximo número de enlaces consecutivos de la topología final, sobre los que se mapea un único enlace de la topología inicial
- Expansión: relación entre el número de nodos de ambas topologías

# Modulo 3 – Operaciones de comunicación

## Operaciones de Comunicación en MPI

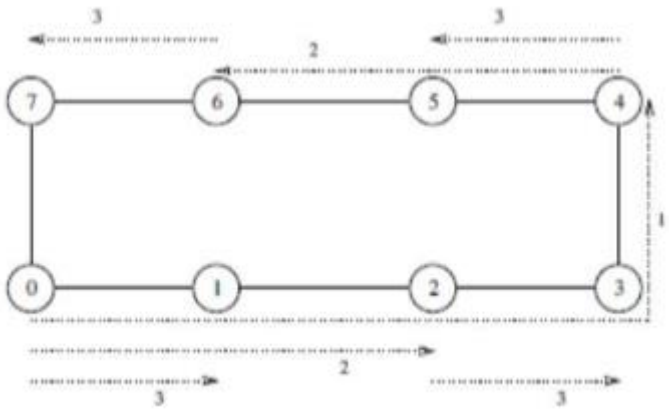
- Difusión Uno-a-todos
- Reducción todos-a-uno
- Difusión todos-a-todos
- Reducción todos-a-todos
- Dispersión
- Agrupamiento
- Todos-a-todos personalizado

## Difusión uno-a-todos y reducción todos-a-uno

- Difusión uno-a-todos. Es frecuente que un proceso tenga que mandar datos idénticos a todos los otros procesos
- Reducción todos-a-uno. Es frecuente que los datos calculados en los p procesos deban juntarse en un solo proceso

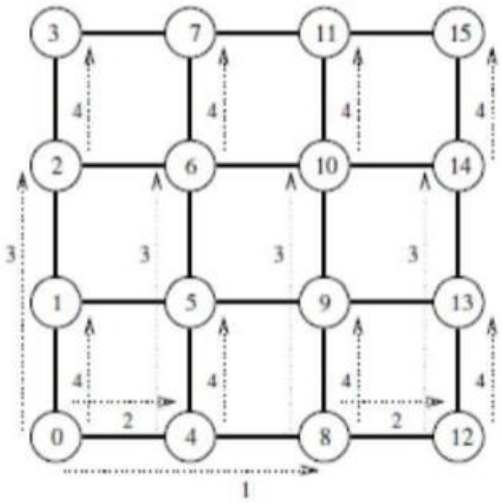
## Comunicación en anillo-array lineal

- Comunicación secuencial:
  - p-1 mensajes
  - Cuello de botella nodo fuente
  - Desaprovechamiento de la red: una única conexión entre un par de nodos al mismo tiempo
- Recursive doubling:
  - Log p mensajes (etapas)
  - Los nodos destino en una etapa, se convierten en fuente en las siguientes
  - Si el nodo 0 mandase mensaje al nodo 1 en el primer paso y después los nodos 0 y 1 intentaran mandar dos mensajes a los nodos 2 y 3 respectivamente durante el segundo paso, el link entre el nodo 1 y 2 resultaría congestionado



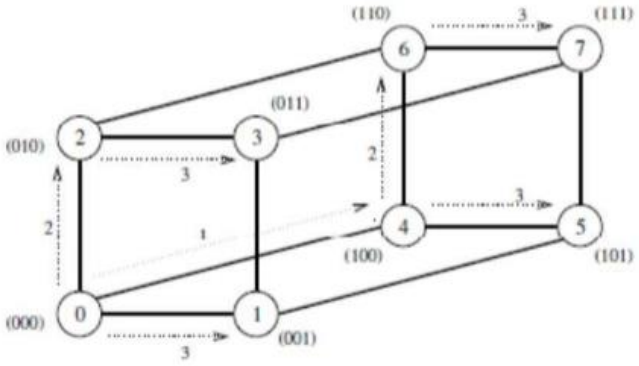
## Difusión en una malla

- Envió de un mensaje desde el nodo 0 al resto de nodos de una malla 2D (p = 16)
- Primera fase:
    - Distribuir el dato a los $p^{1/2} - 1$ nodos de la misma fila a la que pertenece la fuente:
        - Difusión array lineal por la fila
- Segunda fase:
    - Distribuir array lineal por las columnas

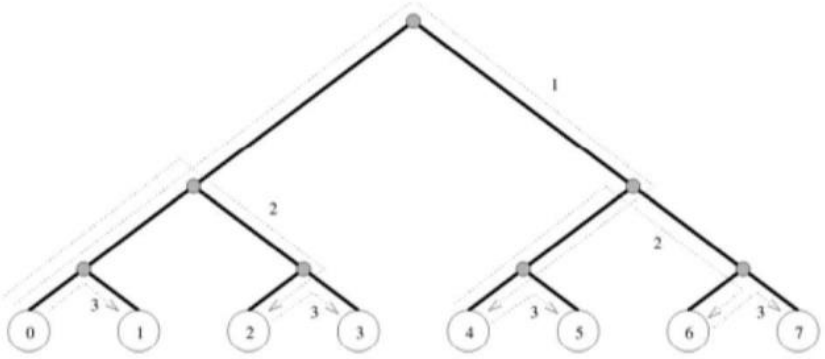



## Difusión en hipercubo

- Hipercubo de $2^d$ nodos (d-hipercubo)
- La difusión se lleva a cabo en d etapas




## Difusión en árbol binario balanceado



## Análisis de coste

p: procesadores — difusión y reducción: $\log p$

m: palabras — transferencia de mensaje punto a punto: $t_s + t_w m$

$$T = (t_s + t_w m) \log p$$

## Difusión y Reducción todos-a-todos

- Difusión todos-a-todos: cada nodo se convierte en fuente de una difusión one-to-all
- Reducción todos-a-todos: cada nodo es destino de una reducción all-to-one

## El resto los puedes ver en: (pág. 12-16. Tema 3 - módulo 3 - Operaciones de comunicación.pdf)

## Análisis de coste

- En anillo o array lineal:

$$T = (t_s + t_w m)(p - 1)$$

- En malla:

$$T = 2 * t_s(\sqrt{p} - 1) + (t_w m) \log p$$

- En hipercubo:

$$T = t_s * \log p + t_w m(p - 1)$$

## Reducción total

- Operación de reducción todas-a-uno más difusión uno-a-todos
- Se puede conseguir mediante difusión todos-a-todos

## Dispersión y Agrupamiento

- Dispersión: varios datos almacenados en un único nodo, se reparten entre distintos procesadores
- Agrupamiento: varios datos almacenados en distintos procesadores, se almacenan simultáneamente en un único nodo

# Modulo 4 – Métricas de rendimiento

## Fuentes de overhead en programas paralelos

- Comunicación entre procesos
- Procesadores ociosos: desequilibrado de carga, sincronización, componentes serie.
- Exceso de computación con respecto al mejor algoritmo serie

## Métricas de rendimiento para programas paralelos

- Tiempo de ejecución paralelo: tiempo empleado para resolver el problema en una plataforma paralela con p procesadores

$$T_p(n,p) = T_{ari} + T_{com}$$

- Función de overhead total:

$$T_o = p * T_p - T_s$$

- Speedup: ganancia de rendimiento de la ejecución paralela, con respecto a la secuencial

$$S = \frac{T_s}{T_p} (Límite\ teórico = p)$$

- Eficiencia: mide la fracción de tiempo que un proceso es utilizado de forma útil (sin overhead)

$$E = \frac{S}{p} = \frac{T_s}{p * T_p}$$

- Coste

$$Coste = p * T_p$$

## Efectos de la granularidad en el rendimiento

- Utilizar el máximo número de procesadores que el algoritmo permite no ser factible
- Solución: sub escalar el sistema (aumentar la granularidad, y utilizar menos procesadores)
    - Si un sistema es óptimo en coste, sigue siéndolo tras sub escalarlo
    - Si un sistema no es óptimo en coste, puede o puede que no lo sea tras sub escalarlo

## Escalabilidad de los sistemas paralelos

Frecuentemente, los programas paralelos se testean en problemas pequeños con pocos elementos de procesamiento, para simplificar

## Función de isoeficiencia

La función de isoeficiencia indica cuánto tiene que aumentar el tamaño del problema para poder incluir más procesadores sin que la Eficiencia del sistema se resienta

$$W = k * T_p(W,p)$$

# Tema 4

## Clasificación de los Sistemas de Tiempo Real

Sistema en tiempo real son aquellos que deben producir respuestas correctas dentro de un intervalo de tiempo definido. Si el tiempo de respuesta excede ese límite funcionamiento erróneo.

Clasificación:

- Real estricto (hard real time): absolutamente necesario
  - Ej.: Control de vuelo
- Real no estricto (soft real time): se permite la pérdida ocasional de especificaciones temporales
  - Ej.: Sistema de adquisición de datos
- Real firme (firm real time): se permite la pérdida ocasional de especificaciones temporales ya que la respuesta retrasada es descartada
  - Ej.: Sistema multimedia

## Clasificación de Tareas en Sistemas de Tiempo Real

- Semántica:
  - Críticas (puede ser catastrófico)
  - No críticas/Opcionales
- Forma de ejecución:
  - Tareas periódicas
  - Tareas esporádicas
  - Tareas aperiódicas

## Sistemas Operativos de Tiempo Real

Definición:

- Aquel que ha sido desarrollado para aplicaciones de tiempo real y que debe ser predecible para garantizar el comportamiento correcto en el tiempo requerido

Objetivos que persigue la planificación

- Garantizar la correcta ejecución de los procesos críticos
- Ofrecer un buen tiempo de ejecución de todos los procesos no periódicos
- Administrar recursos compartidos
- Posibilidad de recuperación ante fallos software y hardware
- Soporta cambios de modo (cambiar en ejecución el conjunto de tareas)

## Consideraciones extras para el Diseño

- Qué consideramos inicio de respuesta/retorno útil
- Precisar el plazo máximo para ejecución
- Precisar la frecuencia de iteración (si son periódicas)
- Considerar sincronización de tareas
- Nivel de criticidad

Si las plataformas subyacentes no pueden garantizar el cumplimiento de estos criterios, puede resultar necesario la realización de un planificador (Scheduler) y lanzador (Dispacher)

Scheduler:

- Descompone una tarea paralela en un conjunto de hebras secuenciales
- Asignación de prioridad y partición algoritmo que establece prioridades para cada secuencia hebra

Dispacher:

- Responsable de hacer cumplir los horarios generados previamente y proporcionar sincronización al final de cada segmento durante el tiempo de ejecución
- Esto requiere que el vigilar las prioridades de programación, la preferencia en el tiempo de ejecución y la sincronización utilizando los servicios de capas inferiores

## Planificación de tareas

Un método de planificación puede ser:

- Estático: análisis antes de la ejecución (planificación con prioridades fijas y desalojo)
- Dinámico: análisis durante la ejecución

### Propiedades

- No hay concurrencia
- Compartir datos
- Períodos armónicos
- No hace falta analizar

### Prioridades monótonas en frecuencia

La asignación de mayor prioridad a las tareas de menor período es óptima para modelo de tareas simple

### Condición de garantía de los plazos basada en la utilización

Para el modelo simple, con prioridades monótonas en frecuencia, los plazos están garantizados si

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N * (\sqrt[N]{2} - 1)$$

La cantidad es la utilización mínima garantizada para N tareas

$$U_0(N) = N * (\sqrt[N]{2} - 1)$$

### Análisis del tiempo de respuesta

- La prueba de factor de utilización no es exacta, ni se puede generalizar a modelos de tareas más complejos
- La construcción de un cronograma es compleja, incluso considerando que el instante inicial es crítico
- Veremos una prueba basada en el cálculo del tiempo de respuesta de cada tarea

### Cálculo de la interferencia máxima

$$I_i = \sum_{j \in hp(i)} \frac{R_i}{T_j} * C_j$$

## Iteración lineal

La ecuación del tiempo de respuesta se puede resolver mediante la relación de recurrencia

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \frac{w_i^n}{|T_j|} * C_j$$

Un valor inicial aceptable es

$$w_i^0 = C_i + \sum_{j \in hp(i)} C_j$$

Se termina cuando

- $w^{n+1} = w^n$
- $w^{n+1} > T_i (no\ se\ cumple\ el\ plazo)$

## Tiempo de cómputo

Hay dos formas de obtener el valor de C para una tarea:

- Medida del tiempo de ejecución
- Análisis del código ejecutable

## Tareas esporádicas

- Para incluir tareas esporádicas hace falta modificar el modelo simple de tareas
- El análisis de tiempo de respuesta sigue siendo válido
- Funciona bien con cualquier orden de prioridad

## Interacción entre tareas

- En la mayoría de los sistemas de interés práctico las tareas interaccionan mediante
    - Datos comunes
    - Citas o mensajes
- Bloque, inversión de prioridad (es posible limitar su duración, pero no se puede eliminar)

## Protocolos de techo de prioridad

El techo de prioridad de un recurso es la máxima prioridad de las tareas que lo usan

El protocolo consiste en:

- La prioridad dinámica de una tarea es el máximo de su prioridad básica y las prioridades de las tareas a las que bloquea
- Una tarea sólo puede usar un recurso si su prioridad dinámica es mayor que el techo de todos los recursos en uso por otras tareas

### *Propiedades*

(Pág. 25. Planificación.pdf)

## Modelo de tareas generalizado

El modelo de tareas básico que hemos visto incluye:

- Tareas periódicas y esporádicas
- Plazos menores o iguales que los periodos
- Interacción mediante secciones criticas

El análisis del tiempo de respuesta se  puede extender con

- Planificación cooperativa
- Variación (jitter) en el esquema de activación
- Plazos arbitrarios

Ventajas

- Mejora la planificación
- Se pueden reducir los tiempos de computo
- Los tiempos de computo de los bloques son más fáciles de calcular

Problemas

- El coste de ofrecer un cambio de contexto puede ser alto