Tema 8: Multiprocesadores de memoria compartida y distribuida

Indice

1. Introducción al procesamiento paralelo

Fuentes de paralelismo en los programas

• En relación con la explotación del paralelismo para su traducción en mejora de rendimiento las aplicaciones (programas) manifiestan tres tipos:

- Paralelismo de control
- Paralelismo de datos
- Paralelismo de flujo

• El **paralelismo de control** proviene del hecho de que determinadas sub-tareas de la aplicación son independientes y en consecuencia pueden realizarse en paralelo (simultáneamente).

- Los elementos de proceso se asocian a las sub-tareas independientes

• El **paralelismo de datos** proviene del hecho de que ciertas aplicaciones trabajan con estructuras de datos muy regulares (vectores y matrices), repitiendo una misma acción sobre cada elemento de la estructura.

- Los elementos de proceso se asocian a los datos elementales de las estructuras.

• El **paralelismo de flujo** proviene del hecho de que ciertas aplicaciones funcionan como un flujo de datos sobre el que debe efectuarse una sucesión de operaciones en cascada (etapas).

- Los elementos de proceso se asocian a las múltiples etapas de la aplicación.

1. Introducción al procesamiento paralelo

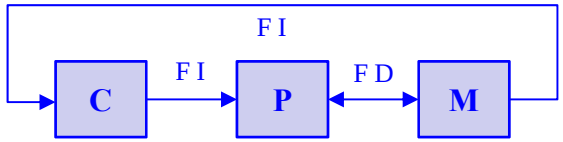
Clasificación de *Flynn* de los computadores paralelos

• Flynn clasificó los computadores paralelos atendiendo al carácter Simple (S) o Múltiple (M) del Flujo de Instrucciones y el Flujo de Datos

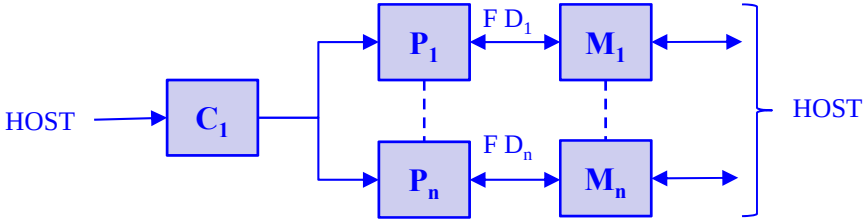
CLASIFICACION DE FLYNN		Flujo Instrucciones	
		S	M
Flujo Datos	S	SISD	MISD
	M	SIMD	MIMD

C = Control
P = Proceso (Unidad)
M = Memoria
MC = Memoria Compartida
FI = Flujo de Instrucciones
FD = Flujo de Datos

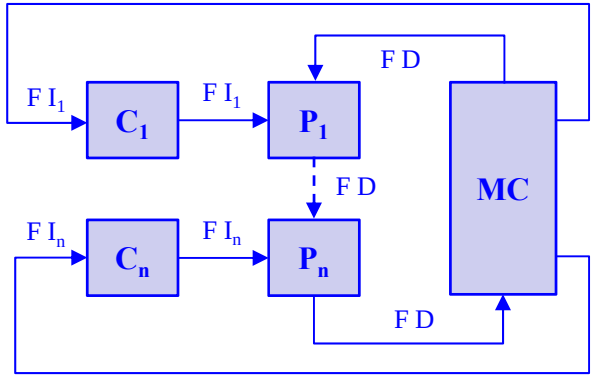
SISD (Single Instruction Single Data)



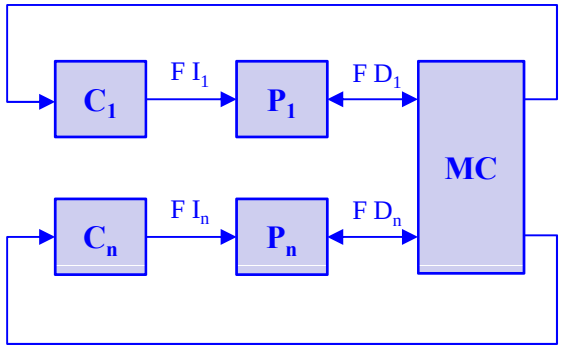
SIMD (Single Instruction Multiple Data)



MISD (Multiple Instruction Single Data)



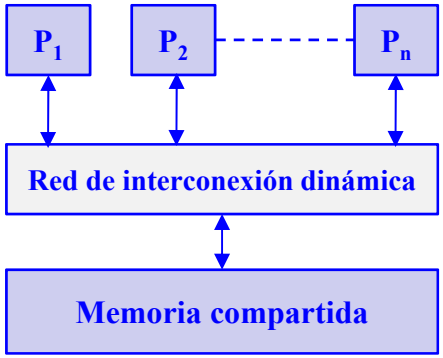
MIMD (Multiple Instruction Multiple Data)



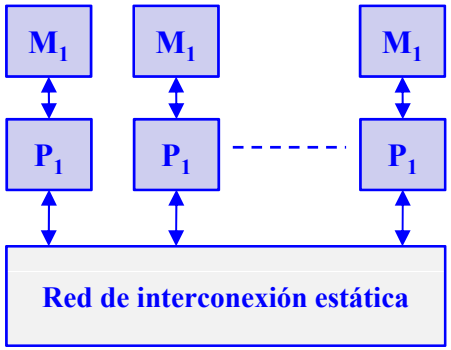
1. Introducción al procesamiento paralelo

Clasificación de los multiprocesadores por la ubicación de la memoria

- Multiprocesadores de memoria compartida
  - Todos los procesadores acceden a una memoria común
  - La comunicación entre procesadores se hace a través de la memoria
  - Se necesitan primitivas de sincronismo para asegurar el intercambio de datos



- Multiprocesadores de memoria distribuida o multicomputadores
  - Cada procesador tiene su propia memoria
  - La comunicación se realiza por intercambio explicito de mensajes a través de una red

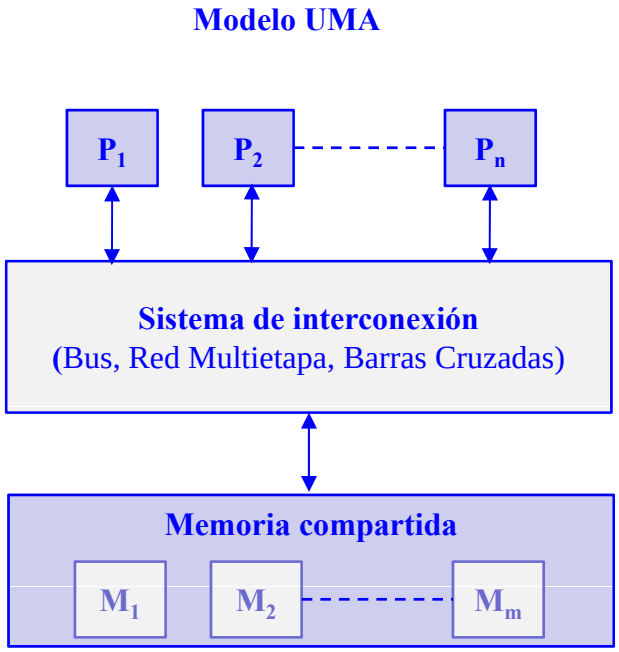


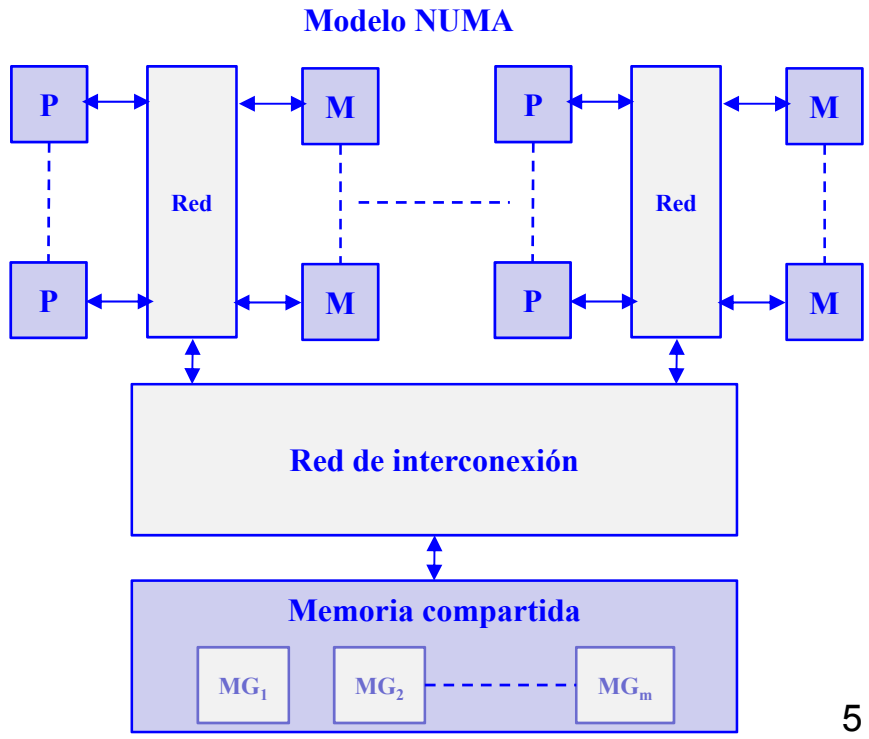
2. Estructura de los multiprocesadores de memoria compartida

• La mayoría de los multiprocesadores comerciales son del tipo UMA (*Uniform Memory Access*): todos los procesadores tienen igual tiempo de acceso a la memoria compartida.

• En la arquitectura UMA los procesadores se conectan a la memoria a través de un bus, una red multietapa o un conmutador de barras cruzadas (*crossbar*) y disponen de su propia memoria caché.

• Los procesadores tipo NUMA (*Non Uniform Memory Access*) presentan tiempos de acceso a la memoria compartida que dependen de la ubicación del elemento de proceso y la memoria.

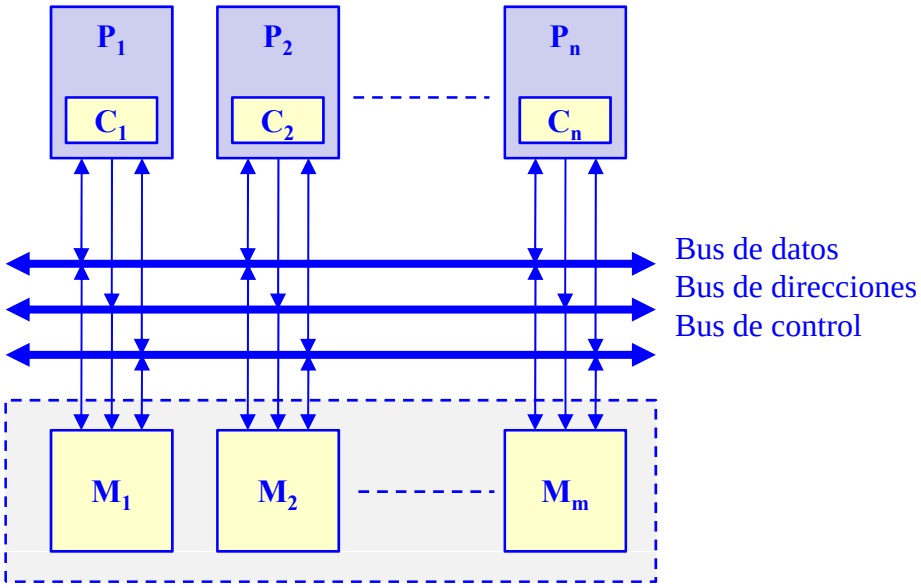




3. Medio de interconexión de los procesadores con la memoria

Multiprocesadores de memoria compartida: conexión por bus compartido

- Es la organización más común en los computadores personales y servidores
- El bus consta de líneas de dirección, datos y control para implementar:
  - El protocolo de transferencias de datos con la memoria
  - El arbitraje del acceso al bus cuando más de un procesador compite por utilizarlo.
- Los procesadores utilizan cachés locales para:
  - Reducir el tiempo medio de acceso a memoria, como en un monoprocesador
  - Disminuir la utilización del bus compartido.



3. Medio de interconexión de los procesadores con la memoria

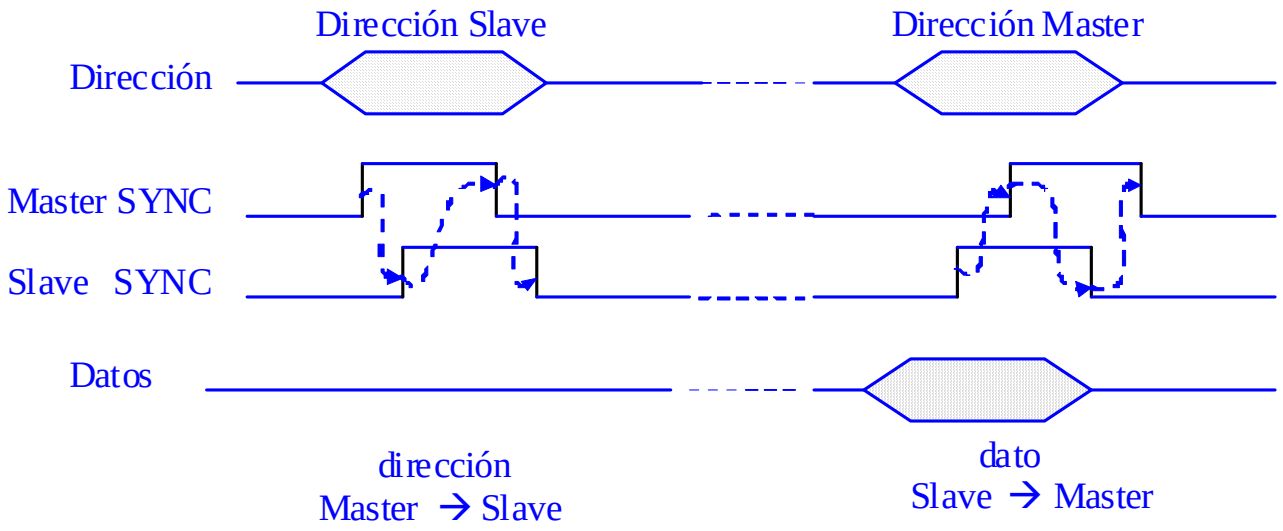
Protocolos de transferencia de ciclo partido

- La operación de lectura se divide en dos transacciones no continuas de acceso al bus.

- La primera es de petición de lectura que realiza el *máster (procesador)* sobre el *slave (memoria)*.

- Una vez realizada la petición el *máster* abandona el bus.

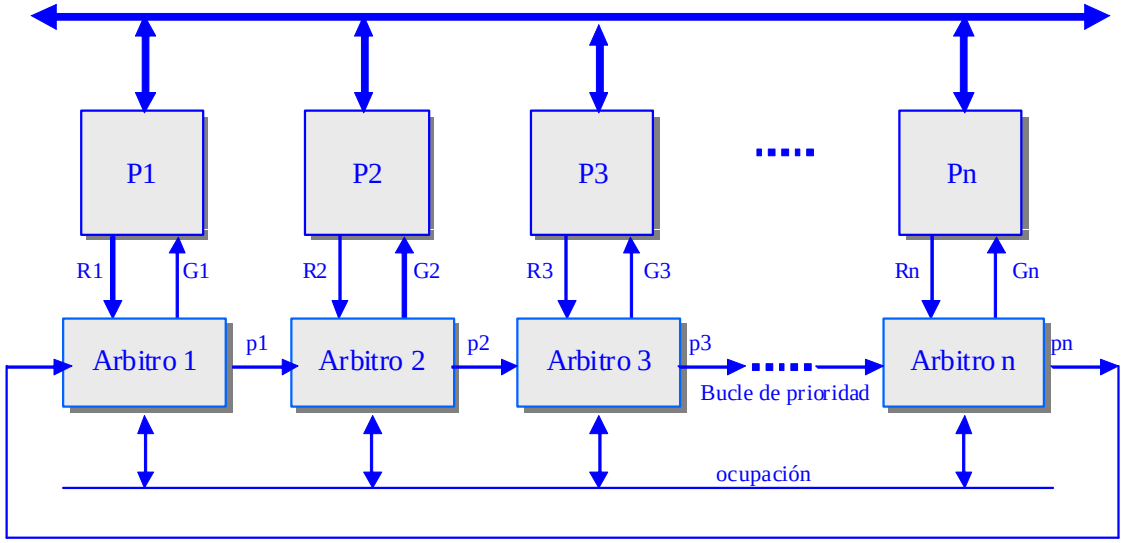
- Cuando el *slave* dispone del dato leído, inicia un ciclo de bus actuando como *máster* para enviar el dato al antiguo *máster*, que ahora actúa como *slave*.



3. Medio de interconexión de los procesadores con la memoria

Protocolo de arbitraje distribuido

- La responsabilidad del arbitraje se distribuye por los diferentes *procesadores* conectados al bus.

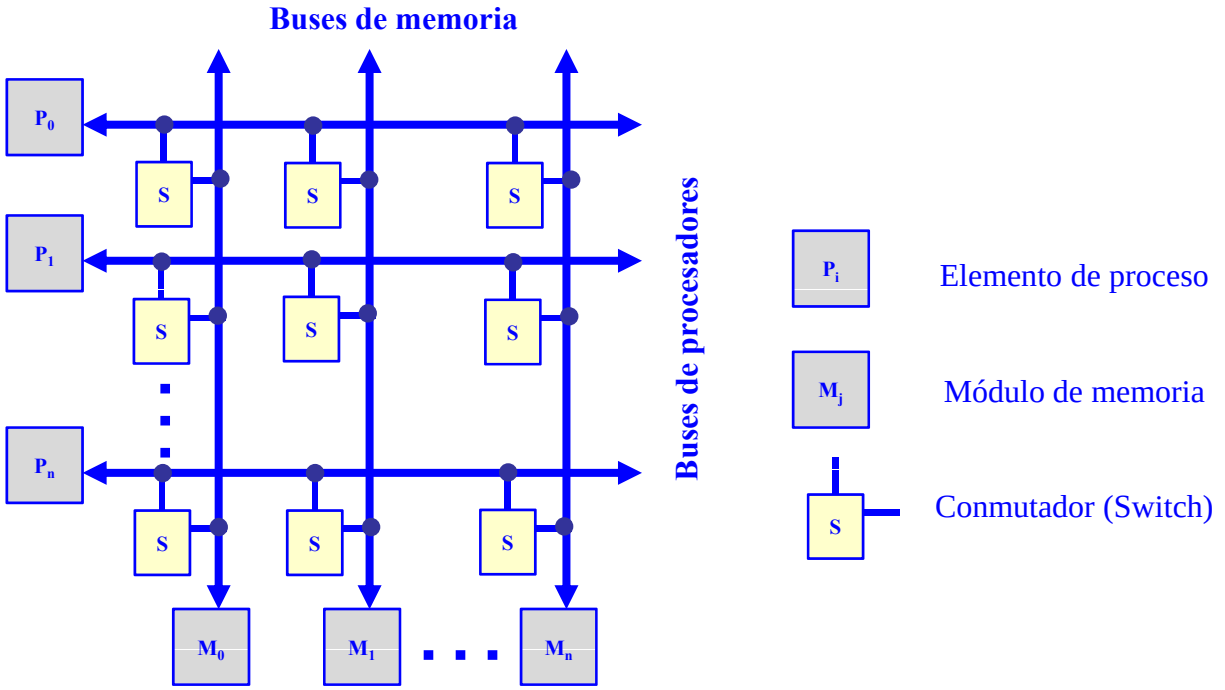


- Arbitro-i concede el bus al *procesador Pi* activando *Gi* si:
  - *Pi* ha activado su línea de petición de bus *Ri*.
  - La línea de ocupación está desactivada.
  - La línea de entrada de prioridad *pi-1* está activada
- El árbitro *i* activa su línea de prioridad *pi* si:
  - *Pi* no ha activado su línea de petición *Ri*
  - La línea de prioridad *pi-1* está activa
  - Finaliza una operación de acceso al bus

3. Medio de interconexión de los procesadores con la memoria

Multiprocesadores de memoria compartida: conexión por conmutadores *crossbar*

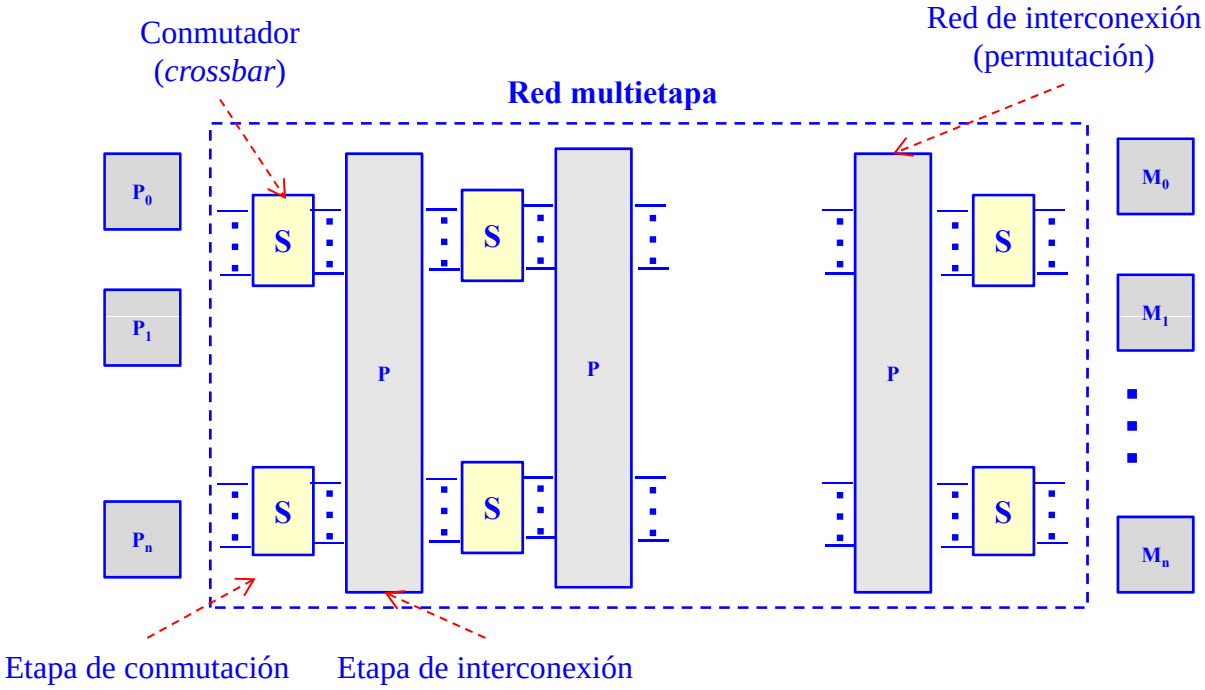
- Cada procesador (*Pi*) y cada módulo de memoria (*Mi*) tienen su propio bus
- Existe un conmutador (*S*) en los puntos de intersección que permite conectar un bus de memoria con un bus de procesador
- Para evitar conflictos cuando más de un procesador pretende acceder al mismo módulo de memoria se establece un orden de prioridad
- Se trata de una red sin bloqueo con una conectividad completa pero de alta complejidad



3. Medio de interconexión de los procesadores con la memoria

Multiprocesadores de memoria compartida: conexión por red multietapa

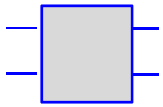
- Representan una alternativa intermedia de conexión entre el bus y el *crossbar.*
- Es de menor complejidad que el *crossbar* pero mayor que el bus simple.
- La conectividad es mayor que la del bus simple pero menor que la del *crossbar.*
- Se compone de varias etapas alternativas de conmutadores simples y redes de interconexión.
- En general las redes multietapa responden al siguiente esquema:



3. Medio de interconexión de los procesadores con la memoria

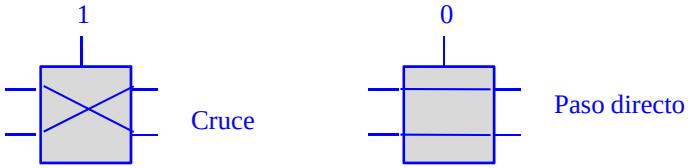
Ejemplo de rede multietapa: red Omega (ω)

- Se trata de una red multietapa compuesta de conmutadores básicos de 2 entradas y 2 salidas

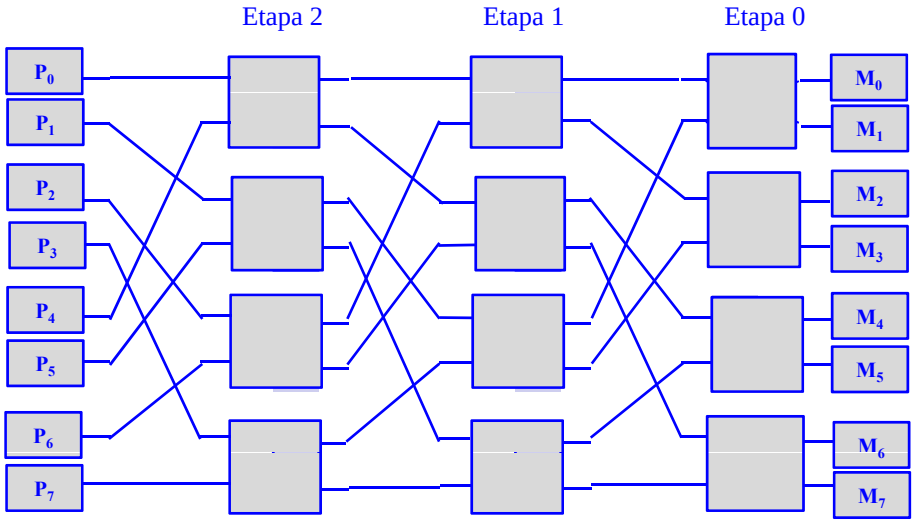


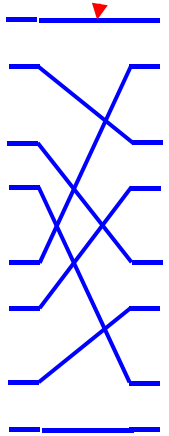
Conmutador básico

- Cada conmutador puede estar en dos estados: *paso directo* y *cruce*



- La interconexión entre etapas se realiza con un patrón fijo denominado *barajadura perfecta*.
- La red de 3 etapas que conecta 8 procesadores con 8 módulos de memoria sería la siguiente:





3. Medio de interconexión de los procesadores con la memoria

Encaminamiento en la red Omega

- Algoritmo 1:

$(d_2, d_1, d_0) = \text{índice en binario del módulo de memoria destino}$

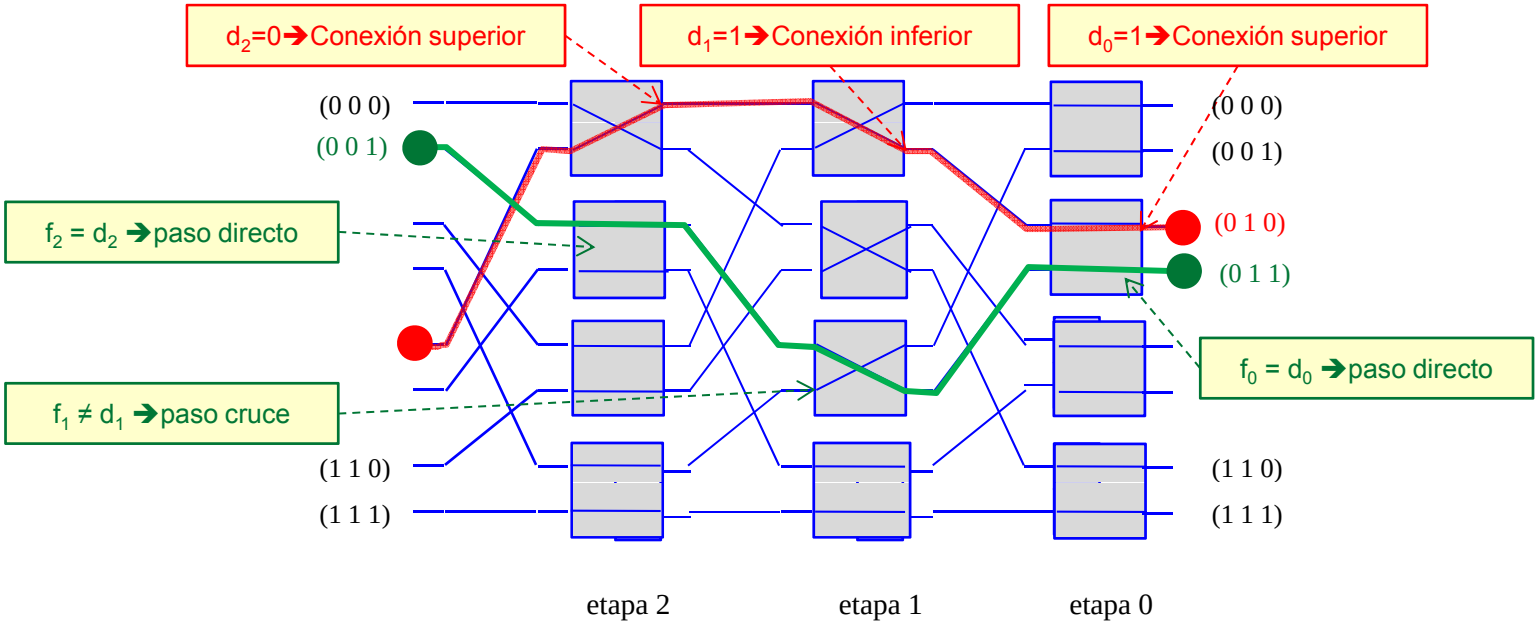
$$\text{Etapa } i \text{ si} \begin{cases} d_i = 0 \Rightarrow \text{Conexión de entrada a salida superior} \\ d_i = 1 \Rightarrow \text{Conexión de entrada a salidainferior} \end{cases}$$

- Algoritmo 2:

$(f_2, f_1, f_0) = \text{índice en binario del procesador fuente}$

$(d_2, d_1, d_0) = \text{índice en binario del módulo de memoria destino}$

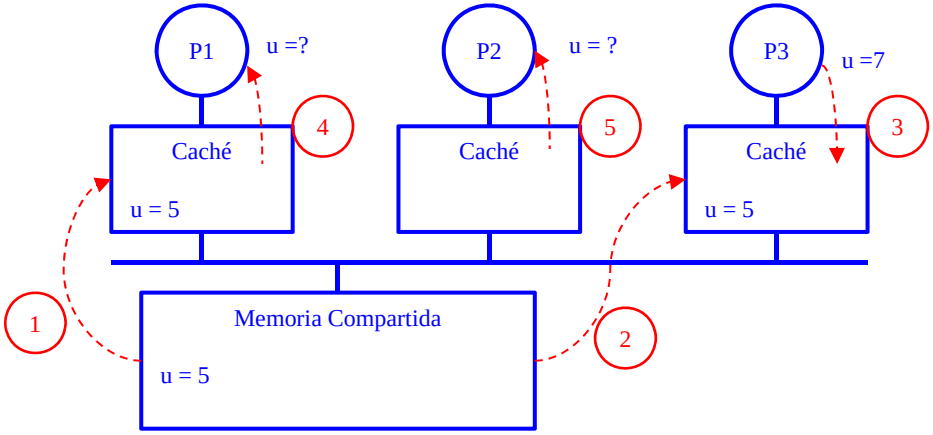
$$\text{Etapa } i \text{ si} \begin{cases} d_i = f_i \Rightarrow \text{Conexión de paso directo} \\ d_i \neq f_i \Rightarrow \text{Conexión de cruce} \end{cases}$$



4. Consistencia de memoria cache

Problema de coherencia caché

• El problema de la coherencia caché en multiprocesadores surge por las operaciones de escritura.
• **Ejemplo**: secuencia de operaciones de acceso a memoria (1, 2, 3, 4, 5) realizadas por tres procesadores P1, P2 y P3 sobre la posición de memoria *u*:



• Caché con política *write-through*
  ▪ La escritura realizada por el procesador *P3* hará que el valor de *u* en la memoria principal sea 7. Sin embargo, el procesador *P1* leerá el valor de *u* de su caché en vez de leer el valor correcto de la memoria principal.

• Caché con política *writeback*
  ▪ En este caso el procesador *P3* únicamente activará el bit de modificado en el bloque de la caché en donde tiene almacenado *u* y no actualizará la memoria principal.
  ▪ Sólo cuando ese bloque de la caché sea reemplazado se actualizará la memoria principal.
  ▪ No será sólo *P1* el que leerá el valor antiguo, sino que cuando *P2* intente leer *u* y se produzca un fallo en la caché, también leerá el valor antiguo de la memoria principal

4. Consistencia de memoria cache

Solución de la coherencia caché

- Existen dos formas de abordar el problema de la coherencia caché.
  - Software, lo que implica la realización de compiladores que eviten la incoherencia entre cachés de datos compartidos.
  - Hardware que mantengan de forma continua la coherencia en el sistema, siendo además transparente al programador.
- Podemos distinguir también dos tipos de sistemas multiprocesadores
  - Sistemas basados en un único bus: se utilizan protocolos de sondeo o *snoopy* que analizan el bus para detectar incoherencia. Cada nodo procesador tendrá los bits necesarios para indicar el estado de cada línea de su caché y así realizar las transacciones de coherencia necesarias según lo que ocurra en el bus en cada momento.
  - Sistemas con redes multietapa: protocolo basado en directorio, que consiste en la existencia de un directorio común donde se guardan el estado de validez de las líneas de las cachés, de manera que cualquier nodo puede acceder a este directorio común.
- Tanto si son de sondeo como si no, existen dos políticas para mantener la coherencia:
  - *Invalidación en escritura (write invalidate):* siempre que un procesador modifica un dato de un bloque en la caché, invalida todas las demás copias de ese bloque guardadas en las otras cachés.
  - *Actualización en escritura (write update):* actualiza las copias existentes en las otras cachés en vez de invalidarlas.

4. Consistencia de memoria cache

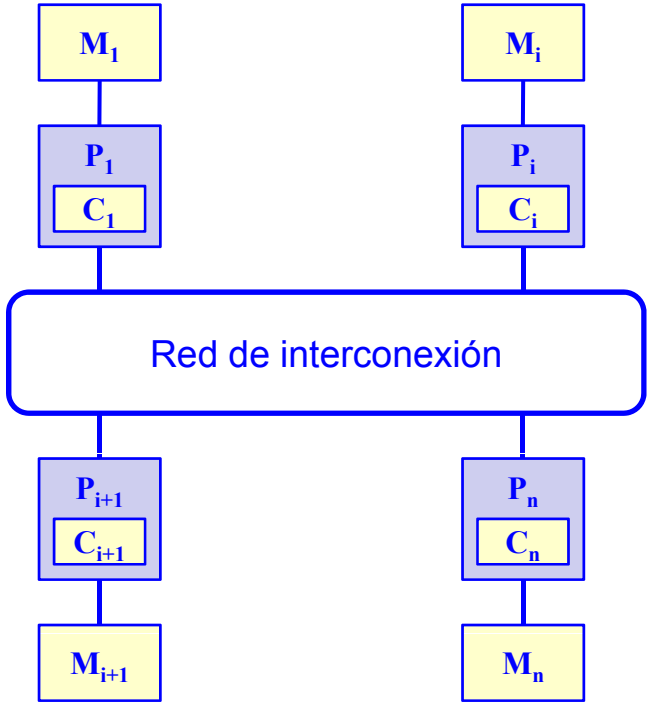
Protocolos de sondeo (snoopy)

- Protocolo MSI: protocolo de invalidación para cachés post-escritura.
- Utiliza los siguientes estados: *inválido* (I), *compartido* (S) y *modificado* (M).
  - Inválido: tiene un significado inmediato.
  - Compartido: el bloque está presente en la caché y no ha sido modificado, la memoria principal está actualizada y cero o más cachés adicionales pueden tener también una copia actualizada (compartida).
  - Modificado: únicamente este procesador tiene una copia válida del bloque en su caché, la copia en la memoria principal está anticuada y ninguna otra caché puede tener una copia válida del bloque .
- Antes de que un bloque compartido pueda ser escrito y pasar al estado modificado, todas las demás copias potenciales deben de ser invalidadas vía una transacción de bus de lectura exclusiva.
- Esta transacción sirve para ordenar la escritura al mismo tiempo que causa la invalidaciones y por tanto asegura que la escritura se hace visible a los demás.
- El procesador emite dos tipos de peticiones: lecturas (*PrRd*) y escrituras (*PrWr*).
- Las lecturas o escrituras pueden ser a un bloque de memoria que existe en la caché o a uno que no existe. En el último caso, el bloque que esté en la caché en la actualidad será reemplazado por el nuevo bloque, y si el bloque actual está en el estado modificado su contenido será volcado a la memoria principal.

5. Estructura de los multiprocesadores de memoria distribuida

Multicomputadores

- Los multiprocesadores de memoria compartida presentan algunas desventajas:
  - Se necesitan técnicas de sincronización para acceder a las variables compartidas
  - La contención en la memoria puede reducir significativamente la velocidad.
  - No son fácilmente escalables a un gran número de procesadores.
- Un **multicomputador** consta de un conjunto de procesadores conectados por una red
- Cada procesador tiene su propia memoria local, incluida la caché, y se comunican por paso de mensajes a través de la red



5. Estructura de los multiprocesadores de memoria distribuida

Propiedades de los multicomputadores

- El número de nodos puede ir desde algunas decenas hasta varios miles (o más).
- La arquitectura de paso de mensajes tiene ventajas sobre la de memoria compartida cuando el número de procesadores es grande.
- El número de canales físicos entre nodos suele oscilar entre cuatro y ocho.
- Esta arquitectura es directamente escalable y presenta un bajo coste para sistemas grandes.
- Un problema se especifica como un conjunto de procesos que se comunican entre sí y que se hacen corresponder sobre la estructura física de procesadores.
- El tamaño de un proceso viene determinado por su *granularidad*:

$$Granularidad = \frac{Tiempo\ de\ cálculo}{Tiempo\ de\ comunicación}$$

- Al reducirse la granularidad, la sobrecarga de comunicación de los procesos aumenta.
- Por ello, la granularidad empleada en este tipo de máquinas suele ser media o gruesa.
- El programa a ejecutar debe de ser intensivo en cálculo, no intensivo en operaciones de entrada/salida o de paso de mensajes

6. Topología de las redes estáticas de interconexión

Redes estáticas de interconexión en multicomputadores

- Los multicomputadores utilizan redes estáticas con enlaces directos entre nodos
- Cuando un nodo recibe un mensaje lo procesa si viene dirigido a dicho nodo
- Si el mensaje no va dirigido al nodo receptor lo reenvía a otro por alguno de sus enlaces de salida siguiendo un protocolo de encaminamiento.
- Las propiedades más significativas de una red estática son las siguientes:

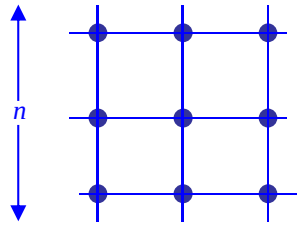
  - **Topología de la red**: determina el patrón de interconexión entre nodos.
  - **Diámetro de la red**: distancia máxima de los caminos más cortos entre dos nodos de la red.
  - **Latencia**: retardo de tiempo en el peor caso para un mensaje transferido a través de la red.
  - **Ancho de banda**: Transferencia máxima de datos en Mbytes/segundo.
  - **Escalabilidad**: posibilidad de expansión modular de la red.
  - **Grado de un nodo**: número de enlaces o canales que inciden en el nodo.
  - **Algoritmo de encaminamiento**: determina el camino que de be seguir un mensaje desde el nodo emisor al nodo receptor.

6. Topología de las redes estáticas de interconexión

Algunas topología de las redes estáticas

- Mallas

simple



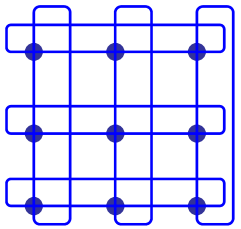
$diámetro = 2 \cdot (n-1)$

$grado = 4$

$nº\ nodos = n^2$

$nº\ enlaces = 2 \cdot n \cdot (n-1)$

toroidal



$diámetro = 2 \cdot \lfloor n/2 \rfloor$

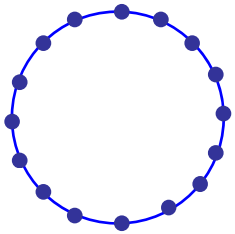
$grado = 4$

$nº\ nodos = n^2$

$nº\ enlaces = 2n^2$

- Anillos

simple



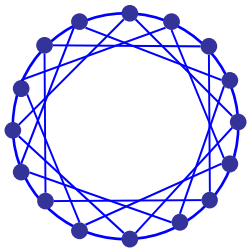
$diámetro = n-1$

$grado = 2$

$nº\ nodos = n$

$nº\ enlaces = n-1$

cordado



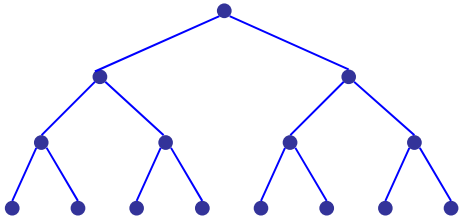
$diámetro = 3$

$grado = 4$

$nº\ nodos = n$

$nº\ enlaces = 32$

- Árbol



$diámetro = 2(k-1)$

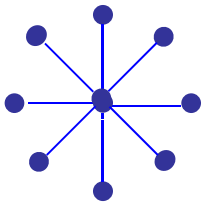
$grado = 4$

$nº\ nodos = 2^k - 1$

$nº\ enlaces = 2^k - 2$

$k = nº\ niveles$

- Estrella



$diámetro = 2$

$grado = n-1$

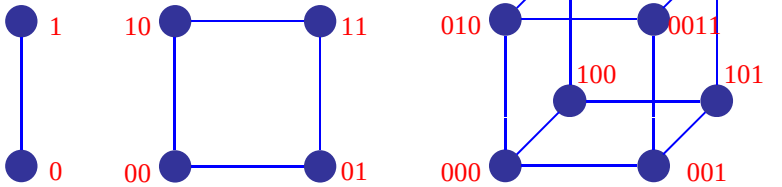
$nº\ nodos = n$

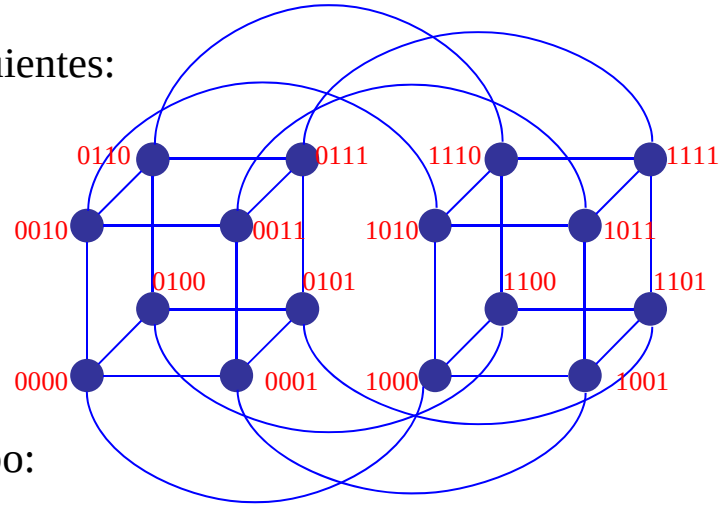
$nº\ enlaces = n-1$

6. Topología de las redes estáticas de interconexión

Hipercubos

- Un *cubo-n* o hipercubo de dimensión *n* consta de $N=2^n$ nodos extendidos a lo largo de *n* dimensiones.
- El grado vale *n*
- El diámetro también vale *n*
- Existen *n* caminos disjuntos entre cualquier par de nodos
- Cada nodo se etiqueta con un número binario de n bits de tal modo asignados que dos vértices conectados se diferencian en un solo bit.
- Los hipercubos de grado 1, 2, 3 y 4 serían los siguientes:





- Ejemplo de computador con topología de hipercubo:

SGI ICE de 11.776 nodos

6. Topología de las redes estáticas de interconexión

Algoritmo de encaminamiento en hipercubos

Número de nodos : $N = 2^n$; *dirección de un nodo genérico* : $v = v_{n-1}...v_1v_0$

dirección del nodo fuente : $s = s_{n-1}...s_1s_0$; *dirección del nodo destino* : $d = d_{n-1}...d_1d_0$;

Objetivo : *Determinar una ruta de s a d con menor número de pasos*

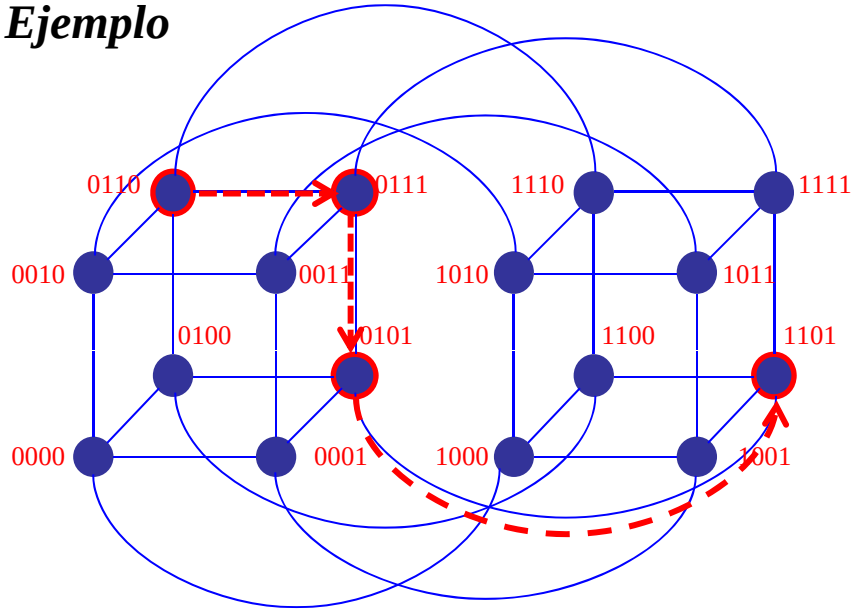
Paso 1 : *Se calcula el vector dirección* $r = r_{n-1}...r_1r_0$ *para las n dimensiones* : $r_i = s_i \oplus d_i$

Paso 2 : *Inicialización* $i = 0; v = s$

Paso 3 : *Si* $r_i = \begin{cases} 1 \Rightarrow \text{se pasa del nodo } v \text{ al nodo } v \oplus 2^i \\ 0 \text{ saltar este paso} \end{cases}$

Paso 4 : $i = i + 1$; *Si* $i \le n - 1 \Rightarrow$ *ir al Paso 2; caso contrario FIN*

Ejemplo



$n = 4$

fuente : $s = 0110$; destino: $d = 1101$

vector dirección : $r = s \oplus d = 1011 = r_3r_2r_1r_0$; $v = 0110$

$r_0 = 1 \Rightarrow s \rightarrow v_1 = v \oplus 2^0 = 0110 \oplus 0001 = 0111$

$r_1 = 1 \Rightarrow v_1 \rightarrow v_2 = v_1 \oplus 2^1 = 0111 \oplus 0010 = 0101$

$r_2 = 0 \Rightarrow nada$

$r_3 = 1 \Rightarrow v_2 \rightarrow d = v_2 \oplus 2^3 = 0101 \oplus 1000 = 1101$ $v = 0110$

7. Encaminamiento de mensajes

Técnicas de conmutación en redes de multicomputadores

- Determinan el método utilizado para transmitir los mensajes entre nodos.
- Existen dos métodos básicos: conmutación de circuitos y conmutación de paquetes.
- En la **conmutación de circuitos** se establece un camino eléctrico entre los nodos fuente y destino para transmitir el mensaje.
- En la **conmutación de paquetes** el nodo fuente descompone el mensaje en paquetes que son enviados por la red hacia el nodo destino.
- Cada paquete almacena el número de secuencia que le corresponde en el mensaje para que el nodo destino pueda reconstruirlo.
- Los paquetes de un mismo mensaje se envían de forma independiente, incluso por caminos diferentes.
- Se utilizan dos alternativas en la conmutación de paquetes:
  - Almacenamiento y envío (*store-and-forward)*
    - Utilizada en los primeros multicomputadores
    - La latencia depende de la distancia entre nodos
  - Segmentación asíncrona (*wormhole)*
    - Utilizada en los multicomputadores actuales
    - La latencia es independiente de la distancia entre nodos.

7. Encaminamiento de mensajes

Conmutación de circuitos

• En la conmutación de circuitos, se reserva un camino físico desde el origen hasta el destino antes de producirse la transmisión de los datos.

• Esto se consigue mediante la emisión de un *flit* cabecera que contiene la dirección del destino, y conforme progresa hacia el destino reserva los enlaces físicos de los nodos intermedios.

• Cuando la prueba alcanza el destino, se habrá establecido un camino enviándose una señal de asentimiento de vuelta al origen.

• El contenido del mensaje puede ahora ser emitido a través del camino hardware.

• El circuito puede liberarse por el destino o por los últimos bits del mensaje.

• La latencia en este esquema viene dada por la siguiente expresión:

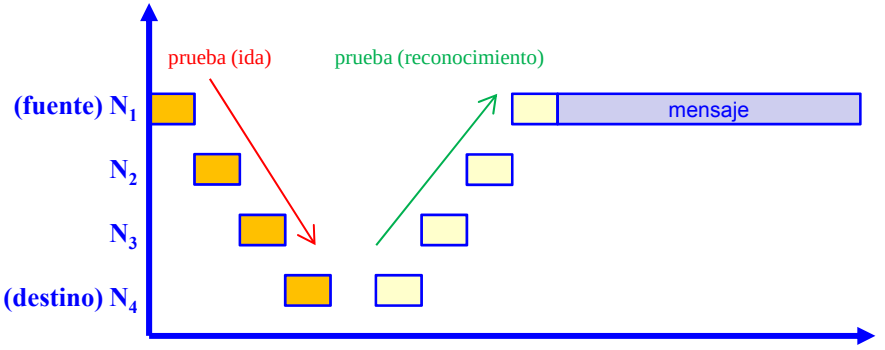
$$T_{CS} = \frac{2P}{W} \cdot D + \frac{M}{W}$$

P = longitud de la prueba en bits

M = longitud del mensaje en bits/segundo

W = ancho de banda

D = distancia en nodos - 1



• Cuando la longitud de la prueba es mucho menor que la del mensaje la latencia es independiente de la distancia entre nodos:

$$Si \;\; P << M \Rightarrow \frac{P}{W} \cdot D \approx 0 \Rightarrow T_{CS} = \frac{M}{W}$$

7. Encaminamiento de mensajes

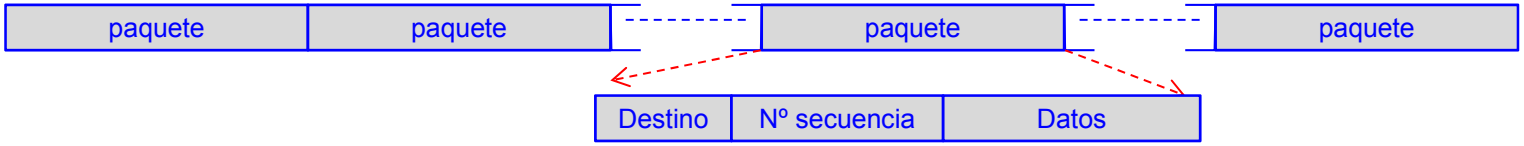
Conmutación de paquetes

- En conmutación de circuitos todo el mensaje se transmite una vez establecido el circuito.
- Una alternativa es dividir el mensaje en paquetes de longitud fija.
- Los primeros bytes del paquete contienen la información de encaminamiento.
- Cada paquete se encamina de forma individual desde el origen al destino.
- Un paquete se almacena completamente en cada nodo intermedio antes de ser enviado al siguiente nodo.
- Esta es la razón por la que a este método de conmutación también se le denomina de almacenamiento y reenvío.
- La información de la cabecera se extrae en los nodos intermedios y se usa para determinar el enlace de salida por el que se enviará el paquete.
- Esta técnica es ventajosa cuando los mensajes son cortos y frecuentes.
- Al contrario que en la conmutación de circuitos, donde un segmento de un camino reservado puede estar sin ser usado durante un período de tiempo significativo, un enlace de comunicación se usa completamente cuando existen datos que transmitir.
- Varios paquetes pertenecientes a un mensaje pueden estar en la red simultáneamente incluso si el primer paquete no ha llegado todavía al destino.
- Sin embargo, dividir un mensaje en paquetes produce una cierta sobrecarga. Además del tiempo necesario en los nodos origen y destino, cada paquete debe ser encaminado en cada nodo intermedio.

7. Encaminamiento de mensajes

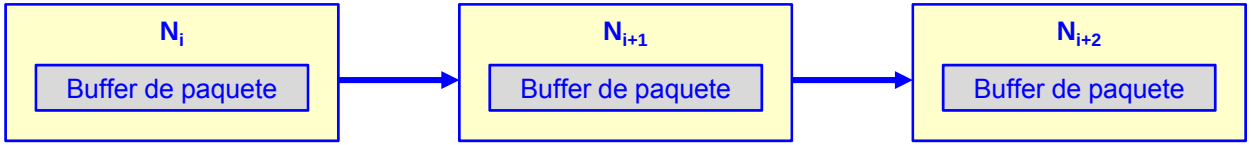
Paso de mensajes por almacenamiento y envío (*store-and-forward)*

• La unidad básica de flujo de información es el paquete que consta de la dirección del nodo destino, el número de secuencia en el mensaje completo y los datos:

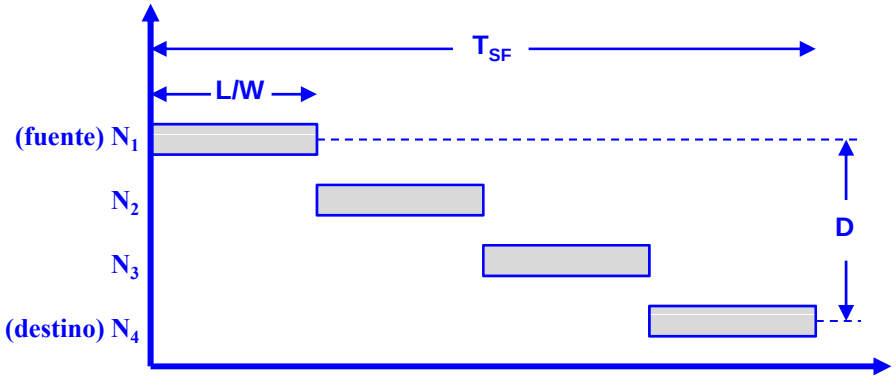


• Un paquete es transferido del nodo fuente al nodo destino a través de una secuencia de nodos intermedios.

• Cada nodo requiere un buffer de paquete para almacenarlo temporalmente hasta que esté disponible el canal de salida.



• La latencia en este esquema es directamente proporcional a la distancia fuente-destino:



$$T_{SF} = \frac{L}{W} \cdot D$$

L = longitud del paquete en bits

W = ancho de banda en bits/segundo

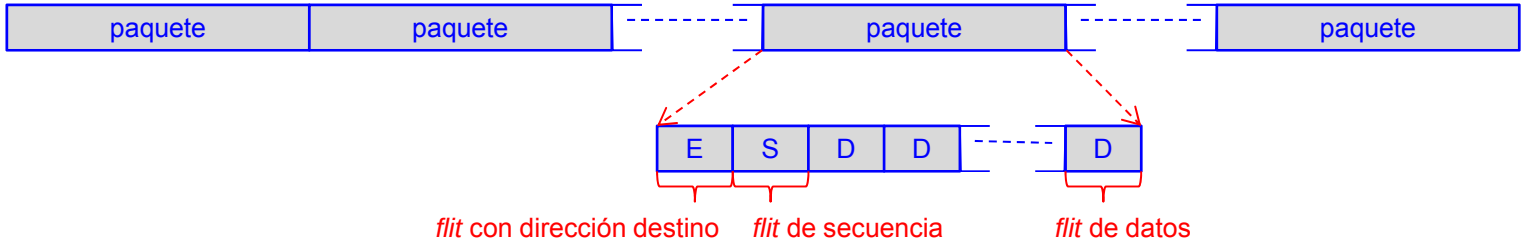
D = distancia en nodos - 1

7. Encaminamiento de mensajes

Paso de mensajes por segmentación asíncrona (*wormhole)*

• Cada paquete se descompone en *flits*: unidad mínima de flujo de transmisión.

• El primer *flit* contiene la dirección destino del paquete y el segundo el número de secuencia del paquete en el mensaje.



• El algoritmo de encaminamiento sólo se aplica al primer *flit*, los restantes *flits* del paquete siguen el camino del primero.

• El tiempo de transmisión entre dos nodos vendrá determinado por la siguiente expresión:

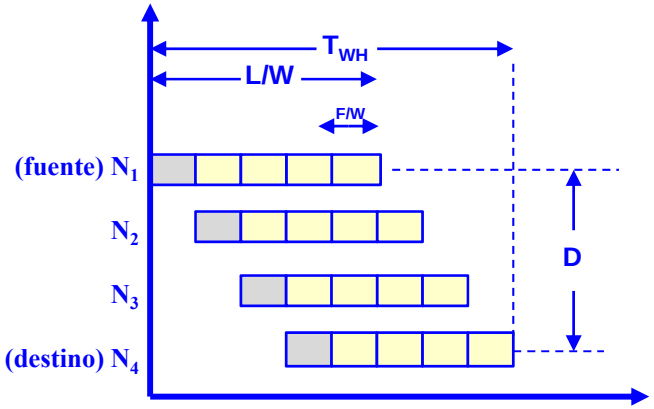
$$T_{WH} = \frac{L}{W} + \frac{F}{W} \cdot D \approx \frac{L}{W} \quad si \quad L >> F$$

L = longitud del paquete en bits

W = ancho de banda en bits/segundo

D = distancia en nodos - 1

F = longitud del flit en bits

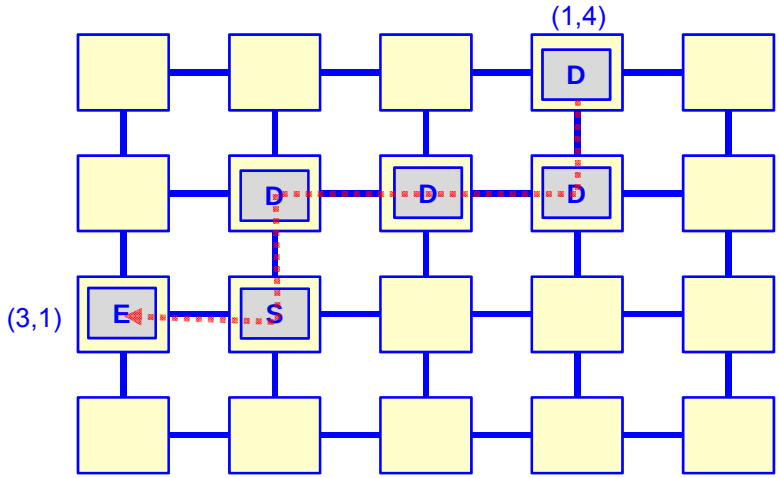


• Como se puede observar cuando la longitud del paquete es bastante mayor que la del flit, el tiempo de transmisión es prácticamente independiente de la distancia entre nodos.

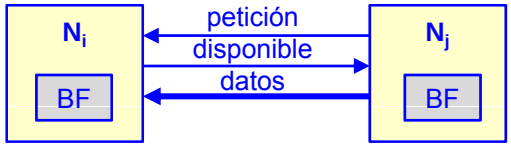
7. Encaminamiento de mensajes

Ejemplo de paso de mensaje con segmentación asíncrona (*wormhole)*

- Consideremos una malla de 4 x 5 nodos.
- Un paquete de 6 *flits* es transmitido del nodo (1,4) al nodo (3,1).
- Cuando el *flit* de cabecera alcanza el nodo destino, una posible ubicación de los restantes *flits* del paquete podría ser la siguiente:



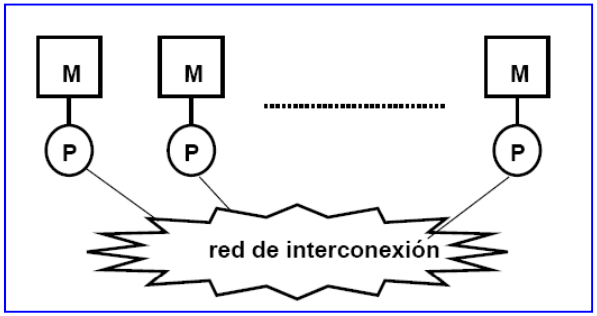
- Cada nodo sólo requiere un buffer con capacidad para almacenar un *flit*.
- Para el control de la transmisión se dispone de una línea de petición de envío y otra de disponibilidad para realizar el envío:



8. Modelo de programación de paso de mensajes

Modelo de Programación Basado en Paso de Mensajes

- Consiste en la replicación del paradigma de programación secuencial
- El programador divide la aplicación en varios procesos que se ejecutan en diferentes procesadores sin compartir memoria y comunicándose por medio de mensajes
- Visión del programador: Lenguaje secuencial con variables privadas + Rutinas de paso de mensajes



- **Envío síncrono y asíncrono (condición de finalización)**

Síncrona:

- El proceso que realiza el envío recibe información sobre la recepción del mensaje
- La comunicación se completa cuando el mensaje ha sido recibido

Asíncrona:

- El proceso únicamente conoce cuando se envía el mensaje (postal)
- La comunicación se completa tan pronto como el mensaje ha sido enviado
- Generalmente copia en un buffer

8. Modelo de programación de paso de mensajes

Introducción a MPI

• MPI (*Message Passing Interface) es un Interfaz estandarizado para la* realización de aplicaciones paralelas basadas en paso de mensajes.

• Define un interfaz de programación independiente de lenguajes, más una colección de *bindings o* instancias de ese interfaz para los lenguajes más extendidos.

• Los procesos tienen un espacio de memoria completamente separado.

• El intercambio de información, así como la sincronización, se hacen mediante paso de mensajes.

• Se dispone de funciones de comunicación punto a punto (que involucran sólo a dos procesos):

- Envío de un mensaje: MPI_Send();
- Recepción de un mensaje: MPI_Recv();

• También dispone de funciones u operaciones colectivas (que involucran a mútiples procesos):

- Barreras de sincronización: MPI_Barrier()
- Broadcast (difusión): MPI_Broadcast()
- Gather (recolección): MPI_Gather
- Scatter (distribución): MPI_Scatter()
- Operaciones de reducción (suma, multiplicación, mínimo, etc.): MPI_Reduce()