1. Revisión de plataformas

Computación de Altas Prestaciones

Grado en Ingeniería de Computadores

Raúl Durán Díaz

Curso académico 2018–2019

Índice

1. Arquitecturas paralelas

1.1. Necesidad de la Computación de Altas Prestaciones

Tendencias de las aplicaciones

La mayor demanda de computación proviene de aplicaciones especializadas en diversos mundos:

- Computación científica.
  - Simulación de fenómenos físicos muy difíciles de observar: evolución de las galaxias, modelado del clima, modelado de la estructura molecular de las proteínas, etc.
- Desarrollo e ingeniería.
  - Simulación de artefactos muy caros o difíciles de construir: diseño de vehículos, perfiles de ala de aviones, túnel de viento, antenas y dispositivos electromagnéticos, etc.
  - Herramientas de CAD.
- Aplicaciones gráficas.
  - Animación y multimedia: *Shrek, Toy Story*.

Tendencias de las aplicaciones

- Computación comercial.
  - Métrica: *on-line transaction processing*, número de transacciones por segundo completadas.
  - *Benchmark*: TPC-C, una aplicación de carga de pedidos, con transacciones interactivas y en *batch*.
  - Los fabricantes ofrecen aplicaciones aptas para ejecutarse en súper-computadores.
- Sistemas empotrados.
  - Seguridad y capacidad de cómputo en dispositivos domóticos, automoción, redes ad-hoc, etc.

Tendencias de las aplicaciones

- *Cloud computing*.
  - El impresionante desarrollo de las comunicaciones permite computación y/o almacenamiento deslocalizados:
    $\Rightarrow$ por ejemplo, herramientas de Google.
  - Hay aplicaciones imposibles de centralizar:
    $\Rightarrow$ por ejemplo, el analizador de partículas del CERN.

Tendencias tecnológicas

La tecnología va evolucionando:

- Reducción permanente de tamaño en las celdas básicas de los VLSI.
  - Ley de Moore: "circuit complexity doubles every 18 months."
  - Transistores, puertas lógicas, circuitos, más pequeños y rápidos.
  - Pero...
    $\Rightarrow$ el número de transistores ha aumentado un orden de magnitud más que la frecuencia de reloj.
- Conclusión: aparición de microprocesadores multi-núcleo y sistemas integrados.

Tendencias tecnológicas

- En el caso de la memoria, tenemos fuerte divergencia capacidad $\Longleftrightarrow$ velocidad.
  - De 1980 a 1995, la capacidad creció 1000 veces...
    - ...pero la velocidad tan solo se duplicó.
  - Algunas posibilidades:
    - reducir la latencia,
    - aumentar el ancho de banda,
    - establecer niveles de jerarquía de memoria,
    - aprovechar la localidad espacial y temporal.
- En el caso de almacenamiento masivo, también aparecen tecnologías jerarquizadas (discos RAID).

Tendencias arquitectónicas

La tecnología determina lo posible: la arquitectura lo traduce a rendimiento activo.

- El paralelismo está presente ya en la arquitectura de las máquinas:
  - a nivel de bits:
    - paso de 4 bits a 8, de 8 a 16, de 16 a 32 y a 64.
  - a nivel de instrucción:
    - procesadores superescalares, caches internos, técnicas de predicción de saltos, etc.
  - a nivel de tareas o procesos:
    - procesadores *multi-core*.

1.2. Taxonomía de las plataformas

Doble visión de las plataformas

Organización lógica

Es la visión que el programador tiene de la plataforma de computación.

Organización física

Se trata del *hardware* propiamente dicho, es decir, los hierros mondos y lirondos.

Organización lógica

- La visión del programador comprende dos aspectos:
  - cómo expresar las tareas paralelas;
  - cómo expresar la interacción entre tales tareas.

Tareas: proceso o flujo de ejecución

Definición 1. Un *proceso* es un *flujo de ejecución*, es decir, una lista ordenada de instrucciones que usan y/o actúan sobre un determinado conjunto de posiciones de memoria y/o registros del procesador.

Comentario 2. Obsérvese que aquí es crucial la actuación *sucesiva* de las instrucciones. La lista de instrucciones suele ser un subconjunto de un programa completo.

Interacción entre tareas

Paradigma de memoria compartida
Varios procesos comparten el mismo conjunto de posiciones de memoria que usan y/o sobre el que actúan. En este contexto, los procesos se suelen denominar *hebras*.

Paradigma de paso de mensajes
El conjunto de posiciones de memoria es exclusivo de cada proceso. La interacción se produce mediante envíos y recepciones explícitas de mensajes.

Modelo o paradigma de programación

- Un modelo se realiza en términos de las *primitivas de comunicación* de nivel de usuario, disponibles en un sistema.
- Las primitivas se implementan
  - en hardware;
  - en el sistema operativo;
  - mediante librerías (específicas de un hardware) que se ejecutan a nivel de usuario.

Memoria compartida

- Paradigma de memoria compartida:
  - es como un tablón de anuncios, que permite comunicar cosas a otros colegas;
  - cada uno puede escribir una nota y clavarla en el tablón;
  - y los demás pueden leerlas;
  - las actividades se pueden coordinar, tomando nota de quién está haciendo qué.

Paso de mensajes

- Paradigma de paso de mensajes:
  - es equivalente a una llamada telefónica, o enviar una carta;
  - existen mensajes bien claros, que indican lo que se puede o debe hacer y cuándo;
  - la actividad se coordina mediante el propio envío de los mensajes;
  - no existen lugares comunes (como el tablón de anuncios anterior) donde todos puedan ver lo de todos.

Paralelismo de datos

- Paradigma de paralelismo de datos:
  - la acción se parece a un ejército en orden cerrado;
  - un conjunto de agentes operan sobre diversos conjuntos de datos;
  - realizan ordinariamente la misma acción simultáneamente;
  - después intercambian información (por alguno de los procedimientos antes examinados);
  - siguen adelante actuando sobre un nuevo conjunto de datos;
  - la coordinación es centralizada, como la voz ejecutiva en el orden cerrado.

Taxonomía de Flynn

Definida por vez primera por M.J. Flynn[1].

- SISD:
  - un flujo de instrucciones único trabaja sobre flujo de datos único (arquitectura clásica, superescalares).
- SIMD:
  - un flujo de instrucciones único trabaja sobre un flujo de datos múltiple (computadores matriciales). Relacionado con el paradigma de *paralelismo de datos*.

Taxonomía de Flynn

- MIMD:
  - un flujo de instrucciones múltiple trabaja sobre un flujo de datos múltiple (multiprocesadores).
- SPMD:
  - es igual al anterior, pero los flujos de instrucciones (iguales o distintos) están contenidos en un único programa. La ejecución se condiciona a la identificación del proceso ejecutor.

[1] M. Flynn, Some Computer Organizations and Their Effectiveness, *IEEE Trans. Comput.* C-21, pp. 948–960, 1972.

Organización física: arquitecturas paralelas

Definición 3. Un computador paralelo es un conjunto de elementos de proceso que se comunican y cooperan para la resolución rápida de grandes problemas[2].

Estructura de una arquitectura paralela

- Las arquitecturas paralelas pueden considerarse una extensión de las arquitecturas convencionales para atacar los problemas relativos a la comunicación y cooperación entre los distintos elementos de proceso.
- Una arquitectura paralela es
  - una arquitectura convencional, más
  - una arquitectura de comunicaciones.

Definición de arquitectura de comunicaciones

Definición 4. La *arquitectura de comunicaciones* define el conjunto de operaciones de comunicación/sincronización disponibles en modo usuario, el formato de estas operaciones y los tipos de datos sobre los que operan.

Comentario 5. Obsérvese el paralelismo con un *repertorio de instrucciones* convencional.

División de los sistemas MIMD

- Los sistemas MIMD caen, *grosso modo*, en uno de estos tipos:
  - Sistemas de memoria compartida.
  - Sistemas de memoria físicamente distribuida.

1.3. Medidas de rendimiento

Micro *benchmarks*

- **LMBENCH**:
  - Mide la sobrecarga debida al sistema operativo y la capacidad de transferencia entre procesador y cache, memoria, red y discos.
  - Mide también la sobrecarga en llamadas al sistema Unix.
  - Es sencilla pero útil.
- **STREAM**:
  - Mide específicamente la velocidad de transferencia de memoria en régimen permanente, y la velocidad de cálculo correspondiente.
  - Propone una tríada de operaciones:

[2] G.S. Almasi, A. Gottlieb, *Highly Parallel Computing*, Benjamin/Cummings, Redwood City, California, 2ª edición, 1989.

Operaciones del STREAM

Nombre	Código	Bytes/iter	Flop/iter
COPY	`a(i) = b(i)`	16	0
SCALE	`a(i) = q` $\times$ `b(i)`	16	1
SUM	`a(i) = b(i)` $+$ `c(i)`	24	1
TRIAD	`a(i) = b(i)` $+$ `q` $\times$ `c(i)`	24	2

Los vectores `a`, `b`, `c`, son de millones de elementos, y cada elemento es un flotante doble de 8 bytes.

Métrica *equilibrio de máquina*

- STREAM proporciona una métrica llamada *equilibrio de máquina*, E, definida:

$$E = \frac{V_p}{AB_s}$$

  donde V_p es la *velocidad pico* y AB_s es el *ancho de banda*.

- Se interpreta como el número de operaciones flotantes que se pueden ejecutar durante la lectura/escritura de un elemento del vector.

Resultados de la métrica

Máquina	V_p (Mflop/s)	AB_s (MW/s)	E
Cray C90	15360.0	12976.5	1.2
Cray T932 321024-3E	57600.0	44908.8	1.3
HP AlphaServer GS1280 1300	166400.0	53931.2	3.1
IBM System p5 595	588800.0	25780.4	22.8
SGI Altix 3000	3276800.0	139989.1	23.4
Sun F15K	151200.0	6340.5	23.8
Sun F25K-1050-72chip	302400.0	9512.2	31.8
HP Integrity SuperDome	819200.0	21354.1	38.4

Comentarios a los resultados de la métrica

- Cray y Digital (el HP AlphaServer es heredado) han prestado mucha atención a este parámetro.
- Conforme sube la velocidad del procesador, resulta más difícil mantener un buen E.
- Sun, con "baja" velocidad, tiene en cambio mal rendimiento de memoria (compárese Sun F15K con el HP AlphaServer).

2. Sistemas de memoria compartida

Sistemas de memoria compartida

- Todos los procesadores acceden a idéntico mapa de memoria.
- La comunicación es implícita:
  - ocurre como resultado de `load`s y `store`s convencionales.
- Se usa en entorno de multi-programación y, por supuesto, en programación paralela.

Paralelismo a nivel de hebras

- Las escrituras que una hebra realiza sobre una dirección compartida son visibles a las lecturas de las demás hebras.
- La cooperación y coordinación entre las hebras se lleva a cabo mediante
  - la lectura y escritura de variables compartidas, y
  - operaciones atómicas para sincronización.

Ejemplo 6. Es el paradigma usado en el sistema operativo, cuyas estructuras son visibles para todos los procesos, si bien son accedidas ordinariamente por código de nivel de sistema.

Estructura del espacio de direcciones

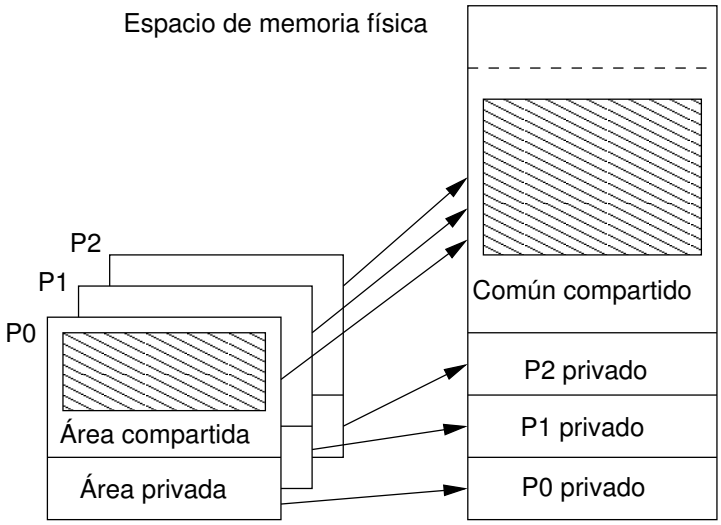

Figura 1: Estructura del espacio de direcciones

Memoria compartida

- Características de los sistemas de memoria compartida:
  - Pequeño número de procesadores (del orden de decenas).
  - Memoria interconectada por diversos medios.
  - Grandes caches para ocultar la latencia.
  - Tiempo de acceso a memoria uniforme para todos los procesadores: se habla de máquinas SMP[3], con arquitectura tipo UMA[4].

[3] *Symmetric Multiprocessors*.
[4] *Uniform Memory Access*.

Comunicaciones en sistemas de memoria compartida

- El hardware de comunicaciones es una extensión natural del hardware estándar.
- El controlador de memoria permite que cada procesador y un conjunto de controladores de entrada/salida accedan a los módulos de memoria.
- La interconexión puede ser de diversos tipos.

Comunicaciones en sistemas de memoria compartida

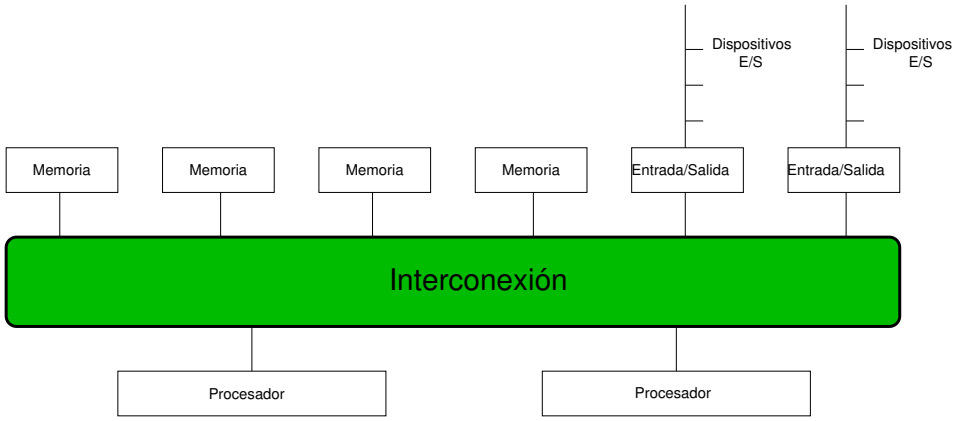

Figura 2: Estructura genérica de comunicaciones

Conectividad por conmutador de barras

Una forma de conseguirlo es interconectar mediante un conmutador de barras cruzadas.

- Ventajas:
  - Buena velocidad de interconexión.
  - El número de dispositivos no influye en la velocidad.
- Inconvenientes:
  - Dificultad de escalado, por el alto coste del conmutador de barras.

Conectividad por conmutador de barras

Conectividad por red multietapa

Una alternativa es utilizar una conectividad por red multietapa.

- Ventajas:
  - mejor escalabilidad;
  - el coste sube más lentamente conforme se aumentan los puertos.
- Inconvenientes:
  - aumento de latencia;
  - disminución de ancho de banda;
  - en ciertos casos, incompatibilidad de conexiones.

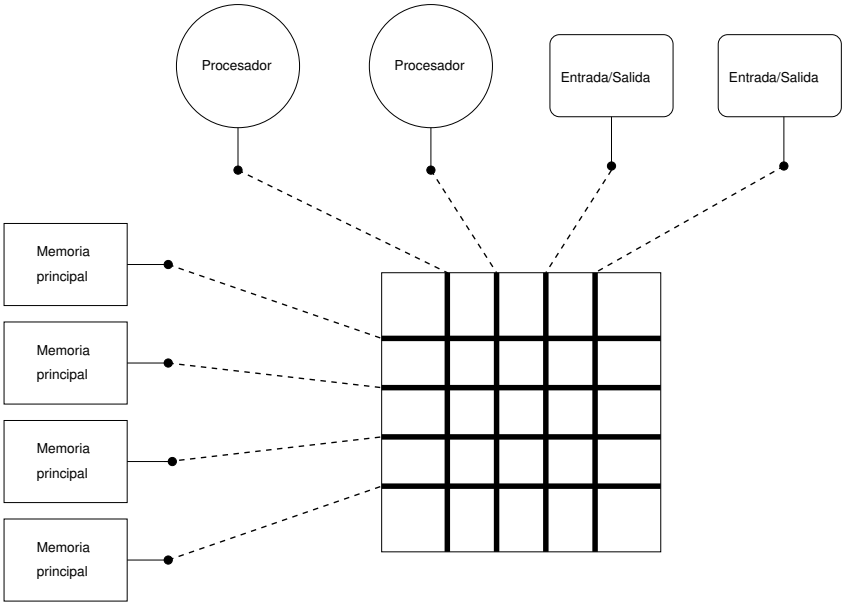

Figura 3: Conectividad por conmutador de barras

Conectividad por red multietapa

Conectividad por bus interno

- A partir de los años 80, se hace lugar común el disponer de placas en donde procesador, memoria, caches, aceleradores de punto flotante conviven.
- Los sistemas se organizan alrededor de un bus común.
- Los mecanismos de acceso al bus permiten a cualquier procesador (o controlador de entrada/salida) acceder a cualquier posición de memoria.
- Cualquier posición de memoria es accesible en el mismo tiempo.

Conectividad por bus interno

Sistema ejemplo: Sun Enterprise Server

- Otro ejemplo de más alto nivel: Sun Enterprise Server.
- Cada placa puede acomodar:
  - o bien un sistema completo, con
    - dos procesadores,
    - cache de primer nivel, L_1, con 16 KB,
    - cache de segundo nivel, L_2, con 512 KB,
    - dos bancos de memoria con un ancho de 512 bits, y
    - un controlador de acceso al bus.
  - o bien ...

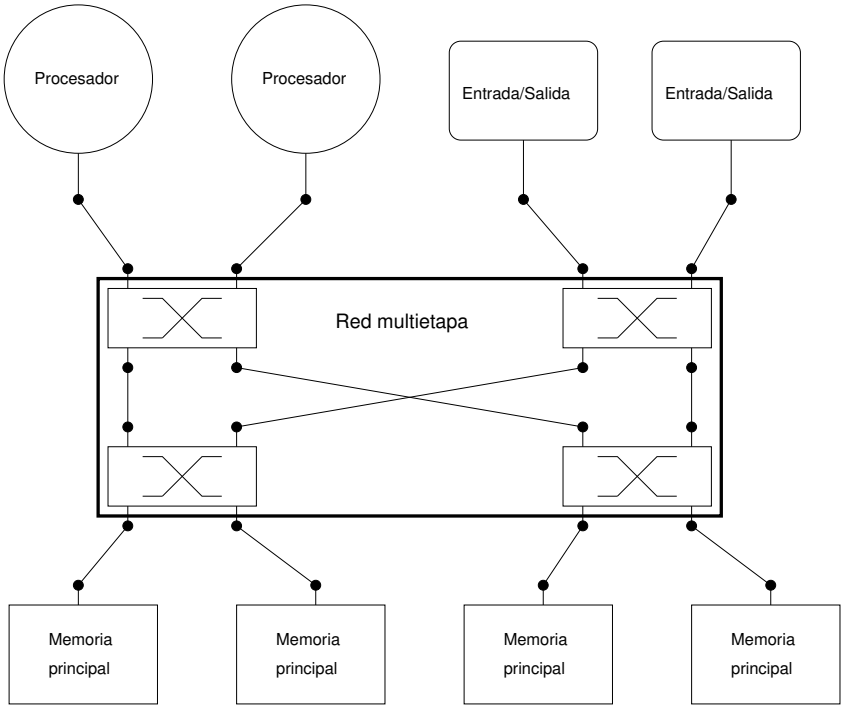

Figura 4: Conectividad por red multietapa

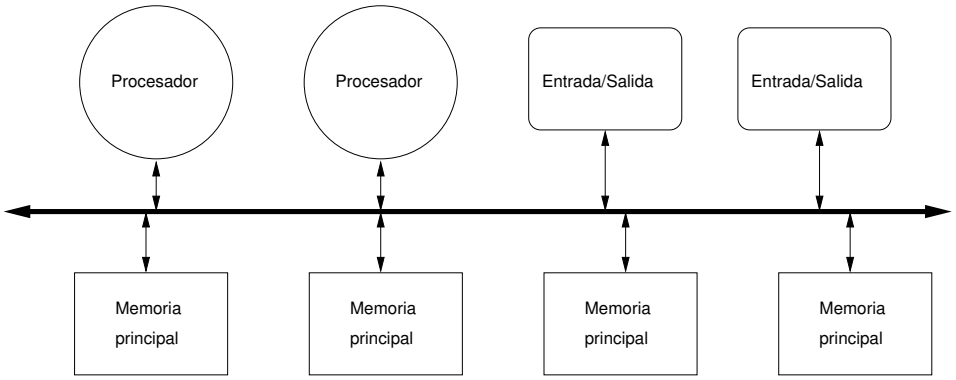

Figura 5: Conectividad por bus interno

Sistema ejemplo: Sun Enterprise Server

- . . . puede acomodar:
  - o bien una placa de entrada/salida dotada de
    - tres ranuras SBUS, para dispositivos de E/S,
    - un conector SCSI,
    - un puerto ethernet, 100bT,
    - dos interfaces FiberChannel.
- El bus principal opera a 83 MHz;
  - el ancho de banda pico es 2.5 GB/s.
- Una configuración completa soporta hasta 16 tarjetas de cada tipo, con un mínimo de una de cada tipo.

Sistema ejemplo: Sun Enterprise Server

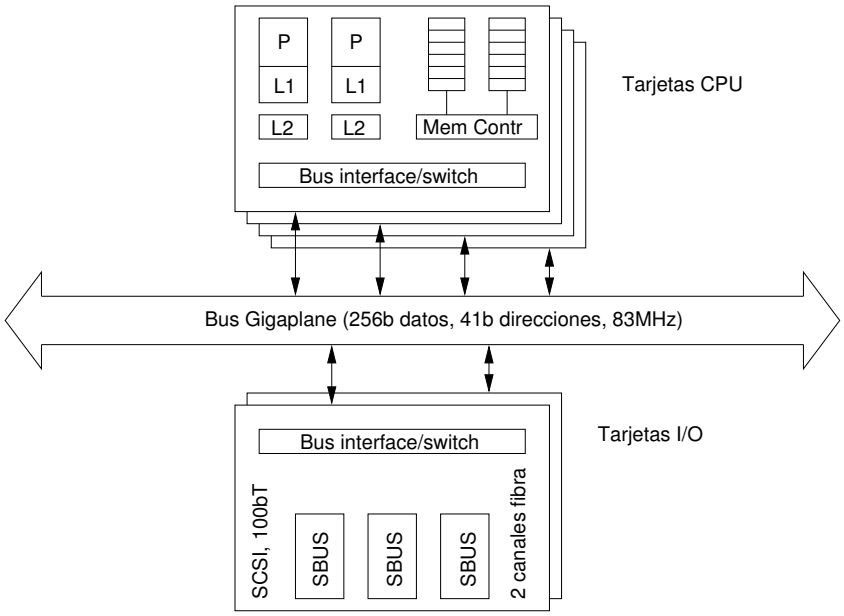

Figura 6: Sun Enterprise Server

Análisis de la conectividad mediante bus

- Ventajas:
  - añadir un procesador no cuesta nada.
- Inconvenientes:
  - el ancho de banda es fijo, y ha de repartirse entre los procesadores existentes.
- Conclusión: la *escalabilidad* es deficiente.

Análisis de la conectividad mediante bus

- El uso de caches puede aliviar la congestión, dada la localidad de los datos.
- Sin embargo, introduce el problema de la "coherencia" de los datos en los distintos caches.

¿Cómo escalar un sistema de memoria compartida?

La pregunta es: ¿qué podemos hacer para escalar nuestra máquina?

- El escalado en procesadores es fácil: añadimos más.
- Claramente el problema es la conectividad entre ellos.

Escalado de la conectividad

- Vemos que la interconexión se escala de modo deficiente, pues
  - el bus se escala mal: el ancho de banda es fijo;
  - el conmutador de barras escala mal: su coste aumenta como el cuadrado del número de puertos.
- En resumen, hay que buscar una red de interconexión, escalable, tal que permita añadir fácilmente nuevos procesadores sin incurrir en un coste excesivo.

Escalado de la conectividad

- Un modo natural de hacerlo es mantener el tiempo de acceso uniforme.
- Cada acceso a memoria se traduce en un mensaje que circula por la red.
- Ventaja: el tiempo de acceso es el mismo para todos.
- Inconveniente: ese tiempo puede ser muy alto.
- Este esquema se llama "salón de baile".

Escalado de la conectividad

- Otro esquema es interconectar máquinas con memoria local.
- Cuando el usuario quiere acceder a una dirección, el controlador local decide si el acceso corresponde a memoria local o debe traducirse en un mensaje por la red.
- El tiempo de acceso ya no es uniforme: el acceso local es (mucho) más rápido que el remoto.
- Este esquema de acceso se llama NUMA[5].

[5] *Non Uniform Memory Access*

Análisis del esquema NUMA

- Ventajas:
  - si hay localidad de datos, disminuye el número de accesos remotos;
  - la disminución de accesos remotos rebaja los requisitos de ancho de banda de la red;
  - el acceso remoto no influye sobre la velocidad de acceso local.
- Inconvenientes:
  - el acceso remoto es lento;
  - el programador (y el programa) han de tenerlo en cuenta, por lo que el software debe ser adaptado;
  - las adaptaciones del software son válidas solo para una organización concreta de memoria.

Extensión del esquema NUMA

- En las máquinas NUMA los datos pueden residir en cualquier parte, es necesario garantizar la coherencia entre los caches de las distintas máquinas.
- Cuando existen unas estructuras de tipo hardware que garantizan esa coherencia, hablamos de máquinas ccNUMA[6].

Sistema ejemplo: Cray T3E

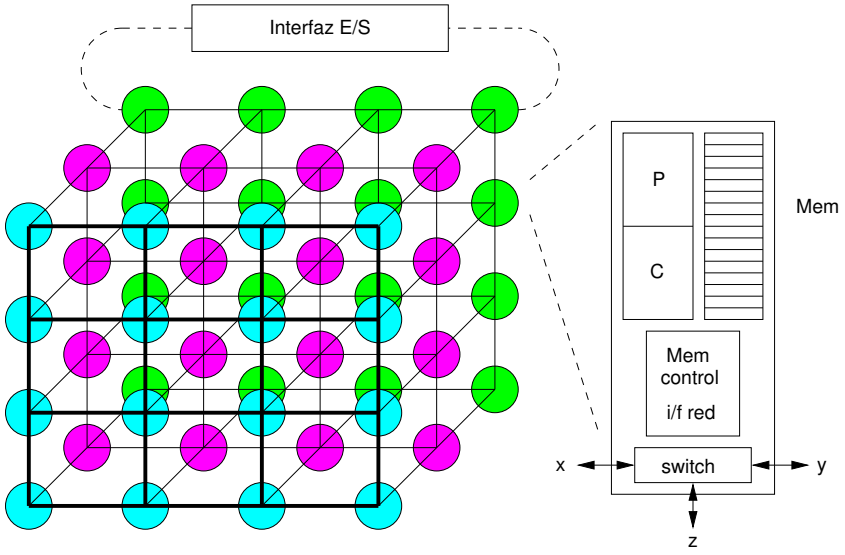

Figura 7: Cray T3E

[6] *cache coherent Non Uniform Memory Access.*

Sistema ejemplo: Cray T3E

- Cada nodo contiene un procesador HP-Alpha, memoria local, una interfase de red integrada en el controlador de memoria, y un conmutador.
- Cada nodo está conectado con sus seis vecinos, a través de un enlace de 650 MB/s de velocidad.
- Los nodos de la "superficie" están conectados a la red de E/S.

Sistema ejemplo: Cray T3E

- Para leer o escribir en una posición de memoria de otro procesador, es necesario un protocolo de inicio de transferencia.
- Establecido el canal, se realizan los `loads` y `stores` como de costumbre.
- El mensaje se encamina a través de todos los nodos necesarios, con un pequeño retardo por "salto".
- Los datos remotos no se almacenan en cache, pues no está previsto ningún mecanismo que garantice la "coherencia".
- Atención: la distribución de memoria entre los distintos procesadores ha de ser conocida por el programador/compilador.

Resumen de la estructura de memoria compartida

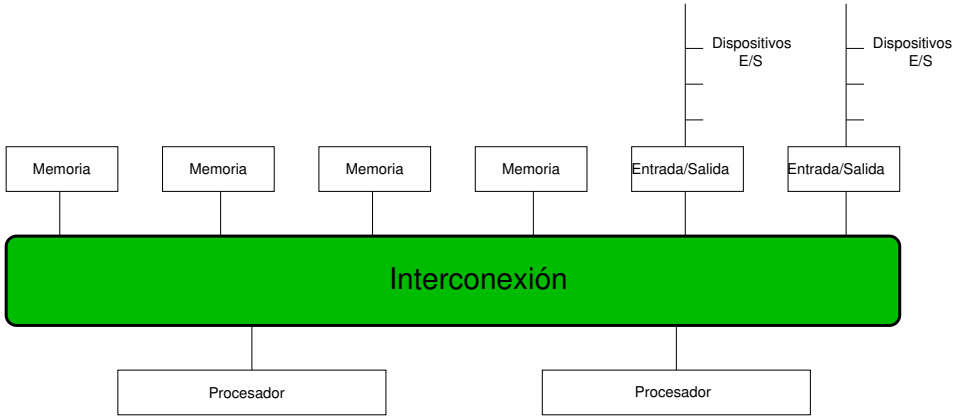

Figura 8: Estructura general

Resumen de la estructura de memoria compartida

- En resumen, en un modelo de programación en memoria compartida:
  - cada procesador ejecuta una tarea, igual o distinta a la de los demás, de manera independiente;
  - cada tarea se comunica y coopera con las demás leyendo y escribiendo variables compartidas;
  - estas operaciones se corresponden directamente con instrucciones de tipo `load` y `store`;

- el modelo y programación y la abstracción de comunicaciones están muy próximas al hardware real.
- el acceso a las direcciones de memoria —locales o remotas— se resuelven directamente en el hardware, sin intervención del programador ni del sistema operativo.

Resumen de la estructura de memoria compartida

- La eficiencia de este tipo de sistemas viene dada, fundamentalmente, por:
  - la capacidad del compilador/programador para optimizar la distribución de las variables en la memoria local y remota;
  - la latencia en que se incurre al acceder a memoria;
  - el ancho de banda de transferencia de memoria.

3. Sistemas de memoria distribuida

Memoria distribuida

- Cuando el número de procesadores sube
  - no es posible soportar la creciente demanda de ancho de banda de memoria,
  - además de que sube la demanda exigida a cada procesador.

Memoria distribuida

- Ventajas de la memoria distribuida:
  - Fácil de escalar a buen coste.
  - Si la mayoría de los accesos a memoria son locales, se reduce el tiempo de acceso medio.
- Inconvenientes de la memoria distribuida:
  - La latencia de acceso a datos remotos puede aumentar mucho.
  - El hardware puede llegar a ser muy complicado si se desean altas velocidades de respuesta.

Memoria distribuida

- Los bloques básicos para esta arquitecturas son máquinas completas, incluyendo procesador, memoria y dispositivos de E/S.
- El diagrama es básicamente el mismo que en una arquitectura NUMA.
- En este caso, la comunicación está integrada en el subsistema E/S y no en los controladores de memoria.
- Análogo a un cluster de estaciones, pero con densa integración de los nodos.
- También la integración entre procesador y red es mucho más intensa, proporcionando así mucha más velocidad de transferencia que en un sistema cluster tradicional.

Paradigma de paso de mensajes

- En estos sistemas, hay gran distancia entre el modelo de programación y el hardware que, de hecho, realiza la comunicación.
- Las operaciones de comunicación suelen ser complejas e involucran muchas instrucciones.
- El usuario necesita, de ordinario, realizar llamadas al sistema operativo y/o usar una biblioteca de programación.

Paradigma de paso de mensajes

- Las operaciones básicas son el *envío* y la *recepción* de un mensaje. Todas las demás suelen ser variantes de éstas.
- El *envío* especifica, típicamente:
  - un buffer a enviar,
  - un tamaño,
  - un proceso receptor (pueden ser varios) que reside, típicamente, en un procesador remoto.
- La *recepción* especifica, típicamente:
  - un proceso, normalmente de una máquina remota, del que se va a recibir,
  - un buffer en donde depositar lo recibido,
  - un tamaño.

Paradigma de paso de mensajes

- La transferencia tiene lugar cuando coinciden el envío y la recepción en los correspondientes procesos.
- Normalmente se puede añadir una etiqueta identificadora que ha de coincidir en ambas operaciones.
- La combinación de un envío y una recepción coincidente determina una sincronización mutua y una transferencia de datos de memoria privada a memoria privada.

Paradigma de paso de mensajes

Paradigma de paso de mensajes

- Hay distintas posibilidades de sincronización para el envío y la recepción.
- Para el envío:
  - esperar a que se verifique la recepción;
  - esperar a liberar el buffer de envío;
  - encargar la operación.

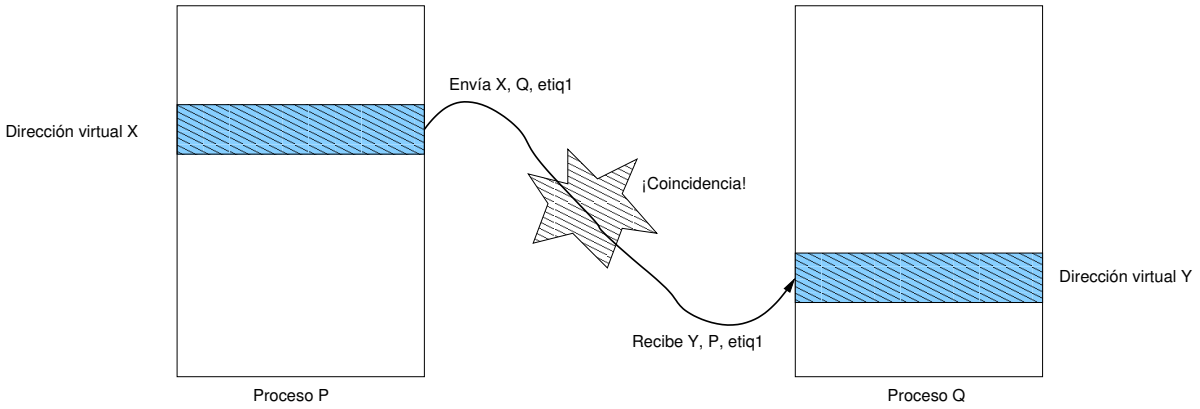

Figura 9: Paso de mensajes

Paradigma de paso de mensajes

- Para la recepción:
  - esperar a que se verifique el envío;
  - simplemente encargar la operación y comprobar si ha finalizado mediante *polling*.

Paradigma de paso de mensajes

En la programación usando este paradigma

- los programas están muy bien estructurados;
- casi siempre todos los nodos ejecutan la misma copia de un programa, con idéntico código y variables privadas (SPMD);
- los procesos pueden nombrarse unos a otros mediante algún mecanismo sencillo: normalmente, cada proceso recibe un número, de cero en adelante.

Ejemplo de arquitectura: IBM SP-2

- Máquina paralela escalable.
- Cada nodo es una estación RS/6000.
- La modificación es insertar una tarjeta de red, TR, en el MicroChannel, que consta de:
  - interfaz de red, IR,
  - memoria para almacenar los mensajes,
  - una tarjeta DMA,
  - un microcontrolador i860, para mover los datos entre la red y la memoria del procesador.

Ejemplo de arquitectura: IBM SP-2

- La red consta de
  - ocho conmutadores de ocho puertos,
  - interconectados en forma de mariposa.
  - El ancho de banda es 40 MB/s en ambos sentidos.

Ejemplo de arquitectura de paso de mensajes

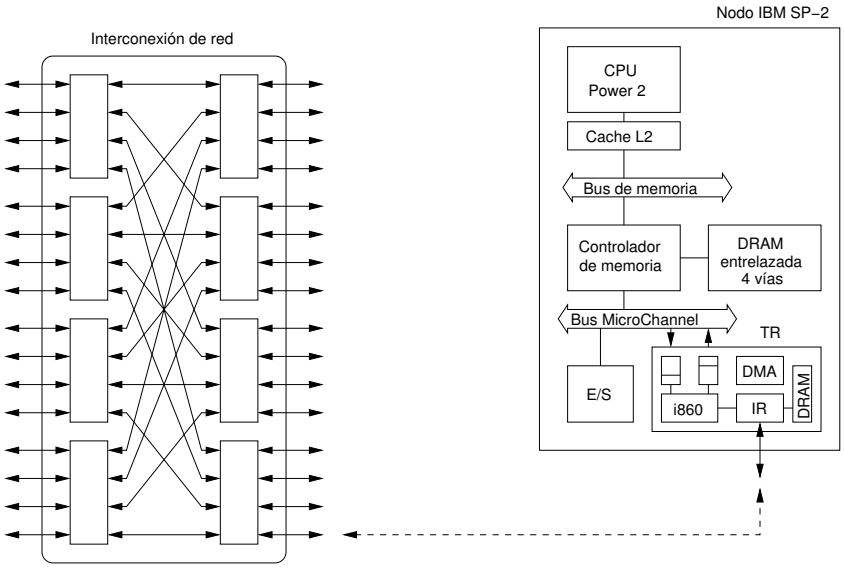

Figura 10: Arquitectura del IBM SP-2

Clusters

- Máquinas sencillas y baratas interconectadas mediante conmutadores (*switches*) estándar.
- Atractivas porque se puede crear un gran sistema a un coste muy bajo.
- Los nodos no son accesibles individualmente, sino mediante un *gestor de trabajos*.
- Los clusters son agrupables en **grids** que, mediante un *middleware*, realizan una colaboración coordinada de los clusters que lo integran.

Clusters

- Aptas para aplicaciones con un acoplamiento débil:
  - bases de datos,
  - servidores de ficheros,
  - servidores *web*,
  - ...
- Evolución del uso de clusters en distintos entornos:
  - corporativo ingenieril,
  - comercial,
  - científico,
  - ...

Clusters Beowulf

- Arquitectura estándar de un cluster:
  - organización del computador básico;
  - tendencia a la simplicidad;
  - interconexiones rápidas;
  - soluciones de almacenamiento:
    - arquitectura SAN (*storage area network*),
    - arquitectura NAS (*network-attached storage*).

4. Espacio de diseño de las redes de interconexión

4.1. Introducción y conceptos generales

Espacio de diseño de las redes de interconexión

Analizaremos los siguientes aspectos:

1. Topología.
2. Estrategia de conmutación.
3. Algoritmo de encaminamiento.

Topología

Definición 7. Definimos como *topología* la estructura de interconexión física de una red.

Comentario 8. La *topología* se puede modelar como un grafo cuyos vértices son los conmutadores o las interfaces de red y los arcos son los enlaces entre conmutadores o entre conmutadores e interfaces. El grafo puede servir para estudiar cómo influye la topología en las prestaciones de la red.

Estrategia de conmutación

- La *estrategia de conmutación* determina cómo atraviesan los datos de un paquete su camino hasta llegar al destino. Hay dos posibilidades típicas:
  - *Conmutación de circuitos*
    - El camino entre fuente y destino se reserva con antelación, antes de comenzar la transmisión del mensaje, y se conserva hasta que ésta finaliza.
  - *Conmutación de paquetes*
    - El mensaje se trocea en paquetes, que se envían individualmente, siguiendo cada uno su camino, potencialmente distinto.

Algoritmo de encaminamiento

- El *algoritmo de encaminamiento* determina la secuencia de enlaces y conmutadores que el paquete ha de seguir para llegar desde su fuente a su destino.
- Dependiendo del tipo de red, puede existir uno o varios caminos para alcanzar un destino desde una fuente.
- Si existen varios caminos, el algoritmo determina cuál es el mejor en función de ciertos criterios: nivel de ocupación, fallo de algún recurso, etc.
- Entraremos muy levemente en este punto.

Parámetros característicos de una red

- *Tamaño de la red*:
  - número total de nodos que la componen.
- *Grado de un nodo*:
  - número de enlaces que inciden en un nodo.
- *Diámetro de la red*:
  - es la longitud del *camino más largo* elegido dentro del *conjunto de los caminos mínimos entre dos nodos cualesquiera*.
- *Simetría*:
  - una red es simétrica si sus nodos son indistinguibles, desde el punto de vista de su conexión con los demás.
- *Ancho de bisección*:
  - número de enlaces que atraviesan el plano imaginario que divide la red en dos mitades iguales en cuanto a número de conmutadores, y nodos terminales.

4.2. Topología

Topología y clasificación de las redes

Atendiendo a la topología, podemos clasificar las redes así:

- Estáticas:
  - ortogonales (cubo, hipercubo, toro, malla);
  - no ortogonales (árbol).
- Dinámicas:
  - barras cruzadas;
  - medio compartido (buses);
  - multietapa:
    - bloqueante,
    - reconfigurable,
    - no bloqueante.

Redes estáticas

- Son características de este tipo de redes:
  - una fuerte integración en el nodo entre la interfaz de red y los conmutadores (cada conmutador se asocia a un nodo);
  - la red visible se forma por las conexiones punto a punto entre los nodos;
  - la conexión interfaz de red–conmutador es muy rápida, quizá integrada en una sola pastilla.

Redes estáticas ortogonales

Definición 9. Una red estática presenta una *topología ortogonal* de dimensión n si cada nodo puede asociarse con un elemento del conjunto $\mathbb{Z}_{k_1} \times \cdots \times \mathbb{Z}_{k_n}$, al que llamamos su coordenada, y dos nodos están interconectados si y solo si la diferencia de sus coordenadas es de la forma $(0,\ldots,\pm 1_{(i},\ldots,0)$ con $1 \leq i \leq n$.

Comentario 10. Obviamente, el número de nodos en la red será $k_1 \cdot k_2 \cdots k_n$.

Tipos de redes estáticas ortogonales

- Existen varios tipos de redes estáticas ortogonales:
  - redes malla;
  - redes toro;
  - redes hipercubo.

Redes ortogonales tipo malla

Definición 11. Una red estática ortogonal es de *tipo malla* si dos nodos $A = (a_1, a_2, \ldots, a_n)$ y $B = (b_1, b_2, \ldots, b_n)$ están interconectados solo cuando se verifique que existe uno y solo un j, $1 \leq j \leq n$, tal que $a_j - b_j = \pm 1$ y $a_i = b_i$ para todo $i \neq j$, $1 \leq i \leq n$.

Red estática tipo malla

Redes ortogonales tipo toro

Definición 12. Una red estática ortogonal se llama de *tipo toro* cuando dos nodos $A = (a_1, a_2, \ldots, a_n)$ y $B = (b_1, b_2, \ldots, b_n)$ están interconectados si se verifica que existe uno y solo un j, $1 \leq j \leq n$, tal que $a_j - b_j \pmod{k_j} \equiv \pm 1$ y $a_i = b_i$ para todo $i \neq j$, $1 \leq i \leq n$.

Comentario 13. Obsérvese que esta red es simétrica.

Red estática tipo toro

Redes ortogonales tipo hipercubo

- Una red ortogonal es de *tipo hipercubo* n-dimensional si se trata de un toro en el que $k_i = \{0,1\}$ para $1 \leq i \leq n$.
- En ese caso, las coordenadas de cualquier nodo se pueden expresar como un número binario. Dos nodos están interconectados si y solo si sus coordenadas difieren en un único bit.
- Por lo tanto, cada nodo está conectado a otros n nodos.

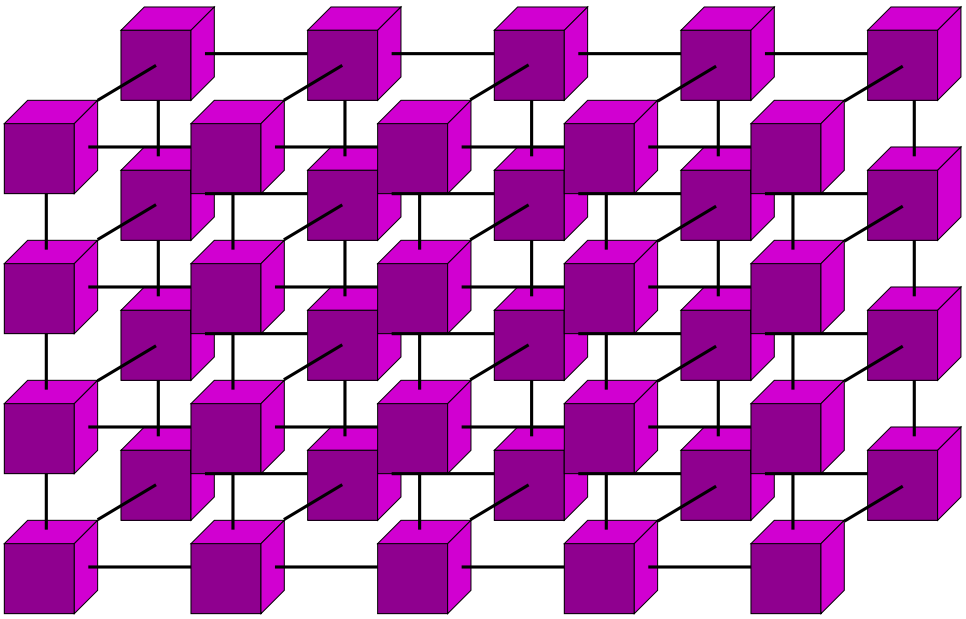

Figura 11: Red estática tipo malla tridimensional $2 \times 5 \times 4$

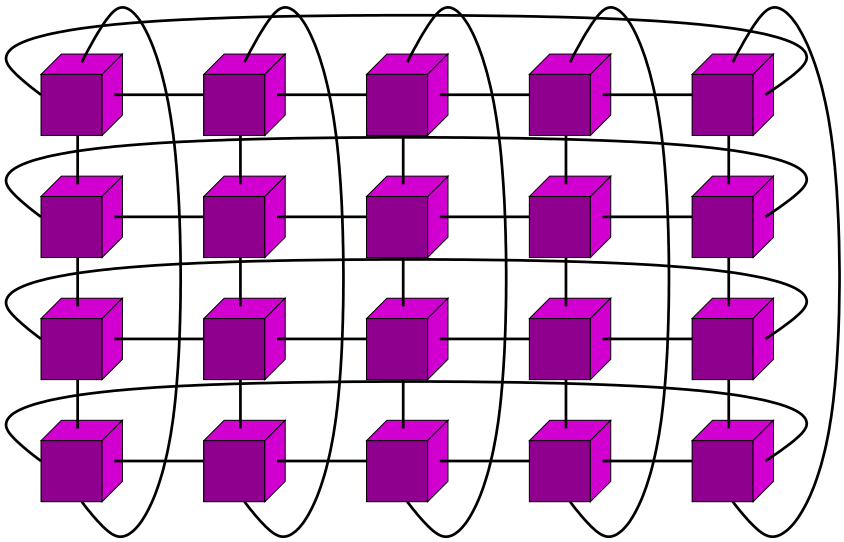

Figura 12: Red estática tipo toro bidimensional 4×5

Redes no ortogonales

- Normalmente, este tipo de redes no son simétricas.
  - Red de tipo árbol:
    - presentan cuellos de botella con distribución de tráfico uniforme: todas las comunicaciones han de pasar por el nodo raíz.
  - Red tipo estrella:
    - mismo problema que antes.

Red tipo árbol

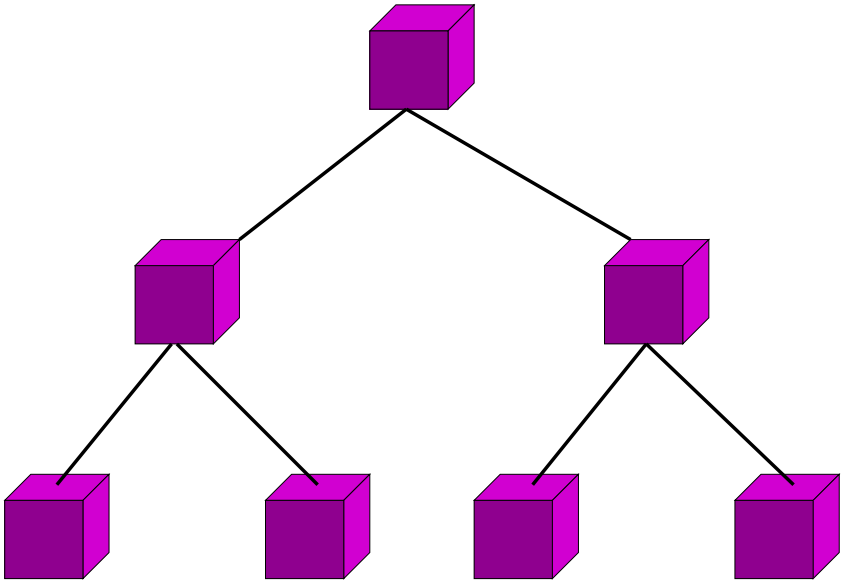

Figura 13: Red tipo árbol

Red tipo árbol grueso

Tabla para redes estáticas ortogonales, $N = k^n$

Parámetros	Malla	Toro	Hipercubo
Grado	$2\log_k N$	$2\log_k N$	$\log_2 N$
Diámetro	$(k-1)\log_k N$	$\lfloor \frac{k}{2} \rfloor \log_k N$	$\log_2 N$
Bisección	N/k	$2N/k$	$N/2$
Nodos	N	N	N

Redes dinámicas: barras cruzadas

- Red de barras cruzadas:
  - se basan en un conmutador que permite cualquier aplicación biyectiva entre entradas y salidas,
  - las interconexiones pueden ser simultáneas,

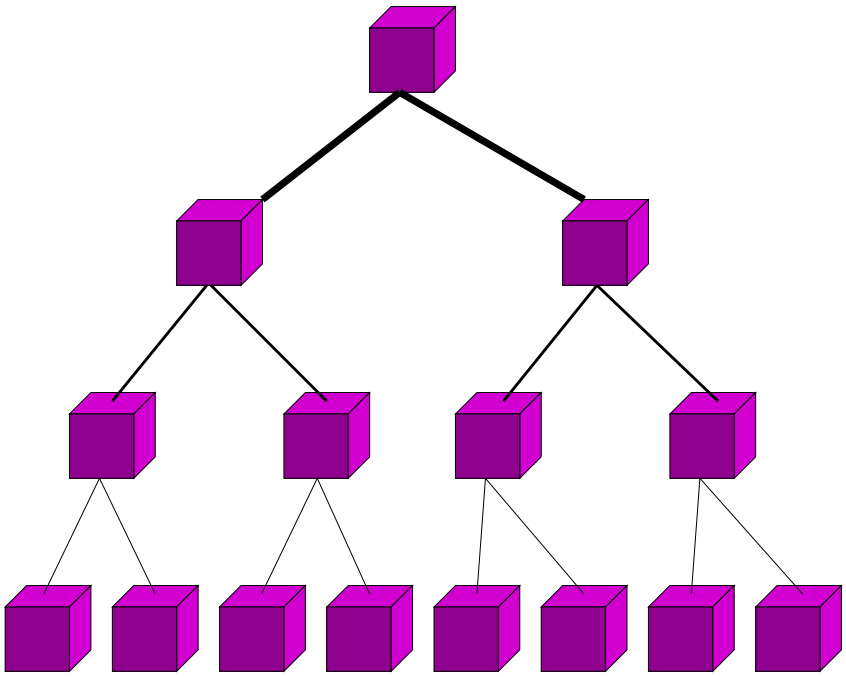

Figura 14: Red tipo árbol grueso

  - el número de entradas y salidas en una red de este tipo implementada en pastilla depende del número de patillas admisible por la tecnología.

- Si el número de puertos en N, el coste de un conmutador es $O(N^2)$.

Redes dinámicas: medio compartido

- Puede considerarse como un subtipo del conmutador, donde
  - hay posibilidad de interconectar cualquier entrada con cualquier salida, ...
  - ... pero no simultáneamente.

Redes dinámicas: redes multietapa

- Se utilizan varios conmutadores, típicamente de barras cruzadas, organizados en varias etapas.
- Un camino desde un nodo fuente a un destino atraviesa varias etapas, no necesariamente todas (por ejemplo, en una red irregular, o en redes bidireccionales).
- Atendiendo a la disponibilidad de caminos para establecer nuevas conexiones entre una entrada y una salida libres habiendo ya conexiones en curso, las redes multietapa se clasifican como
  - redes *bloqueantes*,
  - redes *configurables*,
  - redes *no bloqueantes*.

Redes multietapa bloqueantes

- No es posible siempre establecer una nueva conexión entre un par fuente/destino libres, debido a conflicto de recursos con las conexiones en curso.
- Generalmente solo existe un camino entre una entrada y una salida dadas.

Diseño de una red multietapa

1. Fijar el número de entradas y salidas, k^n. Ello implica
   - n etapas de conmutadores, $E_0, E_1, \ldots, E_{n-1}$.
   - $n \cdot k^{n-1}$ conmutadores $k \times k$.
2. Fijar la subred que interconecta las etapas, $S_1, S_2, \ldots, S_{n-1}$. Cada subred es una permutación de los índices de la salida a los índices de la entrada de la etapa siguiente.
3. Por fin, las subredes S_0, S_n, que interconectan las interfaces de red con los conmutadores.

Identificación de las entradas/salidas

- Puesto que tenemos k^n entradas y salidas, podemos identificarlas con un número que tenga n dígitos en base k. Por ejemplo, si $k = 2$, y $n = 3$, tenemos que las entradas/salidas serían: (000), (001), etc.
- Así podemos representar cada subred S_i como una permutación de los dígitos que identifican una salida con la siguiente entrada.
- Una permutación interesante es el baraje perfecto, B, (*perfect shuffle*), que se implementa así:

$$B[(d_{n-1}, d_{n-2}, \ldots, d_0)] = (d_{n-2}, d_{n-3}, \ldots, d_1, d_0, d_{n-1}),$$

  con $0 \leq d_i < k$, y $0 \leq i < n$.

Red Omega

- Se trata de una red $k^n \times k^n$.
- Las subredes $S_0, S_1, \ldots, S_{n-1}$ implementan una permutación baraje perfecto.
- También se puede usar el baraje perfecto inverso:

$$B^{-1}[(d_{n-1}, d_{n-2}, \ldots, d_0)] = (d_0, d_{n-1}, d_{n-2}, \ldots, d_2, d_1),$$

  con, igual que antes, $0 \leq d_i < k$, y $0 \leq i < n$.

Red Omega

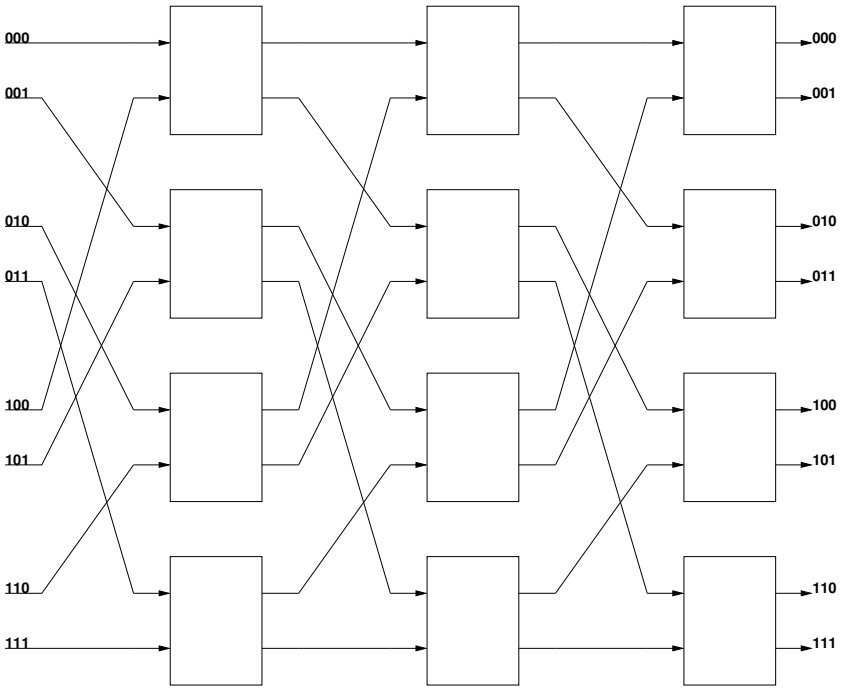

Figura 15: Red Omega

Red línea base

- También es una red $k^n \times k^n$.
- Las subredes aplican la siguiente permutación *línea base*, L_i:

$$L_i[(d_{n-1},d_{n-2},...,d_{i+1},d_i,d_{i-1},...,d_1,d_0)]=(d_{n-1},d_{n-2},...,d_{i+1},d_0,d_i,d_{i-1},...,d_1),$$

  con, igual que antes, $0 \leq d_i < k$, y $0 \leq i < n$.
- Cada subred S_j aplica la permutación L_{n-j}, con $1 \leq j \leq n-1$. La red S_0 aplica un baraje perfecto.

Red línea base

Red mariposa

- También es una red $k^n \times k^n$.
- Las subredes aplican la siguiente permutación *mariposa*, M_i:

$$M_i[(d_{n-1},d_{n-2},...,d_{i+1},d_i,d_{i-1},...,d_1,d_0)]=(d_{n-1},d_{n-2},...,d_{i+1},d_0,d_{i-1},...,d_1,d_i),$$

  con, igual que antes, $0 \leq d_i < k$, y $0 \leq i < n$.
- Cada subred S_j aplica la permutación M_j, con $1 \leq j \leq n-1$.

Red mariposa

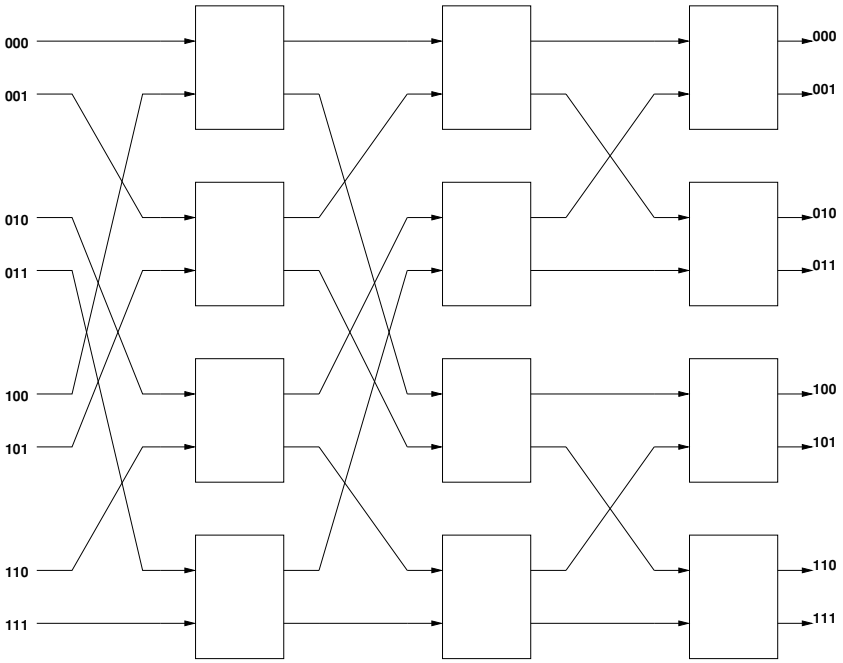

Figura 16: Red línea base

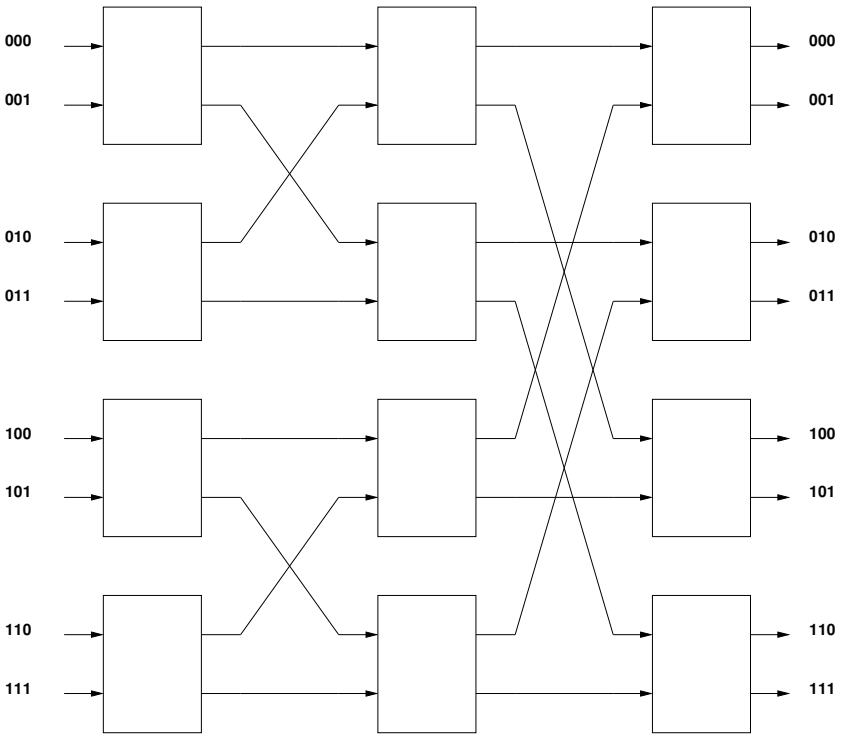

Figura 17: Red mariposa

Tabla para redes dinámicas bloqueantes $k^n \times k^n$, $N = k^n$

Parámetros	Red unidireccional	Red bidireccional
Grado	$k+k$	$k+k, 2k+2k$
Diámetro	$\log_k N + 1$	$2\log_k N$
Bisección	$N/2$	N
Nodos	$N/k\log_k N$	$N/k\log_k N$

▶ Nota: Parámetros válidos solo para una red tipo mariposa.

Redes multietapa configurables

- Siempre es posible encontrar un camino para conectar un par fuente/destino libres, aunque a veces es necesario reencaminar alguna de las conexiones en curso, es decir, reconfigurar la red.
- El coste de la reconfiguración es alto. Este tipo de redes se prefiere allí donde se sabe con antelación todas las conexiones que serán necesarias en la siguiente fase (computadores matriciales).

Ejemplo de red reconfigurable

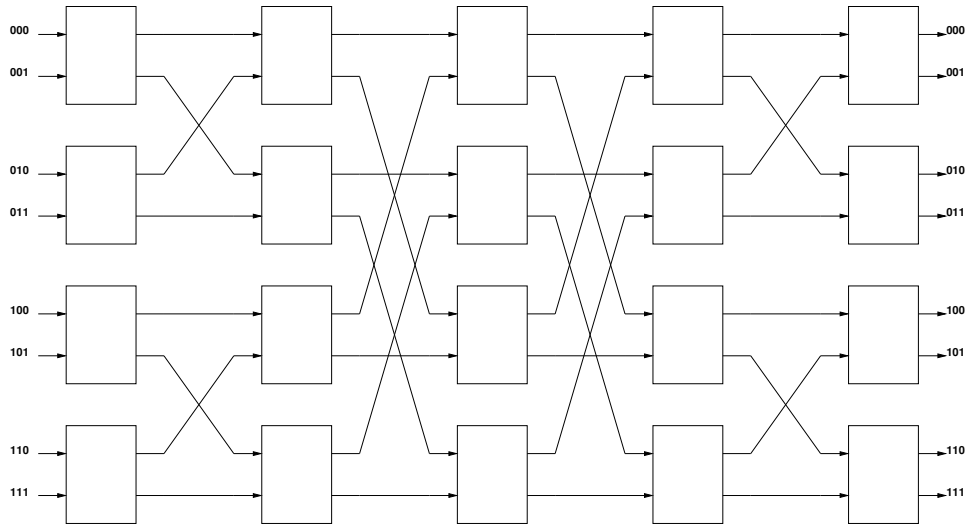

Figura 18: Red de Beneš (dos mariposas *back-to-back*).

Redes multietapa no bloqueantes

- Cualquier fuente se puede conectar con cualquier destino, con independencia de las conexiones en curso.
- Lógicamente, necesita varios caminos entre cada par fuente/destino, lo que conduce a un coste más alto.

Ejemplo de red no bloqueante

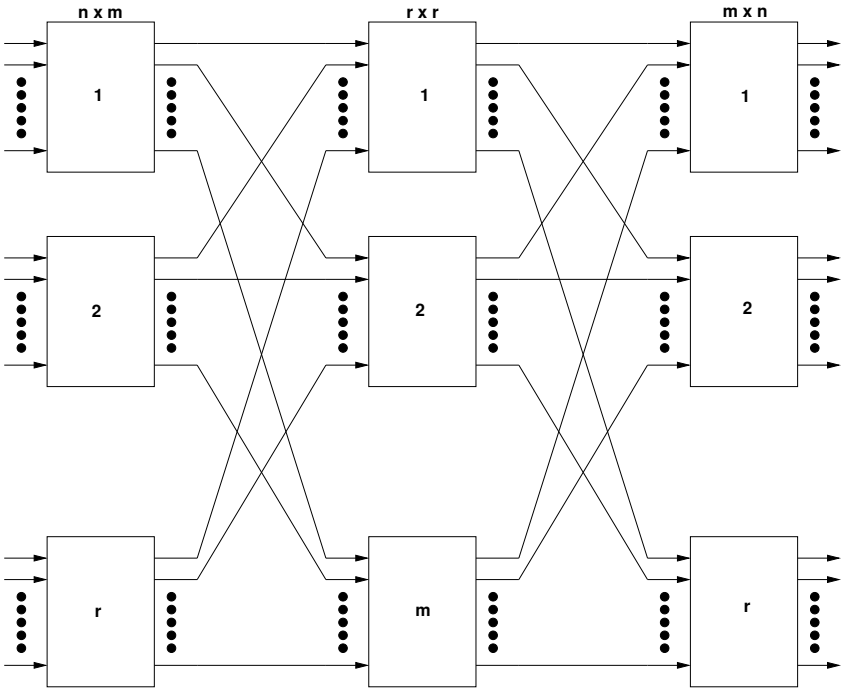

Figura 19: Red de Clos $n \cdot r \times n \cdot r$

4.3. Estrategias de conmutación

Modelos de sobrecarga (punto a punto)

- *Ancho de banda*: bytes por segundo que puede transmitir un enlace. Depende de la tecnología.
- *Retardo de propagación*: tiempo que tarda en llegar la señal de un extremo al otro. Depende del canal de comunicación.
- *Retardo de transmisión*: tiempo que tarda el transmisor en transmitir el mensaje completo. Depende del ancho de banda y del tamaño del mensaje.
- *Sobrecargas*: tiempos necesarios en el emisor y en el receptor para la puesta en marcha de la comunicación.

Modelos de sobrecarga (punto a punto)

- Podemos modelar lo anterior de la siguiente manera:

$$T(m) = O_{\text{send}} + T_{\text{prop}} + m/AB + O_{\text{rec}},$$

donde O_{send} y O_{rec} son las sobrecargas, T_{prop} es es tiempo de propagación, m es el tamaño del mensaje y AB el ancho de banda del canal.

Modelos de sobrecarga (punto a punto)

- Resumiendo lo dicho:

$$T(m) = t_0 + \frac{m}{r_\infty}$$

Aquí t_0 es el tiempo de arranque, y r_∞ es el ancho de banda asintótico.

Modelos de sobrecarga (punto a punto)

- Observemos que la velocidad, $AB(m)$, para transmitir un mensaje de tamaño m es

$$AB(m) = \frac{m}{T(m)} = \frac{mr_\infty}{m + t_0 r_\infty}$$

- Es claro que

$$\lim_{m\to\infty} AB(m) = r_\infty,$$

como era de esperar.

Modelos de sobrecarga (punto a punto)

- Según lo visto, el ancho de banda se degrada cuanto más pequeño es el mensaje que se quiere transmitir.
- Se utiliza, entonces, el valor $m_{\frac{1}{2}}$, longitud de semi-pico, como el tamaño del mensaje que proporciona un ancho de banda igual a la mitad del asintótico.
- Este valor mide la capacidad del sistema de comunicaciones para transmitir mensajes de tamaño pequeño.

Modelos de sobrecarga (punto a punto)

- De su definición, tenemos que la longitud de semi-pico cumple que

$$AB(m_{\frac{1}{2}}) = \frac{1}{2} r_\infty = \frac{m_{\frac{1}{2}} r_\infty}{m_{\frac{1}{2}} + t_0 r_\infty}$$

de donde

$$m_{\frac{1}{2}} = t_0 r_\infty.$$

- Si el ancho de banda asintótico es muy grande, entonces el tiempo de arranque, t_0, ha de ser muy pequeño.

Modelos de sobrecarga (comunicación colectiva)

- Para la comunicación colectiva:

$$T(m,n) = t_0(n) + \frac{m}{r_\infty(n)}$$

Aquí también t_0 es el tiempo de arranque, y r_∞ es el ancho de banda asintótico. La diferencia es que aparece una dependencia con respecto al número de nodos, n, involucrados.

Método *ping-pong*

```
for (i=0 ; i<pasos ; i++)
{
 if (my_id == 0)
 {
  tmp = time();
  start_time = time();
  // Enviar un mensaje de m bytes al nodo 1
  // Recibir un mensaje de m bytes del nodo 1
  end_time = time();
  timer_overhead = start_time - tmp;
  total_time = end_time - start_time - timer_overhead;
  comm_time = total_time/2;
 }
 else if (my_id == 1)
 {
  // Recibir un mensaje de m bytes del nodo 0
  // Enviar un mensaje de m bytes al nodo 0
 }
}
```

Técnicas de conmutación

- Determinan cómo los datos atraviesan el camino desde la fuente al destino.
- Las alternativas son
  - *conmutación de circuitos* (circuit switching),
  - *conmutación con almacenamiento y reenvío* (store & forward).
  - *conmutación recortada* (cut-through switching):
    - *conmutación vermiforme* (wormhole switching),
    - *conmutación virtual recortada* (virtual cut-through switching).

Conmutación de circuitos

Conmutación de circuitos

- Se reserva inicialmente el camino entre las interfaces que se usarán en una transmisión.
- Típicamente, la fuente envía un paquete-sonda con la información del destino, que irá reservando en el camino a lo largo de la ruta. Llegada a su destino, queda establecido el *circuito* fuente-destino.
- A continuación se transmite todo el mensaje. El pie del mensaje irá liberando el circuito conforme avance.
- El espacio para buffers se reduce al tamaño de un *flit*.

Temporización para conmutación circuital

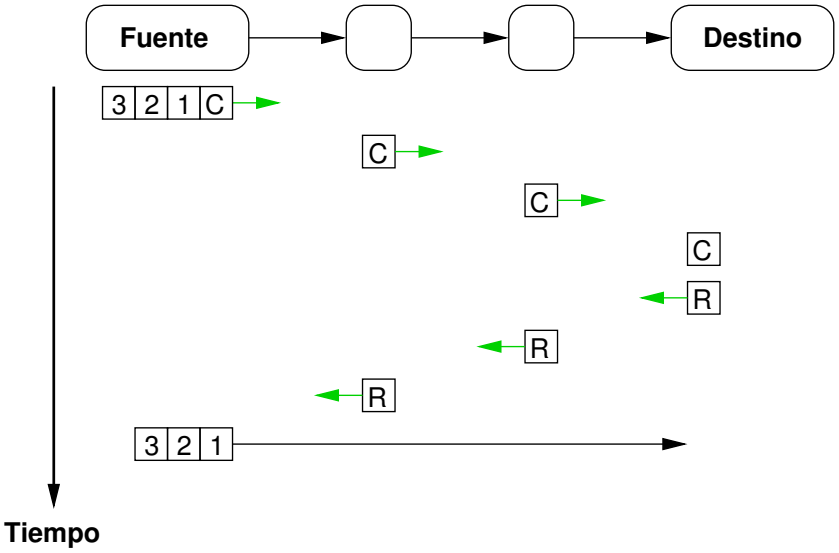

Figura 20: Conmutación de circuitos

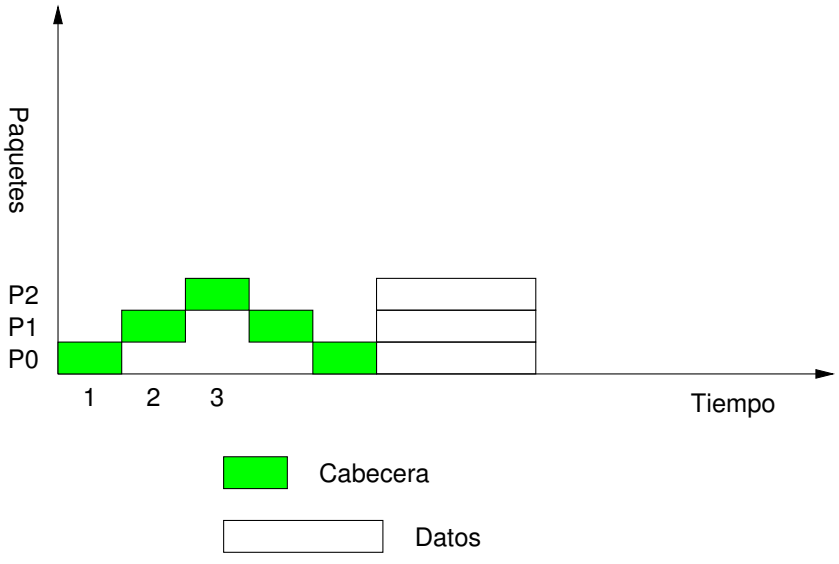

Figura 21: Temporización en conmutación de circuitos

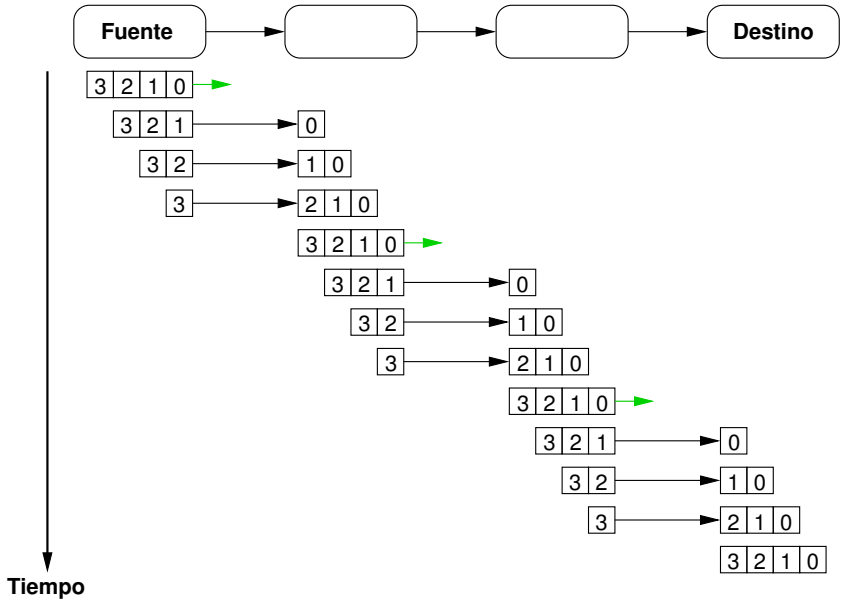

Figura 22: Conmutación almacenamiento y reenvío

Conmutación almacenamiento y reenvío

Almacenamiento y reenvío

- Un conmutador espera a recibir (almacenándolo) un paquete o unidad de transferencia entre interfaces en su totalidad antes de ejecutar el algoritmo de encaminamiento (flecha verde).
- Ejecutado el algoritmo, se envía el paquete a lo largo de los ciclos necesarios.
- En cada momento, el paquete está transfiriéndose a lo largo de un solo canal.
- La memoria en el conmutador receptor debe permitir almacenar todo el paquete.
- El tamaño de los paquetes vendrá limitado por la memoria de los conmutadores.

Temporización para *store & forward*

Modelo de comunicación para *store & forward*

- Nos basamos en el modelo básico para transmitir un mensaje:

$$T(m) = t_0 + \frac{m}{r_\infty}.$$

- Si llamamos n_{ij} al número de enlaces entre i y j,

$$T_{ij}(m) = n_{ij} \cdot \left(t_0 + \frac{m}{r_\infty} \right).$$

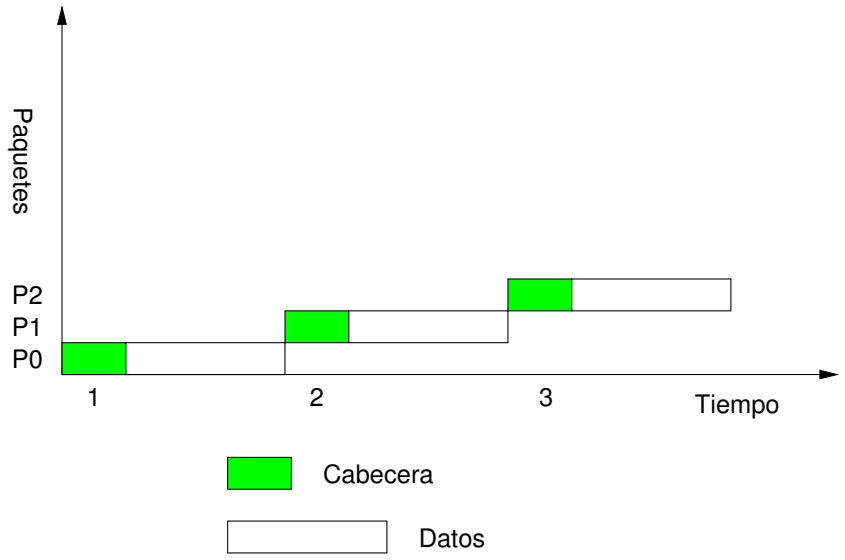

Figura 23: Temporización en conmutación «almacenamiento y reenvío»

Modelo de comunicación para *store & forward*

- Esto se podría mejorar si tratamos de fraccionar los mensajes en s paquetes y los enviamos en forma encauzada:
$$T_{ij}(m) = (n_{ij} + s - 1) \cdot \left(t_0 + \frac{m}{s r_\infty} \right).$$
- Si minimizamos $T_{ij}(m)$ como función de s, se encuentra que el valor óptimo para s es
$$s_{\text{opt}} = \sqrt{\frac{(n_{ij} - 1)\, m}{t_0 r_\infty}}.$$

Conmutación *cut-through*

- El algoritmo de encaminamiento se ejecuta tan pronto como se ha recibido la cabecera de la unidad de información a transmitir.
- Si se produce un bloqueo, se puede almacenar íntegramente en el conmutador.
- Los buffers de entrada del conmutador tienen espacio al menos para un paquete completo.
- Resulta atractivo cuando los mensajes son de tamaño pequeño.
- Se tienen dos variantes: conmutación vermiforme y conmutación recortada virtual.

Conmutación vermiforme

Conmutación vermiforme

- El algoritmo de encaminamiento se ejecuta tan pronto como se ha recibido la cabecera del paquete a transmitir.

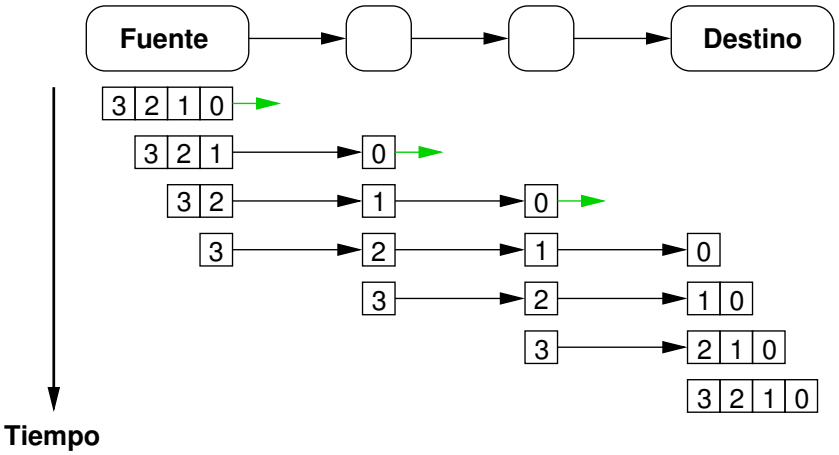

Figura 24: Conmutación vermiforme

- Esta cabecera suele ser un *flit*, que se enviará al siguiente conmutador, al tiempo que se van recibiendo los siguientes *flits* del paquete.
- La cabecera va abriendo camino que los demás siguen, hasta llegar al pie: es el movimiento de los vagones de un tren, o de un gusano.

Conmutación vermiforme

- Ventaja:
  - La necesidad de almacenamiento de los conmutadores decrece con esta técnica.
- Inconveniente:
  - En cada momento, distintos *flits* pueden estar transmitiéndose por distintos canales, incluso en paralelo. En un momento dado, un paquete puede estar ocupando varios canales (en el límite, todos), lo que implica mayor consumo de recursos.
  - Aumenta la posibilidad de errores.

Temporización para conmutación vermiforme

Modelo de comunicación para *cut-through*

- Para este tipo de conmutación se suele emplear este modelo

$$T_{ij}(m) = t_0 + n_{ij}\delta + \frac{m}{r_\infty}.$$

- En la expresión anterior, δ representa el tiempo necesario para ejecutar el algoritmo de encaminamiento.
- Usualmente $\delta \ll t_0$, lo que se traduce en que el tiempo de comunicación, en la práctica, es independiente del número de nodos atravesados.

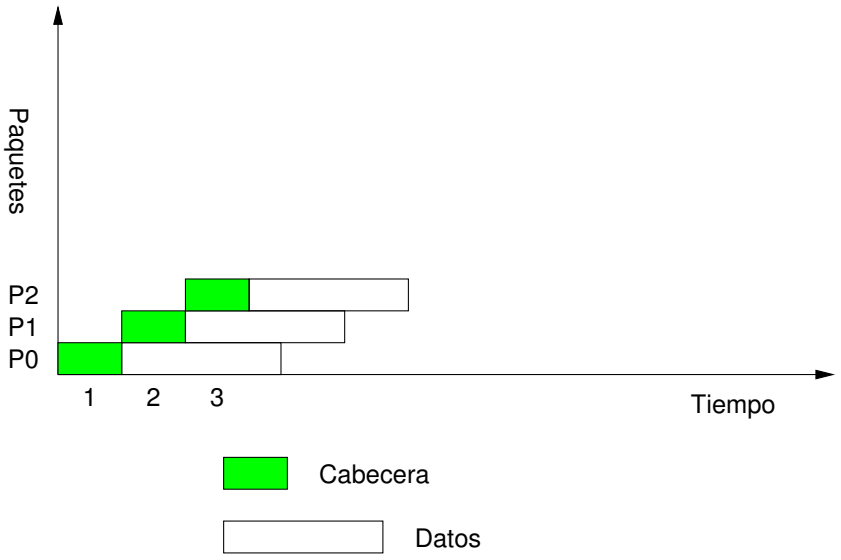

Figura 25: Temporización en conmutación vermiforme

4.4. Encaminamiento

Taxonomía de los algoritmos de encaminamiento

- Mínimo ⇔ no-mínimo.
- Determinista ⇔ adaptativo.
- Basado en fuente, en destino o circuital.

  ► Para el análisis de algoritmos, suponemos un modelo determinista y mínimo.

Encaminamiento en redes estáticas

- En redes ortogonales, el encaminamiento es simple y se basa en la diferencia de las coordenadas.
- El encaminamiento se puede implementar en hardware, con lo que la latencia de transmisión es menor.
- Además, el algoritmo es sencillo, con lo que el coste económico de los conmutadores también es menor.
- El encaminamiento es especialmente simple en redes hipercubo.

  ► ¡OJO! Podrían aparecer problemas de tipo *hot spot*.

Resumen final

- La evolución de hardware y software han convertido en borrosas las fronteras, antaño bien claras, entre los distintos tipos de arquitecturas.
  - Las operaciones de envío de mensajes son realizadas en máquinas de memoria compartida a través de *buffers* compartidos.

    - A su vez, en una arquitectura de paso de mensajes, la memoria compartida se puede emular mediante mensajes que transporten los punteros a las direcciones compartidas.

- En resumen: se ha convergido a una organización compuesta de una colección de máquinas completas a las que se añade una interconexión rápida y escalable.