ARQUITECTURAS PARALELAS
APUNTES COMPLETOS
Curso 24/25

Índice

Tema 1

1. Introducción

- Multiprocesadores: Sistemas formados por un cierto número de procesadores que trabajan en paralelo. Se comunican con variables en memoria compartida.

- Multicomputadores: Sistemas que trabajan en paralelo con memoria independiente. Emplean un sistema de memoria distribuida y la intercomunicación se realiza con mensajes entre ellos

2. Memoria Compartida

- **UMA:**
    - **Características:**
        - **Acceso uniforme:** El tiempo de acceso es independiente del procesador que realiza la solicitud o de la memoria que contiene los datos
        - **Memoria compartida:** Todos los procesadores acceden a la misma memoria
        - **Caché privada:** Cada procesador puede tener su propia caché
        - **Comparación de periféricos:** Los periféricos se comparten entre los procesadores

    - **Funcionamiento:**
        - Procesadores conectados con bus o conmutador crossbar
        - Cualquier procesador accede a cualquier posición de memoria
        - Comunicación entre procesos mediante variables
        - Modelo de programación load/store (similar a monoprocesador)
        - El sistema gestiona el movimiento de datos y la coherencia de la caché

Existen 3 tipos principales de arquitecturas UMA:

- Utilizando arquitecturas SMP basadas en buses
- Utilizando conmutadores crossbar
- Utilizando redes de conmutación multietapa

Aplicaciones:

Aplicaciones de propósito general, sistemas de tiempo compartido multiusuario y acelerar la ejecución de programas grandes en aplicaciones de tiempo crítico.

En **resumen**, la memoria compartida UMA proporciona acceso uniforme y transparente a la memoria para todos los procesadores, facilitando la programación paralela y mejorando el rendimiento de algunas aplicaciones.

- **NUMA:**
  - **Características:**
    - **Acceso no uniforme:** Se accede más rápido a la memoria local que a la de otro procesador.
    - **Escalabilidad:** Ofrece mayor escalabilidad que los sistemas basados en bus (mayor número de procesadores).
    - **Eficiencia:** Generalmente más económico y mejor rendimiento para muchos procesadores.

  - **Funcionamiento:**
    - Memoria dividida en bancos locales asociados a cada procesador
    - Cada procesador tiene acceso directo y rápido a la memoria local
    - Puede acceder a la memoria de otros proc. pero es más lento
    - Hardware o software adicional para mover datos entre bancos de memoria

  - **Aplicaciones:**
    - Beneficioso para servidores de alto rendimiento o para cargas de trabajo con datos fuertemente asociados a tareas o usuarios.

  - **Desventajas:**
    - Mayor latencia para acceder a la memoria no local
    - Complejidad de programación para optimizar el rendimiento

  - **Implementaciones:**
    - AMD implementó NUMA en Opteron (2003)
    - Intel NUMA servidores x86 (2007)

En resumen: **NUMA** ofrece mauor escalabilidad y rendimiento que UMA para sistemas con muchos procesadores a cambio de mayor complejidad en acceso a memoria y programación.

3. Problemas en sistemas multiprocesador

En estos sistemas cada procesador puede tener su propia caché. Eso puede causar inconsistencia si varios procesadores modifican el mismo dato en caché. La coherencia asegura que todos los procesadores tengan una vista consistente de los datos en memoria.

- Inconsistencia de datos: Cuando un procesador modifica un dato y otro lo lee sin estar actualizado
- Condiciones de carrera: Dos o más procesadores modifica un dato a la vez
- Escrituras no propagadas: Los cambios de una caché no se reflejan en las otras

4. Problemas de coherencia

Surgen cuando múltiples procesadores modifican el los mismos datos a la vez.

Hay 3 escenarios que generan estos problemas:

- Compartición de datos
- Migración de procesos
- Entrada/Salida

4.1.1 Problemas de coherencia por compartición de datos

La compartición de datos ocurre cuando varios procesadores acceden a la vez a una misma ubicación de memoria, ya sea para leer o escribir, lo cual es común en sistemas con memoria compartida.

Problema: Cuando un procesador modifica un datos en su caché local, otros procesadores que comparten ese dato pueden no ver los cambios al instante. Esto crea una inconsistencia en la imagen, ya que algunos procesadores podrían estar trabajando con una versión antigua del valor.

4.1.2 Problemas de coherencia por migración de procesos

Esto ocurre cuando un proceso en ejecución se mueve de un procesador a otro para mejorar la eficiencia o distribuir la carga. Esta migración puede generar problemas de coherencia en la caché.

Problema: Cuando se migra un proceso a otro procesador, es posible que el nuevo procesador no tenga una copia válida de los datos que el antiguo estaba usando en su caché anterior. Si el dato ha sido modificado pero no escrito en la memoria, el procesador puede estar trabajando con datos desactualizados.

4.1.3 Problemas de coherencia por entrada/salida

En estos sistemas los periféricos acceden directamente a la memoria principal, los problemas de coherencia aparecen cuando la E/S interactúa con la memoria sin que los cambios se propaguen a la caché de los procesadores.

Problema: Cuando un dispositivo E/S escribe directamente en la mem. principal, los procesadores pueden seguir trabajando con datos obsoletos almacenados en la caché.

4.1.4 Condiciones necesarias para garantizar la coherencia de caché

Para resolver los problemas mencionados se deben cuumplir 2 condiciones:

1. **Propagación de escritura:** Asegura que las operaciones de escritura realizadas por un procesador se reflejan en todas las cachés y en la memoria principal.
2. **Serialización de la escritura:** Todas las operaciones de escritura a una misma ubicación de memoria deben ser vistas por todos los procesadores en el mismo orden.

4.2 Protocolos de coherencia de caché

Existen 2 enfoques principales:

- Protocolos de observación del bus (snoopy):
    - Basados en invalidación: (al modificar un dato las otras caché lo invalidan)
        - Write-trough
        - Write-back
            - MSI
            - MESI
            - MOESI
            - MESIF
    - Basados en actualización: (el cambio se propaga a las otras cache)
        - Firefly
        - Dragon
- Protocolos basados en directorios:
    - De mapeo completo
    - Limitados
    - Encadenados

4.2.1 Write through

Este protocolo con invalidación garantiza que la memoria principal siempre tiene la versión más actualizada de los datos, pero puede generar un aumento de tráfico en el bus (menos eficiente en sistemas con muchas escrituras).

El estado **V** (Válido) indica que el dato en caché es actualizado, mientras que **I** (Inválido) señala que ya no es válido. En **Write-through**, las escrituras se reflejan en la caché y la memoria principal. Si otro procesador modifica el dato, las otras cachés invalidan sus copias.

4.2.2 Protocolo MSI

Tiene 3 estados principales

- M (Modified): Modificado solo en una caché
- S (Shared): Compartido en las cachés, coincide con la memoria
- I (Invalid): La línea de caché no es válida

4.2.3 Protocolo MESI

Añade el estado **E** (Exclusive) lo que optimiza el acceso. En este estado el dato solo se encuentra en una caché y es igual al de la memoria principal, lo que permite la escritura sin invalidaciones.
Reduce el tráfico del bus.

4.2.4 Protocolo DRAGON

El protocolo Dragon **actualiza** las copias de caché en lugar de invalidarlas cuando se escribe en un bloque compartido. El estado **SD** (Compartido-Sucio) indica que puede haber otras copias actualizadas, pero la memoria principal no lo está.

4.3. Conclusión

Los problemas de coherencia de caché en sistemas multiprocesador pueden causar inconsistencias si no se gestionan correctamente. La compartición de datos y la migración de procesos son fuentes comunes de estos problemas. Los protocolos como MSI, MESI y MOESI son esenciales para garantizar la coherencia y mejorar el rendimiento en arquitecturas paralelas.

5. Directorios

Almacenados en la memoria principal, permiten al controlador de memoria mantener la constancia de las copias en las cachés locales. A cada linea de la cache se le asocia una etiqueta.

Existen 3 posibilidades de implementación:

- Directorios de mapeo completo
- Directorios limitados
- Directorios encadenados

5.1 Directorios de mapeo completo

- La etiqueta incluye un campo de un bit por cada posible poseedor de copia
- Un campo adicional de un bit indica si hay perdido de escritura
- Antes de conceder permiso de escritura, se invalidan las copias existentes

Estos directorios tienen un problema de escalabilidad, ya que el tamaño de las etiquetas crece con el número de nodos de la máquina.

5.2 Directorios limitados

- No existe un campo para cada nodo, solo unos pocos
- Los cambios necesitan los bits suficientes como para saber a qué nodo codifican log2 (nº de nodos)
- Si se reduce mucho el nº de campos es posible minimizar el tamaño del directorio
- Si se limita el nº de poseedores de copia y todos los campos están en uso si aparece un nuevo peticionario hay que retirar alguna copia.

5.3 Directorios encadenados

- Sólo se conserva un campo en la etiqueta para apuntar el último poseedor de la copia.
- Cada poseedor dispone de un puntero al anterior de la lista

- Un nuevo peticionario se coloca al final de la lista y el controlador le otorga el puntero al que proceda
- Una petición de escritura se debe propagar por toda la lista, invalidando cada copia y avisando al anterior, cuando el primero confirma, se concede el permiso.

6. Ejemplo de bus: TLSB

- Implementado en máquinas Alpha como el Alphaserver 8400
- Segunda mitad de los 90s
- Bus síncrono con buses de datos y direcciones separados
- Bus de datos de 256 bits.
- BW máximo de 3,2 GB

6.1 TLSB: Direccionamiento

- Direccionamiento geográfico de los módulos mediante 3 líneas. Se pueden conectar hasta 9 módulos ya que la dirección 000 es ocupada por el dispositivo 0 y el 8 tiene que ser de e/s obligatoriamente.

- Direccionamiento virtual para dispositivos como bancos de memoria o CPUs. De esta manera adquieren una dirección propia en el sistema. Cada módulo puede albergar hasta 8 direcciones virtuales.

- Hasta 1TB de memoria es direccionado mediante direcciones de 40 bits.

- La dirección es decodificada por el solicitante para extraer la dirección virtual.

- Las transferencias emplean líneas de caché de 64 bytes

- El solicitante puede lanzar una petición al bus al mismo tiempo que compara la etiqueta de cache local. Si se produce un “hit” la petición se anula.

- De los 40 bits de direcciones, se toman los bits 6-39 como dirección de línea de 64 bytes. 234 líneas de cache x 234 bytes /Línea = 240 bytes direccionables.

- El bit 5 se emplea para definir la secuencia en que los dos bloques de 32 bytes aparecen en el bus: High>low o Low>high.

- Los 5 bits restantes codifican la dirección virtual (hasta 16 CPUs y hasta 16 bancos de memoria).

6.2 TLSB: Arbitraje

- Cualquier transacción en el bus debe iniciarse con una petición por parte del módulo

- El arbitraje es distribuido, lo cual implica que no existe un dispositivo que actúe de árbitro. La contienda tiene que ser resuelta entre los módulos.

- La prioridad de los nodos se inicializa a su dirección geográfica

- En tiempo de funcionamiento, se emplea un mecanismo de Round Robin para actualizar la prioridad.

6.2 TLSB: Transferencias

- Cuando un nodo esclavo está listo para proporcionar los datos que han sido requeridos toma control del bus.

- El esclavo activa la línea denominada TLSB_SEND_DATA, a lo que todos los dispositivos tienen tiempo para activar las líneas TLSB_SHARED o TLSB_DIRTY.

- TLSB_SHARED activada indica que algún dispositivo dispone de copia válida y desea seguir disponiendo de ella. Esta línea puede ser activada por cualquier dispositivo en respuesta a cualquier comando.

- TLSB_DIRTY activa indica que hay un dispositivo cuya copia es más reciente que la que hay en memoria. En tal caso, él se encargará de volcar los datos al bus.

- TLSB_DIRTY sólo puede ser activada en respuesta a comandos de lectura.

- Las especificaciones del protocolo no establecen el protocolo de coherencia, sino sólo el funcionamiento de las líneas.

Tema 2

1. Introducción

Este tema cubre las redes de interconexión usadas en sistemas paralelos y cómo los algoritmos de encaminamiento permiten que los datos se transmitan de manera eficiente. También máquinas de hardware débilmente acoplado y los diferentes niveles de acoplamiento en esta categoría en función del hardware compartido.

Al no disponer de memoria común, la comunicación se lleva a cabo por el paso de mensajes.

- Mensaje: Unidad de intercambio de información
- Paquete: Unidad de conmutación

La conmutación define cómo se mueven los datos en redes de interconexión y afecta la latencia y ancho de banda. Existen varias técnicas (almacenamiento y envío, corte directo, gusano), y su elección depende de la topología y las necesidades del sistema paralelo. Junto con el encaminamiento, influye en la eficiencia de transmisión de mensajes, siendo clave para optimizar la comunicación y el rendimiento en arquitecturas paralelas.

2. Máquinas de hardware débilmente acoplado

Las máquinas de hardware débilmente acoplado son sistemas computacionales que tienen las siguientes características:

- **Conexión**: Están conectadas mediante una red de conmutación, generalmente con una velocidad de transmisión de datos relativamente baja y un retraso alto.
- **Memoria**: Cada máquina tiene su propia memoria local y no comparte memoria directamente con otros procesadores
- **Independencia**: Los procesadores y unidades de memoria son independientes entré sí, lo que permite que cada máquina trabaje en tareas separadas
- **Comunicación**: La interacción entre las máquinas se realiza por el paso de mensajes por la red.
- **Distribución**: Suelen formar parte de sistemas distribuidos donde los recursos y el procesamiento se reparten en diferentes nodos
- **Flexibilidad**: Permiten una mayor flexibilidad en la configuración y escalabilidad añadiendo o quitando nodos
- **Aplicaciones**: Son comunes en redes LAN y en sistemas distribuidos donde cada usuario tiene su propia estación de trabajo

3. ¿Qué es una red de interconexión?

Una **red de interconexión** es un sistema de enlaces y nodos que permite la comunicación de un sistema paralelo, como procesadores, memorias y dispositivos de e/s.
Estas redes facilitan el proceso de intercambiar datos de forma rápida y eficiente.

3.1 Rol de las redes de interconexión en ArPa

En las arquitecturas paralelas, se trabaja con múltiples procesadores o núcleos a la vez. La eficiencia con la que estos elementos se comunican entre sí y con la memoria es clave para lograr un alto rendimiento. Las redes de interconexión son el “puente” que conecta los componentes.

- **Escalabilidad**: La red de interconexión debe ser capaz de soportar mas nodos sin afectar al rendimiento.

- **Flexibilidad**: La red tiene que ser flexible como para manejar varios patrones de datos.

- **Minimizar la latencia**: La latencia es el tiempo que tarda el mensaje de un nodo a otro. Las redes deben estar diseñadas para minimizar la latencia.

3.2 Componentes de una red de interconexión

Una red se compone de nodos y enlaces:

- **Nodos**: Puntos de la red que procesan o almacenan datos (procesadores, memoria, dispositivos e/s),

- **Enlaces**: Son las conexiones entre los nodos por los que se transfieren los datos. Pueden ser físicos (cables, buses) o enlaces virtuales (canales en redes interas de un chip).

3.3 Cómo funciona una red de interconexión

Cuando un nodo necesita comunicarse con otro la red de interconexión se encarga de transmitir la solicitud de datos y devolver la respuesta, este proceso involucra:

- **Encaminamiento**: La red selecciona la mejor ruta para los datos, existen diferentes algoritmos de encaminamiento:
    - **Deterministas** (siempre la misma ruta para los mismos nodos)
    - **Adaptativos** (Cambian la ruta según las condiciones de la red)

- **Transmisión de datos**: Los datos se dividen en unidades más pequeñas llamadas flits que viajan por la red hasta el destino. (Procesos de control ayudan a evitar atascos *Wormhole routing*)

- **Sincronización y arbitraje**: La red debe sincronizar las comunicaciones para evitar conflictos. Los mecanismos de arbitraje gestionan los conflictos y deciden que nodo tiene prioridad para usar el enlace

3.4 ¿Por qué son importantes las redes de interconexión?

En una ArPa, el rendimiento depende tanto de la velocidad de los procesadores como de la eficiencia de la red. Si la red está mal diseñada o es lenta reducirá la velocidad global del sistema.

3.5 Resumen

Una red de interconexión es el sistema que permite que los nodos se comuniquen entre sí. Sus componentes principales son los nodos y los enlaces. Los algoritmos de encaminamiento y de control aseguran que los datos viajan de manera eficiente y sin problemas.

4. Conmutación

4.1 ¿Qué es la conmutación?

Se refiere a la forma en la que se envían datos a través de la red y cómo se manejan los nodos intermedios entre el origen y el destino. Los mensajes grandes se dividen en fragmentos que se transmiten por la red.
La conmutación es el proceso que decide cómo estos fragmentos se dirigen, almacenan y transmiten.

- **Conmutación de circuitos**: Se establece una conexión permanente para transmitir un mensaje completo
- **Conmutación de paquetes (control de flujo)**:
    - Sin división: T=(N-1)(L/W) store & forward (S&F)
    - Con división en flits: T=(L/W) + (F/W)(N-2) wormhole
    - Con división en phits: T = L/W + (P/W)(N-2) virtual cut through (VCT)

  Paquetes: disponen de información de destino y nº de orden dentro del mensaje.

  Flits (flow control unit): unidades en las que se divide un paquete, no disponen de ninguna información salvo el de cabecera, por lo que todos deben seguirle.

  Phits: la cantidad de bits transmitidos por ciclo de reloj, unidad mínima de información posible; equivale al nº de líneas de datos del enlace.

  El pipeline en VCT es análogo a Wormhole pero requiere espacio de almacenamiento intermedio para paquetes completos, como S&F.

4.2 Estrategias de conmutación

Conmutación de paquetes (control de flujo):

4.2.1 Conmutación de almacenamiento y envío (Store and Forward - S&F):

Funcionamiento: En este método cada nodo intermedio de la red recibe un mensaje completo antes de enviarlo al siguiente nodo.

Ventajas: Simple y garantiza que cada fragmento de datos llegue completo ants de ser enviado (minimiza errores).

Desventajas: Introduce una latencia importante ya que cada nodo debe esperar a recibir el mensaje completo. (largos tiempos de transmisión, no eficiente para redes a gran escala)

4.2.2 Conmutación de gusano (Wormhole Switching):

Funcionamiento: Los mensajes se dividen en fragmentos llamados *flits*. El primer fragmento (cabeza de gusano) lidera el camino y reserva una ruta para el resto.

Ventajas: Muy eficiente en el uso de los buffers de los nodos ya que solo almacena los fragmentos (baja latencia y mejor uso de recursos de red).

Desventajas: Puede causar problemas como el bloque de cabeza de línea (*head-of-line blocking*). Si la cabeza se bloquea el resto de fragmentos también, lo que puede producir cuellos de botella.

4.2.3 Conmutación por corte directo (Virtual Cut Through - VCT):

Funcionamiento: El nodo intermedio empieza a reenviar datos tan pronto como recibe la primera parte del mensaje (no espera a todo el mensaje).

Ventajas: Reduce la latencia y aumenta la velocidad (útil para sistemas donde la velocidad es crucial).

Desventajas: Más susceptible a errores, ya que el mensaje puede ser reenviado antes de ser completamente recibido (si hay problemas de red son más difíciles de corregir).

4.3 Conmutación de paquetes

Aspectos relevantes:

- **Control de flujo**: cómo y cuándo se mueven los paquetes por la red. Incluye:
  - Resolución de colisiones
  - Encaminamiento
- **Bloqueo (deadlock)**: Los paquetes entran en bucle y la red colapsa
- **Livelock**: Un paquete se mueve por la red sin llegar a ningún destino
- **Inanición (starvation)**: Un paquete no recibe un servicio y se queda anclado en un elemento de almacenamiento

4.3.1 Problemas de redes: Deadlock e Inanición

- Deadlock (bloqueo mortal): Ocurre cuando un conjunto de nodos queda atrapado esperando recursos indefinidamente

- Inanición: Ocurre cuando un nodo nunca obtiene recurso debido a otros nodos que siempre tienen prioridad

4.3.2 Soluciones:

4.3.2.1 Soluciones al bloqueo

- **Canales virtuales para demorar la aparición del bloqueo**

    - **Descripción**: Permiten que múltiples flujos de datos compartan un mismo enlace físico. Cuando un flujo de datos bloquea un canal los otros usan el resto de canales disponibles (reduce la probabilidad de *head-of-line blocking*).

    - **Ventaja**: Mejora el uso del enlace y previene que un maquete bloquee todo el tráfico del enlace.

      La técnica de c**anales virtuales multiplexados en el tiempo (TDM)** optimiza el uso del ancho de banda en enlaces de red. **No se clasifica como solución a bloqueos, colisiones o encaminamiento,** ya que n**o aborda directamente estos problemas**, sino que mejora la eficiencia en el uso de los enlaces de red.

- **Ciertas combinaciones de interconexión:**

    - **Descripción**: Algunas combinaciones de topologías y algoritmos garantizan que no se produzca bloqueo. Por ejemplo el uso del algoritmo DOR (Dimension Ordenar Rounting) en red de malla.

    - **Ventaja**: Las combinaciones optimizadas evitan la necesidad de mecanismo complejos de control de flujo, garantizando la llegada de los paquetes a destino.

- **Recuperación mediante descarte y reintento:**

    - **Descripción**: Si el bloqueo ocurre, el sistema puede detectar el ciclo de espera y tomar medidas de recuperación. Se puede descartar el paquete bloqueado y notificar para que se vuelva a enviar.

    - **Ventaja**: Resuelve el bloque al romper el ciclo de espera y reiniciar la transmisión del paquete. Es útil en sistemas donde es crucial evitar la pérdida de datos.

4.3.2.2 Resolución de colisiones

Una colisión ocurre cuando dos o más paquetes intentan usar el mismo recurso al mismo tiempo. Existen varias técnicas para resolver y gestionar esto:

- **Descartar**
    - **Descripción**: Cuando hay una colisión, uno de los paquetes involucrados se descarta. El emisor recibe una notificación para que pueda enviar el paquete más tarde

    - **Ventaja**: Es una solución rápida y simple. No necesita almacenamiento adicional.

    - **Desventaja**: Puede aumentar la latencia y el tráfico en la red y puede provocar retrasos en la entrega

- **Almacenar**
    - **Descripción**: En lugar de descartar el paquete, se almacena en un buffer en el nodo donde hubo la colisión hasta que pueda ser transmitido.

    - **Ventaja**: Evita que los paquetes se pierdan, manteniéndolos en la red sin tener que enviarlos. Reduce el tráfico adicional y no aumenta la latencia.

    - **Desventaja**: Consume recursos de memoria en los nodos intermedios, lo que se puede limitar. Si hay muchas colusiones los buffers pueden llenarse y provocar un bloqueo de cabeza de línea o saturación en la red.

- **Reenviar** (Causa de livelock)
    - **Descripción**: Algunos sistemas optan por reenviar el paquete sin descartarlo ni almacenarlo. Se vuelve a enviar desde el origen hasta el nodo de la colisión.

    - **Ventaja**: Los datos no se pierden y la red sigue intentando entregar los paquetes. Puede reducir el tiempo total de transmisión si las colisiones son escasas.

    - **Desventaja**: Puede llevar a un **livelock**, donde un paquete nunca llega al destino pero está en la red. si hay demasiadas colisiones puede afectar al rendimiento global de la red.

- **Bloquear:**
    - **Descripción**: Cuando ocurre una colisión el paquete se bloquea en el nodo donde colisionó, espera hasta que el enlace esté disponible para continuar.

    - **Ventaja**: Minimiza el uso de los buffer y el tráfico innecesario. Asegura que los paquetes lleguen a destino.

    - **Desventaja**: Puede provocar cuellos de botella en la red, lo que puede aumentar la latencia y generar situaciones de deadlock.

Conclusiones:

- **Descartar**: Simple pero puede aumentar la latencia
- **Almacenar**: Útil cuando hay memoria suficiente (puede generar congestión)
- **Reenviar**: Mantiene los paquetes en movimiento pero puede causar livelock
- **Bloquear**: Evita tráfico innecesario pero puede causar cuellos de botella

La elección depende del tamaño de la red, el tráfico esperado y la disponibilidad de recursos. Estas soluciones se suelen combinar para maximizar la eficiencia.

4.3.2.3 Encaminamiento

El **encaminamiento** es el proceso para decidir **por qué ruta deben viajar los paquetes** a través de la red. Los **algoritmos de encaminamiento** son los que determinan la ruta.

La solución de **conmutación de paquetes** basada en caminamiento se centra en elegir rutas eficientes y en prevenir los bloqueos o situaciones que pueden empeorar el rendimiento de la red.

Algoritmos de encaminamiento:

Criterios de clasificación:

- **Número de destino**
  - **Unicast**: Un único destino
  - **Multicast**: Múltiples destinos

- **Lugar donde se toman las decisiones:**
  - **Centralizados**: Tomadas por un controlador central
  - **Fuente**: Tomadas en el nodo de origen
  - **Distribuidos**: Tomadas de manera distribuida

- **Forma de implementación:**
  - **Tabla**: Basado en tablas de enrutamiento predefinidas (Simple y rápido pero puede ser ineficiente).
  - **Máquina de estados**: Maquina de estados finitos (flexibles y adaptables para algoritmos adaptativos)

- **Adaptatividad:**
  - **Deterministas**: Siempre la misma ruta para un mismo origen y destino
  - **Adaptativos**: Ajustan la ruta según las condiciones de la red

- **Nº de caminos alternativos:**
  - **Parcialmente adaptativos**: Cierta flexibilidad en la elección de rutas
  - **Totalmente adaptativos**: Cualquier ruta disponible en la red → + livelock

- **Progresividad:**
  - **Progresivos**: Siempre encaminan los paquetes hacia destino (- latencia)
  - **Regresivos**: Los paquetes pueden retroceder

- **Minimalidad:**
  - **Mínimos**: Siempre la ruta más corta posible (- latencia, + congestión)
  - **No mínimos**: Rutas más largas para evitar congestión o bloqueos (+ livelock)

Resumen: Los **algoritmos de encaminamiento** en redes de conmutación de paquetes se clasifican según varios criterios, como el número de destinos, el lugar donde se toman las decisiones, su implementación, adaptabilidad, el número de caminos, y si son mínimos o no mínimos. La elección del algoritmo depende de los requisitos de la red y el equilibrio entre simplicidad, eficiencia y flexibilidad.

Tema 3 - Redes de interconexión

Las redes de interconexión son un componente fundamental en los sistemas de computación paralela y distribuida. Su función principal es permitir la comunicación eficiente entre los nodos del sistema.

Se pueden clasificar principalmente en 2 grupos:

1. **Redes estáticas o directas**
    - Topología fija y enlaces permanentes entre los nodos
    - Ejemplos: Malla, toros, hipercubos…
    - Ventajas: Latencia predecible y escalabilidad
    - Desventajas: Menor flexibilidad, posible subutilización de enlaces.

2. **Redes dinámicas o indirectas**
    - Enlaces bajo demanda de los elementos de conmutación
    - Ejemplos: Redes multietapa, crossbar
    - Ventajas: Mayor flexibilidad y mejor uso de recursos
    - Desventajas: Posible contención y latencia variable

La elección entre los dos modelos depende de factores como el número de nodos, patrones de comunicación esperados, coste, escalabilidad y rendimiento requerido.

3.1 - Conceptos básicos para comprender las topologías de red

1. **Número de nodos:** Representa el número total de elementos de procesamiento en la red. Afecta a la escalabilidad y la complejidad del sistema.

2. **Grado:** Número de enlaces conectados a cada nodo. Un grado mayor puede mejorar el rendimiento, pero aumenta la complejidad y el coste.

3. **Diámetro:** La distancia máxima entre cualquier par de nodos de la red. Un diámetro menor significa generalmente menor latencia.

4. **Ancho de bisección:** El ancho de banda mínimo entre dos mitades iguales de la red. Es un indicador para manejar patrones de tráfico adversos.

5. **Coste:** El número total de enlaces de la red. Medida aproximada de la complejidad y el coste de implementación de la red.

Estos parámetros están relacionados entre ellos, por ejemplo, reducir el diámetro suele aumentar el grado o el coste.

Información adicional:

- **Métricas avanzadas:** Existen métricas más sofisticadas como la distancia media, la congestión de enlace o la probabilidad de bloqueo.

- **Impacto en aplicaciones:** Distintas aplicaciones pueden beneficiarse de mejorar algunos parámetros. Por ejemplo, las aplicaciones con comunicaciones frecuentes se benefician de un diámetro bajo.

- **Simulación y modelado:** Existen herramientas de software para modelar y simular redes, permitiendo evaluar los parámetros antes de la implementación física.

3.1.1 - Coste del mensaje

El coste de total de transmisión de un mensaje en una red depende de varios factores:

1. **Tiempo de procesamiento:**
    - Nodo fuente: Tiempo para preparar y enviar el mensaje
    - Nodo destino: Tiempo para recibir y procesar el mensaje
    - Depende de la arquitectura del nodo y la complejidad del protocolo de comunicación.

2. **Retardo de propagación:**
    - Tiempo que tarda la cabecera del mensaje en recorrer la distancia entre nodos.
    - Depende de la distancia física y medio de transmisión.

3. **Tiempo de transmisión:**
    - Tiempo necesario para insertar todos los bits en la red.
    - Inversamente proporcional al ancho de banda del canal.

4. **Tiempo de almacenamiento:**
    - Tiempo que el mensaje pasa en buffers intermedios. (sin canalización)
    - Puede ser significativo en redes con conmutación de paquetes store and forward.
5. **Grado de solapamiento:**
    - En redes que usan técnicas de canalización, indica cuando se solapan las transmisiones de los fits del mensaje.
    - Puede reducir la latencia total

6. **Tiempo de contención:**
    - Retrasos debidos a la competencia por recursos de la red (enlaces,, buffers).
    - Altamente variable y difícil de predecir
7. **Latencia:**
    - Tiempo total desde el envío hasta la recepción

- Suma de todos los componentes anteriores

8. **Throughput:**
   - Tasa a la que la red puede procesar mensajes
   - Se mide en mensajes por unidad de tiempo o bits por segundo.

- **Modelos anaíticos**
  Existen modelos matemáticos para estimar los tiempos en diferentes topologías y condiciones de carga.

- **Técnicas de reducción de latencia**
  Incluye el uso de algoritmos de encaminamiento adaptativos, técnicas de conmutación avanzadas y optimizaciones a nivel protocolo.

- **Impacto del software**
  Las capas de software pueden añadir overhead significativo, especialmente en mensajes pequeños.

- **Medición en sistemas reales**
  Se usan benchmarks especializados y herramientas de perfilado para medir los tiempos en implementaciones reales.

3.2 - Topologías estáticas

Las topologías estáticas o redes directas se caracterizan por tener enlaces fijos entre los nodos. Algunas de las más comunes son:

- **Array lineal**
  - Estructura más simple, nodos conectados en una línea
  - Grado = 2 (excepto extremos)
  - Diámetro = N-1 (N= número de nodos)
  - Ancho de bisección = 1
  - Ventajas: Simplicidad, fácil de implementar
  - Desventajas: Alto diámetro, baja tolerancia a fallos

- **Anillo**
  - Nodos conectados en un círculo cerrado
  - Grado = 2
  - Diámetro = N/2
  - Ancho de bisección = 2
  - Ventajas: Simple, todos los nodos con el mismo grado
  - Desventajas: Diámetro relativamente alto

- **Anillo cordal de grado 4**
  - Anillo con conexiones adicionales que “saltan” nodos.
  - Grado > 2 (depende del numero de cuerdas)
  - Diámetro y ancho de bisección dependen de la configuración

    - Ventajas: Menor diámetro que el anillo simple
    - Desventajas: Mayor complejidad de enrutamiento

- **Barrell Shifter**
    - Topologías que permiten rotaciones eficientes de datos
        - De 16 nodos
            - Número de nodos: $N = 2^n$
            - Grado: G = 2n -1
            - Diámetro: D = 2
            - Ancho de bisección: B = N/2 + N/4 - 2

    - Ventajas: Eficiente para operaciones de rotación de bits
    - Desventajas: Complejo de implementar para grandes N

- **Árbol binario**
    - Estructura jerárquica con un nodo raíz y cada nodo con dos hijos
    - Número de nodos: $N = 2^n - 1$
    - Grado: G = 3 (excepto hojas y raíz)
    - Diámetro: $D = 2^n - 1$
    - Ancho de bisección: B = 1
    - Ventajas: Eficiente para algunos algoritmos jerárquicos
    - Desventajas: Congestión en la raíz, ancho de bisección bajo

- **Árbol Grueso**
    - Variante del anterior con más conexiones entre niveles
    - Grado: $G = 2^n - 1$ en el nivel más bajo
    - Ancho de bisección: B = G/2
    - Ventajas: Mejor ancho de bisección que el binario
    - Desventajas: Mayor complejidad de implementación

- **Aplicaciones:**
  El Barrel Shifter es común en unidades aritméticas de procesadores, Los árboles son útiles en divide y vencerás.

- **Variantes:**
  Existen variantes como árboles k-aros o árboles gordos (fat trees) que buscan mejorar ciertas características de rendimiento.

- **Enrutamiento:**
  Los algoritmos de enrutamiento en estas tecnologías suelen ser más simples que en redes más complejas como hipercubos.

- **Malla**

    - Estructura regular con conexiones en 2 o más dimensiones
    - Ventajas: Escalabilidad y simplicidad de enrutamiento
    - Desventajas: Diámetro relativamente alto para redes grandes
    - $N = d^n$
    - $D = n(d-1)$
    - $B = d^{n-1}$
    - $C = n * d^{n-1}(d-1)$
    - $G = 2n$

- **Toro**
    - Malla con conexiones adicionales que unen los bordes
    - Ventajas: Menor diámetro que la malla y simetría
    - Desventajas: Mayor complejidad de calbeado
    - $N = d^n$
    - $D = n(d/2)$
    - $B = 2d^{n-1}$
    - $C = nd^n$
    - $G = 2n$

- **Malla Illiac**
    - Variante de toro con conexiones “torcidas” en los bordes
    - Propiedades similares a toro
    - Ventajas: Puede reducir la longitud de los cables en algunas implementaciones
    - Desventajas: Enrutamiento más complejo que en toros regulares

- **Implementaciones reales:**
  Muchos supercomputadores han usado topologías de malla o toro (IBM Blue Gene/L → toro 3D).

- **Enrutamiento adaptativo:**
  Estas topologías permiten algoritmos de enrutamiento adaptativos que pueden mejorar el rendimiento y la tolerancia a fallos.

- **Mapeo de aplicaciones:**
  El mapeo eficiente de tareas de aplicaciones a nodos físicos es crucial para aprovechar estas topologías.

- **Hipercubo**
    - Cada nodo se conecta a “n” otros nodos en un espacio n dimensional
    - Número de nodos: $N = 2^n$
    - Grado: G = n (da igual el número de conexiones)
    - Diámetro: D = n
    - Ancho de bisección: B = N/2
    - Ventajas: Diámetro logarítmico, alto ancho de bisección, simetría.
    - Desventajas: El grado aumenta con el tamaño de la red, limitando la escalabilidad.
- **Ciclos conectados en cubo**

    - Combinación de hipercubo con anillos en cada dimensión
    - Número de nodos: $N = n * 2^n$
    - Grado: G = 3 (constante)
    - Diámetro: D = 2n
    - Ancho de bisección: $B = 2^{(n-1)}$
    - Ventajas: Grado constate, buena escalabilidad
    - Desventajas: Mayor diámetro que el anterior

- **Aplicaciones históricas:**
  El hipercubo fue popular en supercomputadores de los 80 y 90
- **Propiedades matemáticas:**
  El hipercubo tiene propiedades interesantes como la facilidad de particionarlo en subcubos
- **Enrutamiento:**
  En hipercubos el enrutamiento puede basarse en operaciones de bit sobre las direcciones de nodos
- **Variantes:**
  Existen topologías derivadas como el hipercubo plegado o el cubo retorcido que mejoran ciertas propiedades

3.3 - Topologías dinámicas

Se caracterizan por **establecer conexiones bajo demanda**.

Bloqueantes /No bloqueantes: Una vez establecida una configuración, en las bloqueantes las posibilidades de conexión pueden quedar bloqueadas.

Monoetapa/Multietapa: En función del número de etapas de conmutación que deben atravesar para llegar a destino.

- **Bus**
    - Medio compartido que conecta todos los nodos
    - Ventajas: Simple, bajo coste para sistemas pequeños
    - Desventajas: Baja escalabilidad, potencial cuello de botella
    - Uso: En sistemas pequeños o como parte de jerarquías de interconexión.

- **Crossbar**
    - Permite conectar N entradas a M salidas de forma no bloqueante
    - Estructura: Matriz de puntos de cruce controlados individualmente.
    - Ventajas: Alto rendimiento, no bloqueante
    - Desventajas: Coste cuadrático con el número de puertos, complejidad de control.

- **Crossbar: Punto de conexión**
    - Arbitraje: En buses y crossbars, el arbitraje es crucial para gestionar el acceso al medio compartido

- Implementaciones modernas: Los crossbar son comunes en switches de alto rendimiento y en Network-on-Chip

- Técnicas de optimización: Existen variantes como corssbars dispersos o jerárquicos para reducir la complejidad en sistemas grandes.

- **Red Omega**
  - Es una red multietapa, específicamente una red de Benes
  - Estructura: Compuesta por etapas de conmutadores 2x2 conectados en un patrón específico.
  - Número de etapas: log2(N), donde N es el número de entradas/salidas.
  - Ventajas: Menor complejidad que un corssbar completo (O(N log N) vs $O(N^2)$), capacidad de enrutar cualquier permutación de entradas a salidas.
  - Desventajas: Puede haber bloqueo interno, mayor latencia que un crossbar directo.

- **Red Omega: Rutado:**
  - Se basa en los bits de la dirección de destino.
  - Cada etapa usa un bit de la dirección para determinar la configuración del concepto binario.
  - Aplicaciones: Las redes Omega y sus variantes se han usado en supercomputadores y en el diseño de routers de alta velocidad
  - Comparación con otras redes: Las redes Omega ofrecen un compromiso entre el rendimiento de un corssbar y la escalabilidad de las topologías más simples
  - Variantes: Existen otras redes multietapa similares como la red Butterfly o la red Bayan

- **Mariposa**
  - Similar a la red Omega, topología ligeramente diferente.
  - Estructura:
    - $N = 2^n$ entradas y salidas
    - n + 1 etapas
    - 2^n conmutadores por etapa

  - Propiedades:
    - Diámetro: log2(N) + 1
    - No es recursivamente expansible
    - Proporciona un único camino entre cada par entrada-salida
- **Mariposa: Rutado**
  - Ventajas:
    - Latencia predecible
    - Simplicidad de enrutamiento
    - Paralelismo

- Desventajas:
  - Bloqueos
  - Falta de caminos alternativos
- Comparación con Red Omega: Tiene una estructura más regular que puede simplificar su implementación
- Variantes: Existen versiones extendidas como la red Bennes que no es bloqueante
- Aplicaciones: Se usan en computación paralela y en algoritmos FFT (Transformada Rápida de Fourier)
- Ofrece equilibrio entre rendimiento y complejidad.
- Ideal para implementaciones a muy gran escala. Su latencia baja la hace adecuada para aplicaciones que requieren comunicaciones rápidas.

- **Árbol grueso conmutado:**
  - La red se organiza en niveles o capas.
  - Los nodos están compuestos por switches en los niveles superiores o nodos de computación en los inferiores.
  - Las conexiones entre niveles aumentan su número a medida que se sube.
  - Características:
    - Ancho de banda constante
    - No bloqueante
    - Escalabilidad
    - Redundancia

  - Estructura en niveles:
    1. Nivel de core (Núcleo) Switches de alto rendimiento en la parte superior
    2. Nivel de agregación: Switches que conectan directamente el core con los switches de acceso
    3. Nivel de acceso: Switches que conectan con los servidores o nodos de computación
    4. Nivel de servidores: Nodos de computación final

  - Enrutamiento: Usa enrutamiento en tablas o adaptativo. Hay varias rutas para llegar a destino y se distribuye en tráfico uniformemente
  - Ancho de Banda: El número de enlaces aumenta si se sube en la jerarquía.
  - Ventajas:
    - Alto rendimiento
    - Baja latencia
    - Tolerancia a fallos
    - Escalabilidad
    - Costo y efectividad

  - Desventajas:
    - Complejidad de gestión
    - Costo inicial
    - Consumo de energía

- Aplicaciones:
  - Centro de datos
  - Supercomputadores
  - Clusters de computación

- **Hipercubo conmutado**
  - Enrutamiento:
    - Enrutamiento por Dimensión Ordenada
    - Enrutamiento Adaptativo

  - Ventajas:
    - Baja latencia
    - Alta conectividad
    - Simetría
    - Tolerancia a Fallos
    - Escalabilidad

3.4 - Encaminadores modernos

Son los dispositivos responsables de dirigir el tráfico de datos de manera eficiente. Su diseño se basa en 3 elementos principales:

1. **Colas de almacenamiento**
   Son estructuras de memoria que gestionan los paquetes de datos en diferentes etapas de su tránsito por el encaminador. Hay 3 tipos:

   a. **Colas de Inyección**
      - Función: Permiten al nodo colocar paquetes en la red.
      - Características:
        - Actúan como buffer entre el procesador local y la red.
        - Gestionan la congestión local antes de introducir paquetes en la red.

   b. **Colas de Tránsito**
      - Función: Almacenan los paquetes que no tienen origen ni destino en el nodo local.
      - Características:
        - Manejan el tráfico que pasa por el nodo.
        - Crucial para prevenir la congestión.

   c. **Colas de consumo**
      - Función: Entregan paquetes al nodo local.
      - Características:
        - Actúan como buffer antes de la entrega al procesador local.
        - Gestionan la recepción de paquetes cuando el nodo está ocupado.

2. **Interconexión (Crossbar Limitado)**
    - **Función:** Conecta físicamente las colas y los puertos del encaminador.
    - **Características:**
        - Permite varias conexiones simultáneas entre entradas y salidas.
        - "Limitado" → No todas las conexiones son posibles a la vez.
        - Optimiza el rendimiento al permitir transferencias paralelas.

3. **Árbitro**
    - **Función:** Toma decisiones sobre encaminamiento y resolución de conflictos.
    - **Características:**
        - Usa algoritmos de enrutamiento para determinar la mejor ruta.
        - Resuelve conflictos cuando múltiples paquetes compiten por un mismo recurso.
        - Gestiona la prioridad y la calidad de servicio de los paquetes.

Funcionamiento integrado:
1.Los paquetes llegan a las colas de inyección o tránsito.
2.El árbitro determina la ruta y prioridad de cada paquete.
3.El crossbar establece las conexiones según las decisiones del árbitro
4.Los paquetes se mueven por el crossbar hacia las colas de tránsito o consumo.
5.Los paquetes destinados al nodo local se entregan desde las colas de consumo.

Tecnologías y tendencias avanzadas
- **SDN (Software-Defined Networking)**
- **Virtualización**
- **Análisis de tráfico en tiempo real**
- **Integración con IA**

3.4.1 - Encaminadores modernos - Colas

Un **encaminador moderno** está diseñado para gestionar el flujo de datos en redes usando una estructura modular y simétrica que facilita la **escalabilidad y el mantenimiento**. Cada lado (norte, sur, este y oeste) incluye una **lógica de canal**, compuesta por:

1. **Receptor de canal**: recibe paquetes entrantes.
2. **Emisor de canal**: envía paquetes salientes.
3. **Demultiplexor**: separa paquetes según su destino.
4. **Multiplexor**: combina paquetes salientes.

Características clave:

- **Flujo de datos direccional**: Representado con flechas, se gestiona de manera eficiente gracias al diseño modular, procesando paquetes de cualquier dirección hacia su destino correcto.
- **Espacio de almacenamiento**: Controlado por mecanismos como *store & forward*, *wormhole*, y *virtual cut-through*, típicamente con colas FIFO. Estas pueden enfrentar problemas como el **bloqueo de cabeza de línea (HLB)**, donde un paquete bloquea a los siguientes en espera.

3.4.2 - Encaminadores modernos - Arbitraje

- **Selección**: Determina el puerto de salida adecuado para un paquete según el protocolo de encaminamiento. No aplica si el encaminamiento es estático.
- **Arbitraje**: Decide qué cola de entrada obtiene acceso a un puerto de salida cuando varias compiten por él.
- **Objetivo**: Maximizar la eficiencia y evitar problemas críticos como la inanición.

3.4.3 - Encaminadores modernos - Políticas de selección

- **Aleatoria:** Se asigna una salida al azar entre las posibles.
- **Cola más corta:** Se elige el puerto de salida con más espacio en la cola.
- **Smart:** Se realizan intentos sucesivos. Lógico si el encaminamiento es adaptativo. Primero se intenta continuar por el canal de entrada, si no está disponible, se cambia de dirección.

3.4.3 - Encaminadores modernos - Políticas de arbitraje

- **Aleatoria:** Se elige uno de los competidores al azar.
- **RoundRobin:** Se sigue una lista ordenada para dar servicio a todos.
- **Más antiguo:** Se da servicio al que más tiempo lleva en espera.
- **Cola más larga:** Se da servicio al canal con la cola más larga.

Tema 3 - Infiniband

Es una arquitectura de interconexión a nivel de sistema. Desarrollada por un consorcio de empresas líderes en tecnología, Infiniband se diseñó para abordar las demandas de ancho de banda y baja latencia en entornos de computación de alto nivel, centros de datos y aplicaciones empresariales críticas.

Objetivos:

- Comunicación rápida y eficiente entre servidores, almacenamientos y otros sistemas informáticos
- Ofrecer un alto ancho de banda y baja latencia para aplicaciones que requieren un rendimiento excepcional
- Facilitar la creación de redes de área sistema escalables y de alto rendimiento

3.5 - Conceptos

En la arquitectura IFB se integran diferentes dispositivos:

- Nodos de computación
- Periféricos
- Nodos de entrada/salida
- Switches
- Routers

Se trata de una red conmutada multietapa que adquiere una topología similar a la malla.

Características principales de IFB:

1. **Baja latencia:**
   Para lograr esto se emplean varios mecanismos:
   - **RDMA (Remote Direct Memory Access):** Permite que un dispositivo acceda a la memoria de otro sin involucrar los SO o otros nodos (reduce sobrecarga y latencia).
   - **Offloading del protocolo:** Gran parte de los protocolos de comunicación se implementan en el hardware de red. (Reduce carga CPU y menor tiempo de procesamiento).
   - **Conmutación cut-through:** Los switches usan cut-through en lugar de store-and-forward, lo que permite reenviar un paquete nada más ser recibido.

2. **Alto ancho de banda:**
   Para lograr esto se combinan dos factores clave:
   - **Altas tasas de transferencia:** IFB ha evolucionado aumentando la tasa de transferencia.
   - **Enlaces de múltiples líneas:** IFB permite agregar varias líneas en un solo enlace. (1x, 4x, 8x, 12x), multiplicando el ancho de banda del enlace.

3. **Arquitectura de red:**
   IFB usa una topología similar a una malla, esta arquitectura permite:
   - **Escalabilidad:** Se pueden agregar fácilmente más nodos y switches.
   - **Redundancia:** Varias rutas entre nodos mejoran la tolerancia a fallos.
   - **Balanceo de carga:** El tráfico se puede distribuir entre varias rutas.

3.6 - Componentes principales

- **Nodos de computación:** Realizan el procesamiento principal. Pueden ser servidores individuales o nodos de un cluster.
- **Nodos de entrada/salida (E/S):** Proporcionan conectividad a dispositivos de almacenamiento, redes externas u otros dispositivos.
- **Adaptadores de Canal (HCA - Host Channel adapter):** Interfaces de red de IFB, manejan la complejidad del protocolo liberando la CPU de esta tarea.
- **Switches:** Enrutan los paquetes entre los nodos. De 3 a 255 puertos.
- **Routers:** Permiten la comunicación entre subredes y otro tipo de redes.
- **Gestor de Subred (Subnet Manager):** Configura y gestiona los dispositivos de la subred. Se ejecuta en el nodo de computación, aunque también puede hacerlo en switch dedicado para redes más grandes.

3.7 - Direccionamiento en Infiniband

Infiniband utiliza un **direccionamiento jerárquico** que facilita la comunicación eficiente en subredes locales (intra-subred) y entre diferentes subredes (inter-subred).

Direccionamiento Local (Intra-subred):

- Cada puerto tiene un **Local Identifier (LID)** único, asignado por el Gestor de Subred (16 bits, hasta 65,536 direcciones).
- Rango de direcciones:
  1. *0000h*: Reservada
  2. *0001h - BFFFh*: Unicast
  3. *C000h - FFFEh*: Multicast
  4. *FFFFh*: Propósito especial
- **Proceso de enrutamiento local**:
  1. El paquete incluye DLID (destino) y SLID (origen).
  2. En los switches, el DLID determina el puerto de salida mediante una tabla de enrutamiento.
  3. El paquete sigue el proceso hasta el destino.

Direccionamiento Global (Inter-subred):

- Las direcciones globales tienen 128 bits (compatibles con IPv6), divididos en:
  1. *Prefijo de subred* (64 bits altos).
  2. *GUID* (64 bits bajos).
- **Proceso de enrutamiento global**:

1. Los paquetes incluyen un *Local Routing Header (LRH)* y un *Global Routing Header (GRH)* con identificadores de origen y destino.
2. El paquete se enruta localmente hasta un router de salida, que utiliza el GRH para identificar la subred destino.
3. El router decide si el destino está directamente conectado o necesita un nuevo salto.
4. Una vez en la subred destino, se usa direccionamiento local para alcanzar el puerto final.

Este esquema permite escalar desde pequeños clusters a redes globales de centros de datos.

Tema 4 - Programación de Arquitecturas Paralelas

La programación de arquitecturas paralelas es un campo fundamental en la informática. Este tema se estructura en 3 niveles de aproximación: Nivel de SO, nivel de aplicación y nivel de gestión.
Cada uno de estos niveles juega un papel crucial en el uso de los recursos de hardware paralelo y en la optimización de rendimiento de las aplicaciones paralelas.

4.1 - Nivel de sistema operativo

Es la base sobre la que se construye la infraestructura de la programación paralela. Podemos distinguir 3 tipos de sistemas operativos para manejar arquitecturas paralelas:

4.1.1 - Sistemas Operativos de Red

Representan la forma más básica de soporte para la computación distribuida.

Características principales:

- Permiten realizar operaciones en máquinas a través de un adaptador de red.
- Proporciona una capa de abstracción para el acceso a los recursos distribuidos.
- Casi todos los SO modernos incluyen esta funcionalidad por defecto.

Limitaciones:

- Cierto grado de computación paralela → Poco eficiente
- No están optimizados para alto rendimiento.

Ejemplos:

- Versiones de Windows NT con servicios de red.
- Distribuciones de Linux con servicios Network File System y rsh/ssh

4.1.2 - Sistemas Operativos Multiprocesador

1. **Características principales:**
    - Gestionan entornos de multiprogramación sobre hardware multiprocesador.
    - Optimizan el uso de múltiples procesadores o núcleos.
    - Proporcionan abstracciones para la programación paralela a través de sistema.

2. **Conceptos clave**
    - Proceso: Unidad de protección y asignación de recursos
    - Hilo: Unidad de planificación

    Un proceso puede contener múltiples hilos y cada hilo tiene su propia pila, estado de ejecución y contexto del procesador.

3. **Tipos de multiprocesadores:**
    - **Simétricos (SMP):**
        - Todos los procesadores son iguales y pueden ejecutar cualquier tarea.
        - El sistema operativo equilibra la carga.
    - **Asimétricos (Maestro-Esclavo):**
        - Un procesador maestro controla y distribuye las tareas a los esclavos.
        - Más eficiente para cargas específicas.

Planificación en sistemas multiprocesador:

1. **Decisiones clave:**
    - Asignar tiempo de CPU a procesos.
    - Determinar qué CPU ejecuta cada proceso.
2. **Modelos de planificación:**
    - **T**iempo compartido: Hilos encolados y planificados independientemente.
    - Espacio compartido: Hilos relacionados se ejecutan juntos y sin interrupciones.
    - Gang scheduling: Hilos relacionados son interrumpidos y retomados de forma conjunta.

Ejemplo: Soporte de Linux para multiprocesadores:

- Desde el kernel 2.6, Linux soporta multiprocesadores con un planificador de tiempo compartido.
- Mantiene colas de procesos activos y expirados por CPU.
- Ofrece 140 niveles de prioridad (100 para tiempo real).
- Rebalancea la carga cada 200ms y optimiza la eficiencia de la caché con colas específicas por CPU.

4.1.2 - Sistemas Operativos Multiprocesador

Los **sistemas operativos distribuidos** permiten que un conjunto de computadores independientes funcionen como una **máquina virtual unificada** para el usuario.

Implementaciones:

1. **Arquitectura cliente-servidor:** Los clientes acceden a servidores remotos (ej. sistemas de información).
2. **Llamadas a procedimiento remoto (RPC):** Ejecutan procedimientos en máquinas remotas.
3. **Sistemas de paso de mensajes:** Procesos en distintas máquinas se comunican mediante mensajes, usados en computación paralela de alto rendimiento.

Middleware:

Una capa adicional que facilita la funcionalidad de máquina virtual, basada en:

- **Migración de procesos:** Permite mover procesos entre máquinas en tiempo de ejecución.
- **Tolerancia a fallos:** Maneja la incorporación o desaparición de componentes sin interrupciones.

Características del paso de mensajes:

1. **Fiabilidad:**
    - Fiable: Garantiza la entrega del mensaje.
    - No fiable: No asegura la entrega.
2. **Bloqueo:**
    - Bloqueante: Detiene al proceso hasta completar la operación.
    - No bloqueante: Permite continuar el procesamiento.
3. **Sincronización:**
    - Síncrono: Emisor espera hasta que el receptor procese el mensaje.
    - Asíncrono: Depende del bloqueo.

Estrategias de middleware:

- **Extensión del sistema operativo:** Funcionalidad integrada para sistemas distribuidos.
- **Capa intermedia independiente:** API para aplicaciones (ej. MPI).

4.2 - Nivel de Aplicación

Se refiere a cómo los desarrolladores pueden aprovechar las capacidades de las arquitecturas paralelas en sus programas.

- **Paradigmas de desarrollo de aplicaciones paralelas:**
    - **Paralelismo implícito:**
        - El usuario no se involucra en el uso del hardware paralelo.
        - Confía en que las capas inferiores optimicen automáticamente.
        - **Ventajas:** Simplicidad para el programador.
        - **Desventajas:** Puede no aprovechar el potencial del hardware.

    - **Paralelismo explícito**
        - El usuario desarrolla sus aplicaciones para explotar el hardware paralelo.
        - Requiere más esfuerzo y conocimiento por parte del programador.
        - Puede lograr mejor rendimiento.

4.2.1 - Modelos de paralelismo explícito

- **Lenguajes tradicionales con librerías paralelas (ej. MPI)**
    - **Ventajas:** Solución simple en cuanto a esfuerzo inicial.
    - **Desventajas:** Alta implicación del usuario. Riesgo de pérdida de rendimiento.

- **Extensiones a lenguajes tradicionales**
    - Requieren nuevos compiladores
    - Implican un esfuerzo de aprendizaje adicional

- **Directivas de compilación**
    - El paralelismo se especifica mediante directivas
    - Si se ignoran las directivas el lenguaje se compila como secuencial.

4.2.2 - Programación concurrente

El programador se encarga explícitamente de la creación, destrucción y sincronización de hilos, procesos o tareas.

Servicios necesarios en un entorno de programación concurrente:

- Expresión de la ejecución concurrente (procesos, hilos, tareas)
- Comunicación entre procesos
- Sincronización entre procesos

Tipos de procesos concurrentes:

- Independientes: No necesitan ni comunicación ni sincronización.
- Cooperativos: Se comunican y sincronizan para hacer una tarea común.
- Competitivos: Independientes, pero compiten por recursos comunes.

Métodos de comunicación:

4.2.2.1 - Sincronización en variable compartida

Objetivos: Evitar condiciones de carrera en el acceso a secciones críticas.
Sección crítica: Zona de código donde se modifica una variable compartida.

Mecanismos de sincronización:

1. **Espera ocupada**
    - El proceso comprueba constantemente si se puede entrar en la sección crítica.
    - Ventajas: Simple de implementar
    - Desventajas: Ineficiente para muchos procesos, consume recursos.

2. **Semáforos**
    - Variables enteras no negativas son manejadas mediante operaciones atómicas (wait y signal)
    - Los procesos en espera son encolados y suspendidos
    - Ventajas: Eficiente, permite la suspensión de procesos
    - Desventajas: Propenso a errores sin un buen uso

3. **Regiones críticas condicionales**
    - Zonas del codigo que solo se ejecutan en exclusión mutua
    - Los procesos en espera despiertan periódicamente para comprobar la condición de acceso.

4. **Monitores**
    - Módulos que encapsulan las regiones críticas de un programa.
    - Proporcionan exclusión mutua automática para los métodos monitor

5. **Objetos protegidos**
    - Herramientas proporcionadas por lenguajes específicos para facilitar la sincronización.

6. **Métodos sincronizados**
    - Mecanismo proporcionado por lenguajes orientados a objetos para sincronizaciones a nivel de método

4.2.2.2 - Sincronización en paso de mensajes

Existen 3 tipos de sistemas de paso de mensajes:

- Sistemas síncronos
- Sistemas asíncronos
- Invocación remota: el emisor continúa cuando recibe una respuesta del receptor.

4.2.2 - Gestión de procesos concurrentes

Aspectos a considerar:

1. **Estructura**
    - Número fijo de procesos
    - Número variable en tiempo de ejecución

2. **Nivel**
    - Modelo plano
    - Procesos anidados

3. **Granularidad**
    - **Grano grueso:** Pocos procesos → tareas complejas
    - **Grano fino:** Muchos procesos → tareas simples

4. **Inicialización**
    - Información pasada en el arranque
    - Información pasada en tiempo de ejecución

5. **Finalización**
    - Finalización del cuerpo de ejecución
    - Suicidio (autofinalización)
    - Abortro por otro proceso
    - Situación de errores sin tratar
    - Nunca (aplicaciones embebidas)
    - Cuando no son necesarios

6. **Explicitación de la creación de procesos**
    - **Fork/join:** Creación dinámica e inicialización mediante paso de parámetros
    - **Cobegin:** Ejecución concurrente de una secuencia
    - **Declaración explícita:** Estructuras de lenguaje para crear otros procesos

7. **Motivación para la programación concurrente**
    - Modelo ajustado a la realidad: El mundo es concurrente
    - Incremento de rendimiento en sistemas paralelos
    - Mejora del rendimiento incluso en sistemas no paralelos

4.3 - Nivel de Gestión: Planificación (Scheduling)

Es crucial para optimizar el rendimiento global de sistemas paralelos y distribuidos. Este nivel se encarga de decir cómo y cuándo se ejecutan los trabajos en el sistema.

1. **Complejidad de la planificación**
   No existe un algoritmo eficiente conocido para encontrar una solución óptima, por eso se usan heurísticas y aproximaciones.

   Factores que influyen en la complejidad:
    - Heterogeneidad de los recursos
    - Variabilidad en los requisitos de los trabajos
    - Dinámicas cambiantes del sistema

2. **Conceptos clave**
    a. **Partición:** Conjunto de recursos asignados a un trabajo (Puede ser estática o dinámica).
    b. **Precedencia:** Capacidad de interrumpir otros trabajos para atender a otros más prioritarios (mayor flexibilidad pero puede producir overhead).

3. **Estrategias de asignación de recursos**
    a. **Estática:**
        - Fija: Número de procesadores determinado por el administrador.
        - Variable: Basada en la petición de recursos del usuario .
        - Adaptativa: Determinada por el planificador al iniciar el trabajo.

b. **Dinámica**
    - Permite más cambios en la asignación de recursos en la ejecución.
    - Más difícil de implementar, pero más eficiente.

4. **Flexibilidad de los trabajos**
   La flexibilidad es crucial para una planificación eficiente.
   - **Trabajos rígidos:** Número fijo de procesadores.
   - **Trabajos moldeables:** El planificador determina el número de procesadores
   - **Trabajos evolutivos:** Requerimientos cambiantes en diferentes fases
   - **Trabajos maleables:** Número de procesadores modificable dinámicamente

5. **Políticas y objetivos de planificación**
   Los objetivos pueden cambiar según las necesidades del sistema:
   - Minimizar el tiempo de espera en cola
   - Minimizar el tiempo de ejecución total
   - Maximizar el throughput del sistema
   - Maximizar la utilización de recursos

   Técnicas avanzadas
   - **Backfilling:** Usa recursos reservados para ejecutar trabajos pequeños.
   - **Co-scheduling:** Coordina la ejecución de trabajos relacionados.

6. **Sistemas de gestión de carga (PBS):**
   PBS (Portable Batch System) es un ejemplo de sistema de gestión de carga usado en computación de alto rendimiento.

   Componentes principales:
   - Servidor de trabajos: Gestiona las colas y el estado de los trabajos.
   - Ejecutor de trabajos: Pone los trabajos en ejecución en los nodos de computo.
   - Planificador: Implementa políticas de selección de trabajo y asignación de recursos.

   Ventajas de PBS:
   - Gestión eficiente de recursos para grandes clusters
   - Interfaz unificada para usuarios y administradores
   - Implementación de políticas complejas de planificación

Conclusión:
La **programación de arquitecturas paralelas** abarca tres niveles interconectados:

1. **Sistema operativo:** Gestiona los recursos hardware y ofrece abstracciones para la computación paralela.
2. **Aplicación:** Permite desarrollar software eficiente y escalable.
3. **Gestión:** Optimiza la utilización global del sistema para múltiples usuarios y aplicaciones.

Dominar estos niveles es clave para diseñar y gestionar sistemas paralelos eficientes, especialmente en un contexto de hardware cada vez más paralelo y heterogéneo.

4.4 - PBS (Portable Batch System)

Es un sistema de gestión de cargas de trabajo usado en entornos de computación de alto rendimiento. Desarrollado originalmente para gestionar los recursos de computación de la NASA, se ha convertido en una herramienta estándar en centros de supercomputación y clusters.

4.4.1 - Introducción a PBS

PBS proporciona una interfaz para que los usuarios envíen, monitoreen y controlen los trabajos, mientras que ofrece a los administradores herramientas para gestionar los recursos de computación.

Componentes principales:

a) Gestión de colas
b) Planificación
c) Monitorización → Arquitectura de PBS

4.4.2 - Arquitectura de PBS

Está diseñada para ser modular y flexible, permitiendo una gestión eficiente de recursos. Los componentes principales son:

1. **Comandos**
    - De usuario: Permiten enviar, monitorizar y controlar trabajos
    - De administrador: Utilizados para configurar y gestionar el sistema
    - De operador: Para realizar tareas de mantenimiento y control del sistema

2. **Servidor de trabajos**
    - Núcleo del sistema PBS.
    - Gestiona las colas y el estado de los trabajos
    - Mantiene la base de datos de trabajo y recursos
    - Interactúa con el planificador para tomar decisiones sobre la ejecución de trabajos.

3. **Ejecutor de trabajos (MOM - Machine Oriented Miniserver):**
    - Se ejecuta en cada nodo de cómputo
    - Responsable de la ejecución local de los trabajos
    - Monitorea el uso de recursos y reporta al servidor.
    - Gestiona la E/S de los trabajos

4. **Planificador:**
    - Implementa las políticas de selección de trabajos y asignación de recursos.
    - Puede ser personalizado según las necesidades específicas del centro de cómputo.
    - Comunica sus decisiones al servidor de trabajos.

4.4.3 - Terminología clave en PBS

- **Nodo:** Sistema de computación con un único SO y espacio de memoria virtual unificado.
- **Nodo (nodo virtual):** Unidad básica de recurso computacional. Puede ser un núcleo CPU o procesador completo.
- **Host:** Máquina física que puede contener varios nodos o vnodos
- **Chunk:** Conjunto de recursos asignados a un trabajo. (CPUs, memoria…)
- **Cola:** Contenedor lógico de trabajos. Dos tipos principales
    - De rutado: Para mover trabajos a otras colas
    - De ejecución: Trabajos para ser ejecutados
- **Walltime:** Tiempo máximo de ejecución para un trabajo.

4.4.4 - Operativa de envío de trabajos

El proceso típico de envío y ejecución es el siguiente:

1. El usuario se conecta al servidor PBS (generalmente de forma remota).
2. El usuario prepara un script con el trabajo a realizar. Incluye:
    - Especificación de shell (opcional)
    - Directivas PBS: Para solicitar recursos o especificar atributos
    - Tareas: Programas o comandos a ejecutar
3. El usuario envía el trabajo usando el comando qsub
4. PBS asigna un identificador único al trabajo y lo coloca en cola
5. El planificador evalúa los trabajos en cola y decide cuáles pueden ejecutarse.
6. Cuando se ejecuta un trabajo se asignan recursos necesarios y se inicia en los nodos seleccionados
7. El usuario puede monitorear el progreso del trabajo (qstat)
8. Cuando se completa, los resultados se devuelven al usuario

4.4.5 - Ventajas de usar PBS

1. **Gestión eficiente de recursos:** Optimiza el uso de CPU, memoria y otros recursos.
2. **Flexibilidad:** Permite la implementación de políticas de planificación complejas y personalizadas.
3. **Escalabilidad:** Puede manejar desde pequeños clusters hasta supercomputadoras
4. **Interfaz unificada:** Experiencia consistente para usuarios y administradores
5. **Control granular:** Control detallado sobre la asignación de recursos y prioridades.
6. **Monitorización y contabilidad:** Herramientas para el seguimiento del uso de recursos y generación de informes.

4.4.6 - Desafíos y consideraciones en el uso de PBS

1. **Complejidad de configuración:** La configuración puede ser compleja y requiere conocimiento profundo del sistema y necesidades de los usuarios.
2. **Curva de aprendizaje:** Los usuarios deben familiarizarse con los comandos y la sintaxis de PBS.

3. **Optimización de políticas:** Encontrar el equilibrio entre eficiencia y satisfacción del usuario.
4. **Gestión de recursos heterogéneos:** En clústeres, la asignación eficiente de recursos puede ser un desafío.
5. **Integración con otras herramientas:** La integración con otras herramientas de HPC puede requerir configuración adicional.

Conclusiones y perspectivas futuras:

PBS es una herramienta sólida para gestionar recursos en computación de alto rendimiento (HPC). En el futuro, deberá adaptarse a nuevos desafíos, incluyendo:

1. **Computación en la nube:** Integración con entornos híbridos que combinen nube pública y privada.
2. **Contenedores y virtualización:** Mejor soporte para tecnologías como Docker y Kubernetes.
3. **Inteligencia artificial y aprendizaje automático:** Ajuste a las demandas específicas de cargas de IA/ML, diferentes de las HPC tradicionales.

4.5 - Patrones de diseño en programación paralela

1. **Master - Worker (Master-Slave)**
   Estructura:
   - Un proceso maestro que coordina y distribuye en trabajo
   - Varios procesos trabajadores que ejecutan tareas
   - Comunicación bidireccional entre maestro y trabajadores

   Ventajas:
   - Balanceo de carga dinámico
   - Fácil escalabilidad
   - Tolerancia a fallos

2. **Pipeline**
   Estructura:
   - Serie de etapas conectadas
   - Cada etapa procesa datos y pasa resultados a la siguiente
   - Permite paralelismo de flujo

   Ventajas:
   - Aprovecha el paralelismo de los procesos secuenciales
   - Buen rendimiento en procesamiento de streams
   - Uso eficiente de recursos

3. **Divide y vencerás**
   Estructura:
   - División recursiva del problema en subproblemas
   - Resolución paralela de subproblemas
   - Combinación de resultados

Ventajas:

- Escalabilidad natural
- Fácil de razonar y depurar
- Adecuado para problemas recursivos

4. **Map - Reduce**

   Estructura:

   - Fase Map: transformación paralela de datos
   - Fase reduce: agregación de resultados
   - Comunicación mínima entre nodos

   Ventajas:

   - Excelente escalabilidad
   - Tolerancia a fallos
   - Ideal para big data

5. **Stencil**

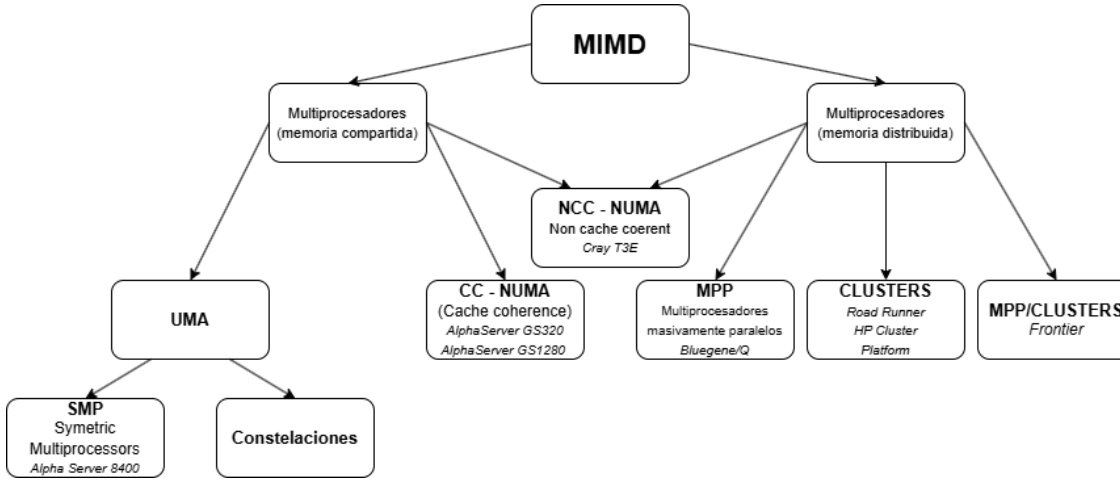
   Estructura:

   - Operaciones sobre matrices/grids
   - Cada elemento se actualiza basado en sus vecinos
   - Patrón común en simulaciones científicas

   Ventajas:

   - Localidad de datos
   - Paralelismo de datos natural
   - Eficiente en arquitecturas vectoriales

Tema 1 - Arquitecturas Comerciales



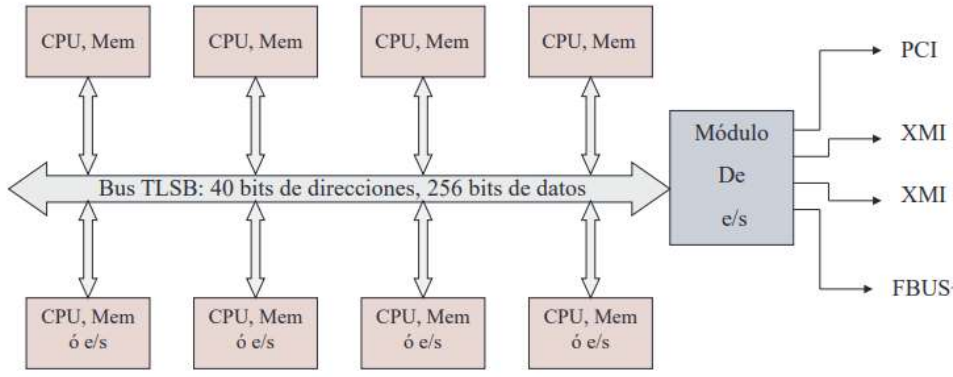
Tema 2 - Multiprocesadores

2.1 - Arquitectura SMP - AlphaServer 8400

Características principales:

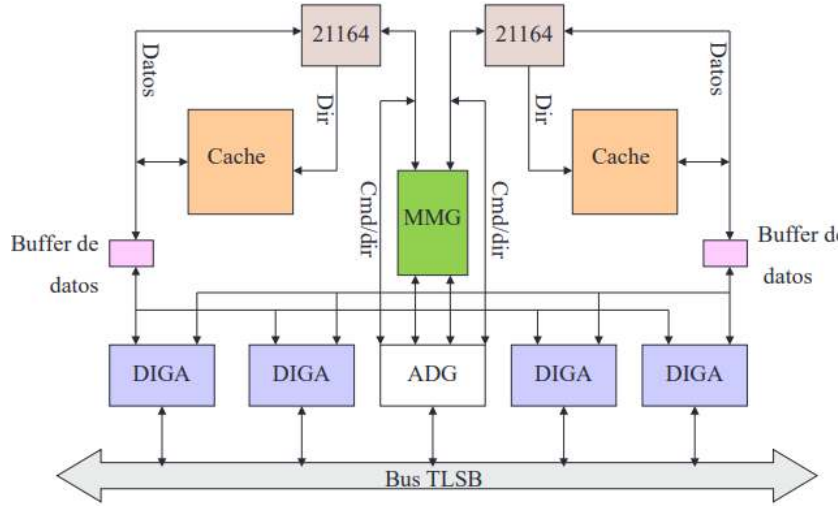
- Multiprocesador simétrico de **acceso uniforme a memoria**
- Hasta **12** CPU **Alpha 21164**
- Hasta 14 GB de memoria
- Bus de sistema de **3200MB/s**
- Sistemas operativos OpenVMS y Digital UNIX

Diagrama del sistema:



Un módulo de e/s, de CPU y de memoria obligatorios.

Módulo de CPU



2.1.1 - Alpha 21164

Características:

- Fabricante: Digital
- Año: 1996
- Tecnología: CMOS 0.35 micras
- Caché L1: 8 + 8 Kb. Write through.
- Caché L2: 96 Kb. Write back
- Caché L3 externa (opcional)
- Transistores: 9,3 millones.
- Potencia: 25W

2.1.2 - Módulo de CPU

- Puede albercas 1 o 2 CPUs
- Cada CPU trabaja independientemente y tiene su propia caché L3
- Soportan CPUs de 142 a 357 MHz
- La caché L3 es de 4MB con líneas de 64 bytes
- Dispone de multiplexores - demultiplexores (DIGA) para adaptar el ancho del micro (128) al del bus (256) usando el buffer de datos.
- El multiplexor de direcciones (MMG) alberga también el espacio de etiquetas duplicado.
- La interfaz de direcciones también recibe comandos y maneja el espacio de etiquetas duplicado → Responsable de la coherencia de caché.

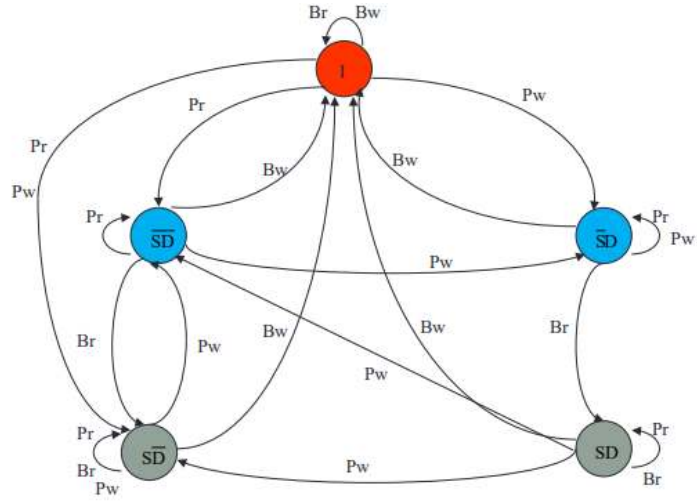
2.1.3 - Bus de sistema

- Capacidad para 9 módulos
- Uno de ellos debe ser de e/s, otro de memoria y otro de CPU. Si hay más de 3 CPU debe haber 2 de memoria.
- 40 bits de direcciones y 256 de datos
- Funcionamiento síncrono con el reloj múltiplo del de CPU (22-100 MHz)
- Velocidad máx de transferencia → 3200 MB/s
- Paridad simple en direcciones y comandos, ecc en datos.

2.1.4 - Módulo de memoria

- De 128MB a 2GB
- Máxima capacidad = 14 GB
- ECC
- Protocolo de escritura invalidación
- La escritura sobre bloques compartidos exige la toma del control del bus, haciendo que la memoria principal se actualice de forma automática.

2.1.5 - Protocolo de coherencia



- I: inválido
- S: compartido
- D: modificado
- Pw en una línea compartida actualiza la memoria principal y deshabilita D en la copia local.

2.1.6 - Módulo de e/s

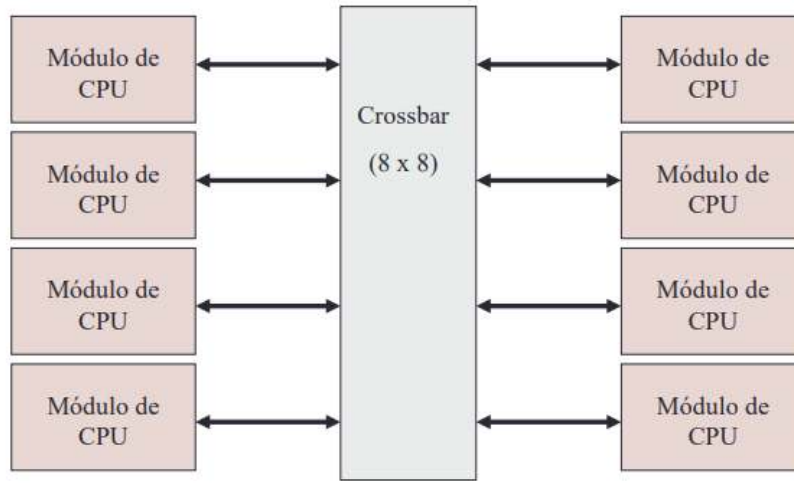
- Proporciona conexión a buses estándar:
  - XMI (2)
  - PCI
  - Futurebus+

- Se pueden conectar puertos a los nodos 4 a 7 también
- El nodo 8 es el de mayor prioridad de bus

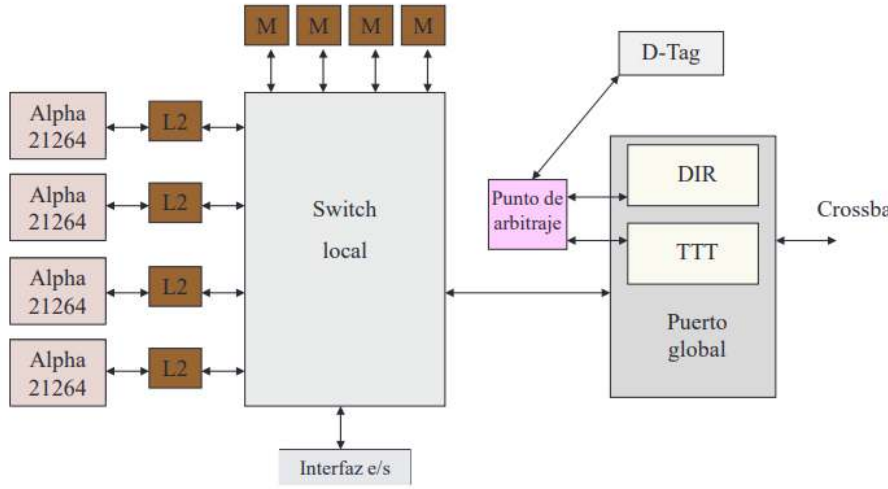
2.2 - Máquinas de acceso no uniforme a memoria - AlphaServer GS320

- Arquitectura CC-NUMA
- Módulos de 4 procesadores Alpha 21264
- Hasta 8 módulos (max 32 procesadores)
- Conexión crossbar 8x8 a 1,6 GB/s
- Directorios de mapeo completo
- Sistemas operativos: Tru64 UNIX, Open VMS, LINUX

Arquitectura:



Módulo de CPU



2.2.1 - CPU: Alpha 21264

- 731 MHz en esta máquina
- Cache L1 de 64 + 64 Kb parcialmente asociativa de dos vías
- Caché L2 externa de 4MB de mapeo directo
- Permite ejecución sin orden
- Pipeline simplificado

2.2.2 - Memoria

- 4 módulos de 1 a 8 GB cada uno
- Capacidad máxima de 32 GB por módulo
- Entrelazado de 8 vías en cada módulo
- Ancho de banda total de 6,4 GB/s
- 2 niveles de coherencia:
    - Intra-módulo: La tag duplicada funciona como directorio de mapeo completo
    - Inter-módulo: El directorio se encuentra en el puerto global

2.2.3 - Switch

- El switch local es parecido a un crossbar pero con una construcción asimétrica.
- No permite todas las posibles conexiones (no se pueden conectar módulos de memoria entre sí)
- No todas las conexiones tienen el mismo ancho de banda. Puerto global tiene ancho de banda doble.

2.2.4 - Coherencia

- Involucra varios elementos: DIR, TTT, D-tag y punto de arbitraje
- Bloques de 64 bytes con 14 bits: 6 de propietario y 8 para los poseedores a nivel de módulo, no de CPU
- Espacio de etiquetas duplicado permite identificar cada CPU
- El TTT es una tabla asociativa que mantiene una lista de hasta 48 transacciones pendientes de actualizar en memoria remota (write-backs)
- 4 posibles peticiones de acceso: lectura, lectura exclusiva, exclusivo y exclusivo sin datos.

2.2.5 - Desajustes de funcionamiento

- Una petición llega a un propietario después de un write-back
    - Se mantiene la línea en un buffer víctima hasta que se atienden todas las peticiones pendientes en la D-Tag.
    - Luego, se mantienen la línea en el TTT hasta que la memoria notifique su recepción
- Una petición llega a un propietario antes de que reciba los datos
    - El procesador compara la petición con la lista de fallos pendientes, si coinciden retrasa la petición

2.3 - Máquinas de acceso no uniforme a memoria - AlphaServer GS1280

- 64 CPU Alpha 21364 a 1,5 GHz
- Red toroidal 2D
- Rutado adaptativo mínimo con 3 canales virtuales
- Cada nodo tiene una CPU, memoria y e/s
- Coherencia basada en directorios de mapeo completo

2.2.1 - CPU: Alpha 21364

- Caché L2 integrado 1.75 Mb parcialmente asociativa de 7 vías
- Controlador de memoria integrado
- Router integrado para conexión a toro 2D
- 0,18 micras
- 152 millones de transistores

2.2.2 - Memoria

- Caché L1 integrada de 64 + 64 Kb parcialmente asociativa de 2 vías
- Cache L2 integrada de 1,75 Mb parcialmente asociativa de 7 vías con ECC
- Memoria principal de 8Gb por procesador con ECC

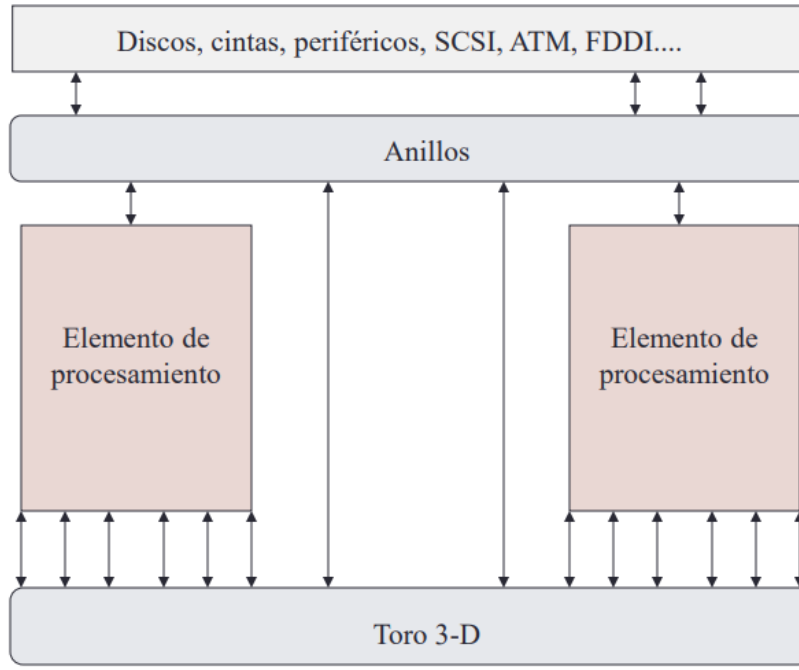
2.2.3 - Interconexión

- Toro 2D
- Enlaces bidireccionales de 6,2 Gb/s (2 x 3,1)
- Tres canales virtuales: dos DOR y uno adaptativo. Los paquetes viajan por el adaptativo, en caso de bloqueo saltan a los DOR
- Red libre de bloqueo: Interdimensional por ser DOR, intradimensional por tener canales virtuales.

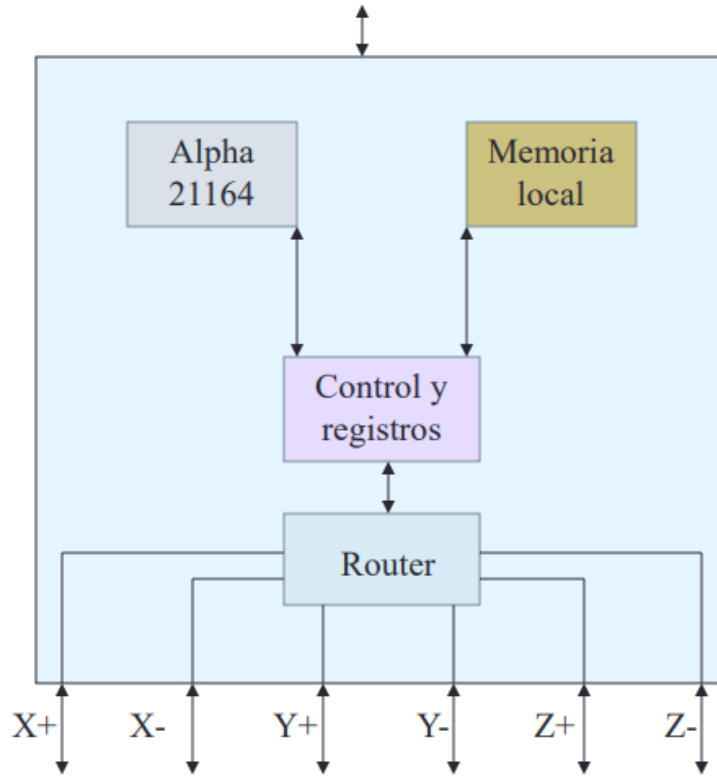
2.3 - CRAY T3E

- Máquina NCC-NUMA
- Hasta 2048 nodos Alpha 21164
- Red interconexión toroidal 3D
- Anillos adicionales para e/s
- Sistema operativo UNICOS

Arquitectura:



Elemento de procesamiento:



2.2.1 - Memoria

- Caché L1 y L2 integradas en el Alpha
- Memoria principal en cada nodo de hasta 2GB con ECC
- Sin mecanismo de hardware de coherencia global
- A nivel local se mantiene mediante el protocolo de tres estados de Alpha
- Los datos remotos se leen individualmente (no por bloques de caché) y se almacenan en los registros

2.2.2 - Interconexión

- Toro 3D con enlaces bidireccionales de 600 MB/s
- Conmutación de paquetes divididos en flits (wormhole)
- Encaminamiento adaptativo
- Anillos independientes formados por enlaces bidireccional (1 GB/s) a los que se conectan 16 nodos (de procesamiento, periféricos y puertos)
- Hasta 128 anillos

2.2.3 - Abrazos mortales

- Interdimensionales: Se solucionan combinando canallas virtuales y orden de dimensión (DOR)
- Intradimensionales: Eliminación (mecanismo de recuperación)

Tema 3 - Computador Masivamente Paralelo - MMP

3.0 - Blue Gene/Q: Arquitectura de Supercomputación Avanzada

Es el tercer representante de una serie que comenzó con el Blue Gene/L y continuó con el Blue Gene/P

- Desarrolladas por IBM para estar al frente del ranking mundial
- Versiones de tamaño intermedio llegan a alcanzar cierta difusión en el mundo de la supercomputación
- La implementación Sequoia, que llegó a alcanza nº1 mundial, se encuentra instalada en el laboratorio Lawrence Livermore de EE.UU e incluye un total de 1.572.864 núcleos de procesamiento.

3.1 - Evolución de Blue Gene

1. **Blue Gene/L - 2004**
   Primera generación, revolucionó la supercomputación con su diseño de bajo consumo y alta densidad.

2. **Blue Gene/P - 2007**
   Segunda generación, mejoró significativamente el rendimiento y la escalabilidad del diseño original.

3. **Blue Gene/Q - 2011**
   Tercera generación, su objetivo era liderar el ranking mundial de supercomputadores con un rendimiento sin precedentes.

3.2 - Características principales de Blue Gene/Q

- **Procesadores masivamente paralelos**
  Usa una arquitectura MIMD con hasta 1.572.864 núcleos de procesamiento en su configuración más grande.

- **Eficiencia energética**
  Diseñado para ofrecer un alto rendimiento con un consumo optimizado. Crucial para operaciones sostenibles a gran escala.

- **Escalabilidad**
  Arquitectura modular que permite configuraciones desde sistemas de tamaño medio hasta supercomputadores de clase mundial.

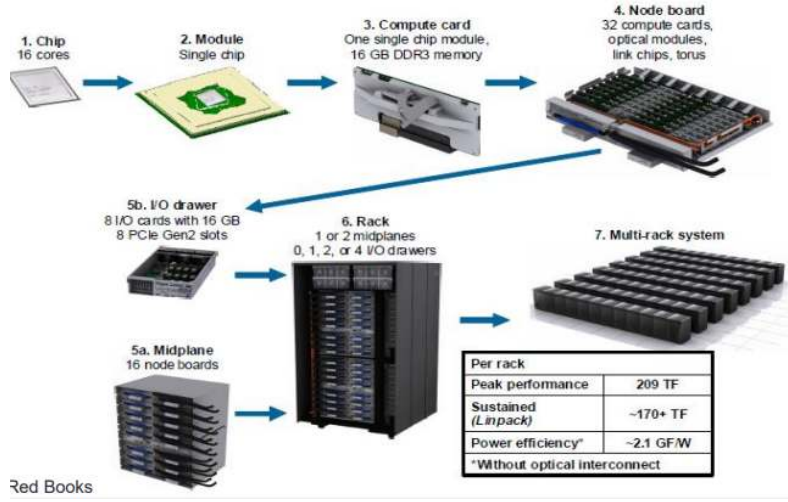
- **Interconexión Avanzada**
  Red de interconexión toro 5D que proporciona comunicación eficiente entre nodos.

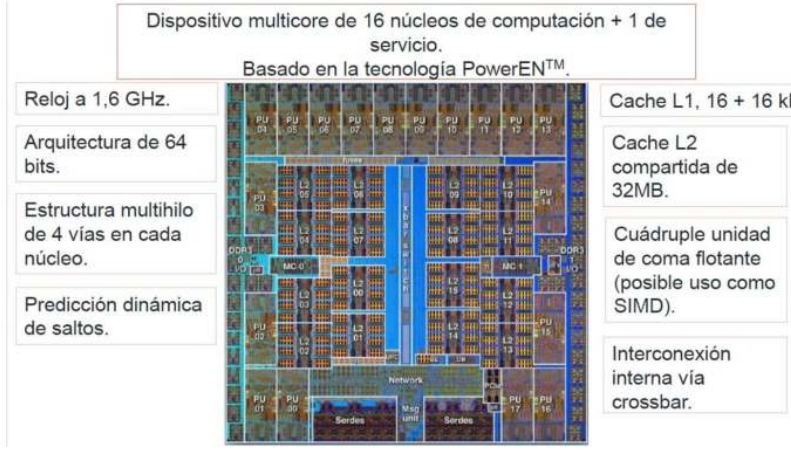
3.3 - Arquitectura Hardware de Blue Gene/Q

- **Tarjeta de Computación**
  Corazón del sistema, contiene un ASIC de computación y 72 chips de memoria SDRAM DDR3. Este diseño permite una alta densidad de cómputo y eficiencia energética.

- **Tarjeta Nodo**
  Agrupa 32 tarjetas de computación en un toro 5D (2x2x2x2x2). Proporciona una interconexión local eficiente entre los nodos de procesamiento.

- **Midplane**
  Conecta hasta 512 nodos de procesamiento (16 tarjetas nodo) en un toro 5D más grande (4x4x4x4x2). Permite una escalabilidad masiva del sistema.





3.4 - El Rack de Blue Gene/Q

- **2 Midplanes**
  Cada rack contiene 2 midplanes, lo que resulta en una configuración toro 5D de 4x4x4x8x2 nodos.

- **1024 nodos**
  Un rack completo tiene 1024 nodos de procesamiento, con una potencia masiva de cómputo relativamente compacto.

- **Interconexión avanzada**
  La red de interconexión dentro del rack permite una comunicación eficiente entre todos los nodos.

3.5 - ASIC de Computación: El Corazón de Blue Gene/Q

- **Núcleos de procesamiento**
  16 núcleos de computación más 1 de servicio, basados en PowerEN. Arquitectura de 64 bits a 1,6GHz

- **Multithreading**
  Estructura multihilo de 4 vías en cada núcleo, permitiendo un alto grado de paralelismo a nivel de hilo.

- **Unidad de coma flotante**
  Cuádruple unidad de coma flotante por núcleo, con posibilidad de uso como SIMD para operaciones vectoriales.

- **Caché**
  Caché L1 de 16 + 16 Kb (datos e instrucciones) y cache L2 compartida de 32 MB, optimizando el acceso a datos frecuentes.

3.6 - Jerarquía de memoria en Blue Gene/Q

1. **Registros del procesador**
   - Acceso más rápido, capacidad limitada.

2. **Caché L1**
   - 64 bytes por línea de caché.
   - 16Kb datos parcialmente asociativa de 8 vías.
   - 16Kb inst. Parcialmente asociativa de 4 vías.

3. **Caché L2 Compartida 32 MB**
   - Parcialmente asociativa de 16 vías. Líneas de caché de 128 bytes.
   - Soporta detección de condiciones de carrera entre núcleos, pero deben ser resueltas por software.

4. **DRAM Externa**
   - En tarjeta, con corrección de errores (ECC)

3.7 - Red de Interconexión: Toro 5D

- **Estructura**
  Toro 5D con configuración máxima de 16x16x16x16x2 nodos de procesamiento. Sequoia es (16x16x16x12x2).

- **Rendimiento**
  Enlaces bidireccionales que proporcionan 2 GBps en cada sentido.

- **Versatilidad**
  Crea redes colectivas embebidas para operaciones específicas, como arboles binarios para operaciones colectivas.

- **Conectividad**
  Conexiones eléctricas dentro de midplane, ópticas fuera de él.

3.8 - Encaminamiento en la Red de Interconexión

1. **Router integrado**
   La lógica de control de red, incluyendo el router y la unidad de mensajes, esta integrada en el ASIC de computación.

2. **Canales virtuales**
   Se implementan 4 canales virtuales: adaptativo, determinista, alta prioridad y de sistema, mejorando la flexibilidad y eficiencia de la comunicación.

3. **Colas múltiples**
   Uso de múltiples colas por canal virtual para reducir el bloque de principio de línea, optimizando el flujo de datos.

4. **Control de flujo**
   Se utiliza un control de flujo virtual cut-through permitiendo una transmisión de datos eficiente y de baja latencia.

3.9 - Unidad de Mensajes: Interfaz entre Red y Memoria

Juega un papel crucial en la eficiencia de la comunicación, actuando como intermediario entre la red de interconexión y la memoria del nodo.

- **Comunicación**
  Proporciona la interfaz entre la red y la memoria del nodo.

- **Colas FIFO**
  Se usan para la inyección y consumo de mensajes, conectando con el router.

- **Composición de mensajes**
  Se encarga de componer y descomponer los mensajes en paquetes para su transmisión eficiente.

3.10 - Interconexión - Sistema de Arbitraje en la Red

1. **Arbitraje distribuido**
   Las unidades de envío (emisores) informan de la disponibilidad de sus canales de salida y el espacio en las colas de destino en los nodos vecinos a las unidades de recepción y a las FIFOs de inyección.

2. **Selección de paquetes**
   El receptor de cada canal virtual selecciona con esa información un paquete al que dar servicio y envía una petición de arbitraje al árbitro de su receptor.

3. **Priorización**
   El árbitro del emisor asigna prioridades dependiendo de la naturaleza de los paquetes: colectivos, del sistema, de usuario de alta prioridad, de usuario prioridad normal. Dentro de cada categoría es posible configurar la prioridad relativa entre las colas de inyección y los canales virtuales.

4. **Arbitraje de inyección**
   Las FIFOs de inyección se arbitran parecido a los canales de red. La política de decisión final no está documentada.

3.10 - Mapeo de nodos en Blue Gene/Q

- **Mapeo por defecto**
  Los rangos MPI se incrementan en orden ABCDET, donde ABCDE son las coordenadas del toro 5D y T es la ID del nodo de procesamiento. Esta estrategia proporciona un mapeo lógico y eficiente para muchas aplicaciones.

- **Personalización**
  El comando ‘runjob’ permite modificar el mapeo por defecto. Se pueden permutar las coordenadas del toro o especificar un fichero de mapeo personalizado. Esta flexibilidad es crucial para optimizar el rendimiento.

- **Integración con planificadores**
  Los planificadores de lotes pueden incluir funcionalidades equivalentes a ‘runjob’ para lograr mapeos personalizados. Facilitando la integración del mapeo de nodos en flujos de trabajo más amplios de gestión de recursos.

3.10 - Software y Entorno de Desarrollo

- **Sistema Operativo**
  Kernel Linux completo en nodos E/S y Compute Node Kernel en nodos de computación.

- **Librerías de programación paralela**
  Soporte para MPI y OpenMP API, facilitando el desarrollo.

- **Herramientas de desarrollo**
  Compiladores especializados, depuradores y herramientas de análisis de rendimiento adaptadas.

- **Gestión de recursos**
  IBM LoadLeveler scheduler para la administración eficiente de trabajos y recursos

Tema 4 - Clusters

- **Acoplamiento Débil:** Permite flexibilidad de configuración y mantenimiento en cada nodo
- **Escalabilidad:** Capacidad de crecimiento según demanda, añadiendo más nodos al cluster
- **Comunicación por Red:** Intercambio de datos e instrucciones a través de redes de alta velocidad
- **Componentes comerciales:** Uso de hardware estándar para reducir costos y facilitar mantenimiento.

4.1 - Tipos de clusters

- **Homogéneos vs Heterogéneos:** Nodos idénticos o diversos en hardware y software.
- **Dedicados vs No Dedicados:** Exclusivos para HPC o aprovechando recursos de propósito general.
- **Clusters vs Constelaciones:** Agrupaciones locales o distribuidas geográficamente.

4.2 - Elementos de un cluster

1. **Software:** SO, middleware, aplicaciones
2. **Hardware de Red:** Switches, routers, cables de alta velocidad
3. **Nodos de procesamiento:** CPUs, GPUs, memoria, almacenamiento

4.2 - Nodo de Procesamiento

Cada nodo es una computadora completa, con uno o varios procesadores, memoria y almacenamiento y bus de sistema para la comunicación interna.

- **Procesadores:** Intel, AMD Opteron, GPUs para cómputo paralelo.
- **Memoria:** Alta capacidad y velocidad para procesamiento eficiente.
- **Almacenamiento:** SSD o HDD para datos locales y temporales.
- **Bus de sistema:** Interconexión rápida entre componentes del nodo.

4.3 - Hardware de Red

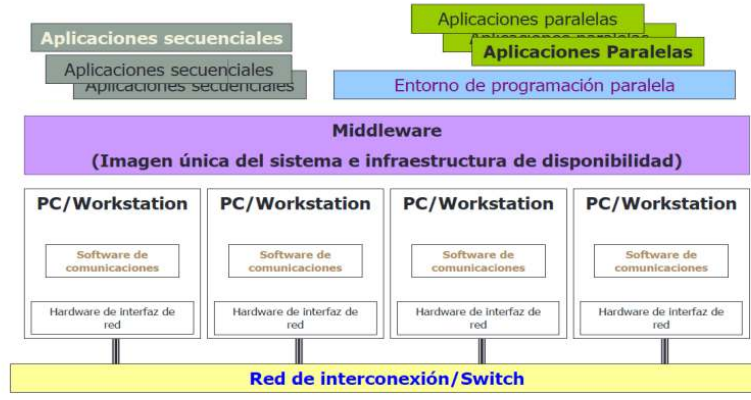
Es crucial para el rendimiento, ya que determina la velocidad y eficiencia de las comunicaciones entre nodos. Las tecnologías varían según el propósito, para no dedicados o “granjas”, se suelen usar LAN estándar, para clusters armados y de alto rendimiento se usan tecnologías como SAN (Storage Area Network) para mejorar la latencia y ancho de banda.

- **Tecnologías LAN para granjas:** Ethernet convencional para clusters no dedicados.
- **Gigabit Ethernet:** Mayor ancho de banda para clusters de gama media.
- **Tecnologías SAN Avanzadas:** Infiniband, Myrinet, baja latencia y alto rendimiento.
- **Otras Tecnologías Emergentes:** Nuevas soluciones como Omni-Path para clusters de última generación.

4.4 - Software en Clusters MIMD

Se estructura en varias capas, desde el SO hasta aplicaciones de usuario, usando middleware especializado y entornos de programación paralela.

- **Aplicaciones de usuario:** Secuenciales y paralelas.
- **Entorno de Programación Paralela:** MPI, OpenMP.
- **Middleware:** Gestión de recursos y tareas.
- **Sistema Operativo:** Linux, Windows HPC.



4.5 - Software de Sistema y Middleware

Proporcionan una base para la gestión eficiente de recursos y ejecución de aplicaciones paralelas. Los SO más comunes son distribuciones de Linux especializadas y Windows HPC Server.

El middleware incluye herramientas como Condor, MOSIX o OpenSSI que facilitan la gestión de recursos y la imagen única del sistema. Para programación paralela, PVM (Parallel Virtual Machine) y MPI son interfaces muy usadas.

- **Sistemas operativos**
  - Linux + Condor/MOSIX/OpenSSI
  - Windows HCP Server
- **Interfaces de programación**
  - PVM
  - MPI
- **Software de Red**
  - TCP/IP
  - VIA
  - Implementaciones de hardware(Infiniband, Myrinet)

4.6 - Caso de estudio: Roadrunner

Desarrollado por IBM en 2008 fue el primer sistema en alcanzar un rendimiento sostenido de 1 petaflop. Con un diseño híbrido (12k → Power Cell de IBM y 6,9k Opteron AMD) demostró la viabilidad de arquitecturas heterogéneas. Consumo de 3MW y coste de 130 M.

4.6.1 - Procesador Cell: Corazón del Roadrunner

Desarrollado por Sony, Toshiba e IBM, fué un componente clave del Roadrunner. Su arquitectura SIMD (Single Instruction Multiple Data) con 8 elementos de procesamiento lo hace ideal para computación paralela de alto rendimiento.

Cada elemento de procesamiento SPE (Synergistic Processing Element), podía operar independientemente en conjuntos de datos diferentes. Esta arquitectura fue cable para alcanzar el rendimiento del petaflop.

Arquitectura SIMD: 8 elementos de procesamiento