Coherencia de Caché en Arquitecturas Paralelas

Jose Alberto Benítez Andrades

71454586A

Arquitectura e Ingeniería de Computadores

Ingeniería en Informática

1.Clasificación de las arquitecturas paralelas.

La idea de utilizar múltiples procesadores es para aumentar desempeño y mejorar la disponibilidad datos anteriores a las computadoras electrónicas. Hace 40 años, Flynn propuso un modelo sencillo de clasificar todas computadoras que todavía hoy es útil. Él miró el paralelismo en las corrientes de instrucción y datos llamado por las instrucciones en el componente más fuerte del multiprocesador, y colocó todas computadoras en una de las cuatro categorías.

1. ***Flujo de instrucciones único, flujo de datos único (SISD)*** - Esta categoría es un uniprocesador.

2.***Flujo de instrucciones único, flujo de datos múltiple (SIMD)*** - La misma instrucción es ejecutada por múltples procesadores usando diferentes flujos de datos. Los ordenadores SIMD explotan el paralelismo a nivel de datos aplicando las mismas operaciones a los múltiples objetos de datos en paralelo. Cada procesador tiene su propia memoria de datos (de ahí datos múltiples), pero hay una memoria de instrucción simple y un procesador de control, que trae y lleva instrucciones. Para aplicaciones que muestran un paralelismo a nivel de datos significativo, el enfoque SIMD puede ser muy eficiente.

3.***Flujo de instrucciones múltiple, flujo de datos único (MISD)*** - No se comercializan hoy en día este tipo de multiprocesadores.

4. ***Flujo de instrucciones múltiple, flujo de datos múltiple (MIMD)*** - Cada procesador trae sus propias instrucciones y opera con sus datos. Los ordenadores MIMD explotan el paralelismo a nivel de hilo, desde múltiples hilos operando en paralelo. En general, el paralelismo a nivel de hilos es más flexible que el paralelismo a nivel de datos y así generalmente más aplicable.

A parte de estas categorías, existen también híbridos de todas ellas. Sin embargo, es bueno poner un marco en el espacio de diseño.

El modelo MMID puede explotar el paralelismo a nivel de hilos, esta es una arquitectura elegida para multiprocesadores de uso general. Otros dos factores contribuyen también al aumento de los multiprocesadores MIMD son:

1.MIMDs ofrecen flexibilidad. Con el soporte correcto de hardware y software, los MIMDs pueden funcionar como multiprocesadores de usuario único enfocados en un alto rendimiento para una aplicación, cuando multiprocesadores hacen funcionar tareas multiprogramación simultáneamente, o cuando se combinan algunas de estas funciones.

2.MIMDs pueden construirse con las ventajas coste / rendimiento de procesadores de plataforma apagada. De hecho, casi todos los multiprocesadores construidos hoy en día usan el mismo multiprocesador encontrado en las "*workstations*" (estaciones de trabajo) y servidores de un procesador. Por otra parte, los chips multinúcleo aprovechan la inversión de diseño en un procesador de un núcleo haciendo una réplica de este.

Una clase popular de ordenadores MIMD son los *clusters*, que a menudo usan componentes estándar y tecnologías de red estándar, así como los productos básicos de la tecnología posibles. Existen 2 tipos de *clusters*, los *clusters básicos*, que confían totalmente en los procesadores de 3 ... y en las interconexiones tecnológicas, y los *clusters personalizados,* en los cuales un diseñador personaliza cada detalle del nodo diseñado o las interconexiones de red o ambos.

En un cluster básico, los nodos de un cluster son a menudo hojas o servidores montados en rack (incluyendo los servidores multiprocesador de menor escala). Los Cluster básicos son a menudo reunidos por usuarios o directores de centro de ordenadores, en lugar de proveedores.

Los clusters personalizados están enfocados en aplicaciones en paralelo que pueden explotar gran cantidad de paralelismo en un único problema. Tales aplicaciones requieren una cantidad significativa de comunicación durante la computación y personalización del nodo y interconectando diseños hace tales comunicaciones más eficientes que en un cluster básico. Actualmente, los multiprocesadores más grandes y rápidos que existen son los clusters personalizados, como IBM Blue Gene.

Con un MIMD, cada procesador está ejecutando su propio flujo de instrucciones. En muchos casos, cada procesador ejecuta un proceso diferente. Un proceso es un segmento de código que puede estar funcionando independientemente; el estado de los procesadores contiene toda la información necesaria para ejecutar el programa en un procesador. En un ambiente de multiprogramación, donde los procesadores pueden estar ejecutando tareas independientemente, cada proceso es típicamente independiente de los otros procesos.

Esto también es útil para poder tener múltiples procesadores ejecutando un programa único y compartir el código y más sobre su espacio de dirección. Cuando varios procesos comparten código datos de esta manera, son los llamados hilos. Hoy, el término hilo es a menudo utilizado de forma ocasional para referirse a múltiples "loci" de ejecución que se puede ejecutar en procesadores diferentes, incluso cuando no comparten un espacio de direcciones. Por ejemplo, una ***arquitectura multihilo*** actualmente permite la ejecución simultánea de varios procesos, con espacios de dirección separados, así como varios subprocesos que comparten el mismo espacio de direcciones.

Para aprovechar las ventajas de un multiprocesador MIMD con *n* procesadores, debemos normalmente tener como mucho n hilos o procesos para ejecutar. Los hilos independientes dentro de un procesador de un núcleo son identificados por el programador o creados por el compilador. Los hilos pueden ser de gran escala, procesos independientes programados y manipulados por el sistema operativo. En el otro extremo, un hilo consiste en unas cuantas iteraciones de un bucle, generadas por un compilador paralelo explotando el paralelismo de datos en el bucle. Aunque la cantidad de computación asignada a cada hilo, llamada "grain size" (tamaño de grano), es importante considerar cómo explotar el paralelismo a nivel de hilos eficientemente, la importante distinción cualitativa del paralelismo a nivel de instrucciones es que el paralelismo a nivel de hilos es identificado como un nivel superior del sistema de software y que los hilos consisten en cientos de millones de instrucciones que pueden ser ejecutadas en paralelo.

Los hilos pueden también ser usados para explotar el paralelismo a nivel de datos., aunque la sobrecarga es normalmente mayor cuando es visto en un ordenador SIMD. Esta sobrecarga significa que el tamaño de grano debe ser lo suficientemente grande para aprovechar el paralelismo con eficacia.

Existen multiprocesadores MIMD que poseen dos clases, dependiendo del número de procesadores que participan, que a su vez dicta una organización de la memoria y estrategia de interconexión. Nosotros nos referimos a multiprocesadores por su organización de memoria porque lo que constituye un pequeño o gran número de procesadores es probable cambiar con el tiempo.

El primer grupo, que llamamos arquitecturas centralizadas de memoria compartida, tienen en la mayoría una docena de procesadores (y menos de 100 núcleos) en 2006. Para multiprocesadores con procesador de cuentas pequeñas, es posible para los procesadores compartir una memoria centralizada única. Con grandes cachés, una memoria simple, posiblemente con múltiples bancos, puede satisfacer las demandas de memoria de un número pequeño de procesadores. Usando muchas conexiones punto a punto, o un switch, y

añadiendo bancos de memoria adicional, una diseño centralizado de memoria compartida puede ser escalado a una docena de procesadores. Aunque escalando más allá que es técnicamente concebible, compartiendo una memoria centralizada se convierte en menos atractivo, como el número de procesos compartidos aumenta.

Porque hay una única memoria principal que tiene una relación simétrica para todos los procesadores y tiempo de acceso uniforme para cada procesador, estos multiprocesadores son más a menudo llamados, multiprocesadores simétricos, y este estilo de arquitectura es muchas veces llamado acceso de memoria uniforme, derivados de el hecho de todos los procesadores con una latencia de memoria uniforme, incluso si la memoria está organizada en varios bancos. Este tipo de memoria compartida simétrica es actualmente la organización más popular.

El segundo grupo consiste en multiprocesadores con memoria física distribuida. Para dar soporte a grandes cuentas de procesadores, la memoria debe estar distribuida entre los procesadores más bien centralizados ; de lo contrario el sistema de memoria no podrá soportar el ancho de banda demandado por el alto número de procesadores sin incurrir excesivamente en la latencia. Con el incremento rápido en el rendimiento de procesadores y el incremento asociado en el ancho de banda de memoria del procesador requerido, el tamaño de un multiprocesador para que la memoria distribuida preferentemente continúe reduciéndose. El gran número de procesadores también plantea la necesidad de interconectar anchos de bandas altos.

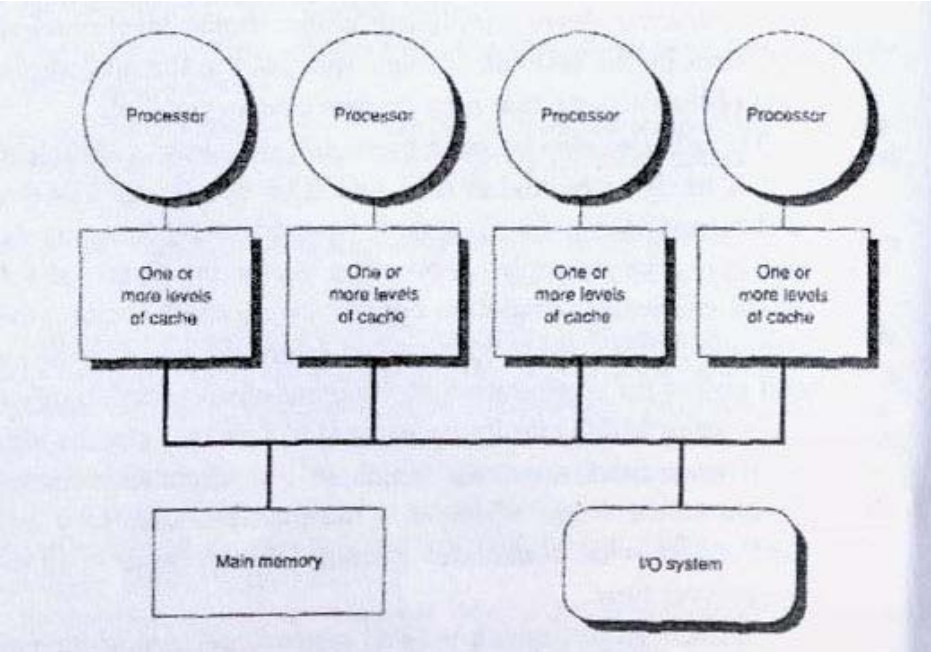


Figura 1. Estructura básica de un multiprocesador de memoria compartida centralizada. Múltiples subsistemas de procesadores caché comparten la misma memoria física, típicamente conectada por uno o más buses o un switch. La propiedad de la arquitectura vlave es el acceso uniforme a todas las memorias desde todos los procesadores.

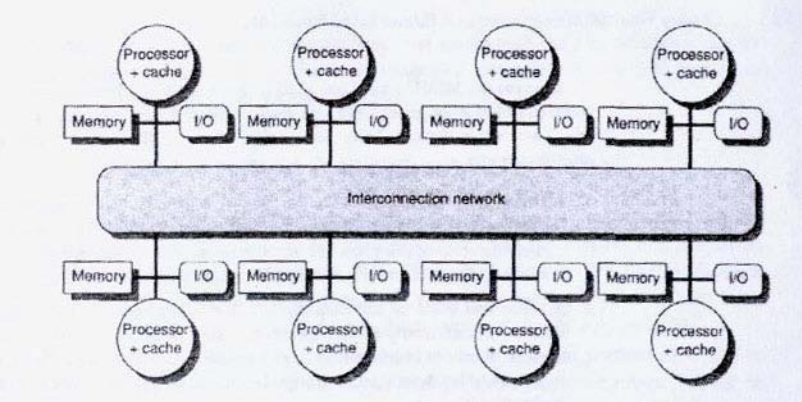


Figura 2.La arquitectura básica de un multiprocesador de memoria distribuida consiste en nodos individuales que contienen un procesador, alguna memoria, típicamente mucha E/S y una interface para una red de interconexiones que conecta todos los nodos. Los nodos individuales pueden contener un pequeño número de procesadores, que deben ser interconectados por un pequeño bus o una tecnología diferente de interconexión, que es menos escalable que la red global de interconexión.

Ambas redes de dirección (switches por ejemplo...) y redes indirectas (típicamente mallas multidimensionales) son usadas.

Distribuyendo la memoria entre los nodos tienes dos beneficios más.

- El primero, es un coste efectivo camino a escalar los anchos de banda de memoria si muchos de los accesos son a la memoria local en el nodo.
- El segundo, reduce la latencia para los accesos a la memoria local.

Estas dos ventajas hacen la memoria distribuida más atractiva para los pequeños procesadores que obtienen cada vez más rápido y requieren más ancho de banda de memoria y baja latencia de memoria. La clave de los inconvenientes para la arquitectura de memoria distribuida son las comunicaciones de datos entre procesadores que se vuelven cada vez más complejas, y esto requiere más esfuerzo en el software para coger ventaja sobre el incremento de ancho de banda de memoria brindado por las memorias distribuidas. El uso de memoria distribuida también se lleva a dos paradigmas diferentes para la comunicación de interprocesos.

2.Problema de la coherencia de la caché.

En una jerarquía para un sistema multiprocesador, la inconsistencia de datos puede ocurrir entre niveles adyacentes o dentro del mismo nivel. Por ejemplo, la memoria caché y principal pueden contener inconsistencia copiando datos del mismo objeto. Múltiples cachés pueden tener diferentes copias del mismo bloque de memoria porque múltiples procesadores operan de manera asíncrona independiente.

Las cachés en un ambiente de multiproceso introducen los problemas de coherencia de caché. Cuando múltiples procesos mantienen copias en caché local de una única localización de memoria compartida, cualquier modificación local en la localización puede resultar en una vista globalmente incoherente de memoria. Los planes de coherencia de caché evitan este problema por mantener un estado uniforme para cada bloque en caché de datos. Las inconsistencia de caché causadas por el compartimiento de datos, migración de procesos o E/S se explican continuamente.

Inconsistencia en la compartición de datos.

El problema de inconsistencia de cachés ocurre sólo cuando múltiples cachés privadas son usadas. En general, se identifican 3 fuentes de problemas : Compartición de datos escribible, proceso de migración, y actividad de E/S. Consideramos un multiprocesador con 2 procesadores, cada uno usando una caché privada y ambos compartiendo la memoria principal. Deja que X sea un dato compartido que ha sido referenciado por ambos procesadores. Antes de actualizar, las 3 copias de X son consistentes.

Si el procesador P1 escribe un nuevo dato X' dentro de la caché, la misma copia será escrita inmediatamente dentro de la memoria compartida usando una política de "write-through". En este caso, la inconsistencia ocurre entre las 2 copias (X' y X) en las 2 cachés.

Por otra parte, la inconsistencia puede ocurrir también cuando una política de "write-back" es usada. La memoria principal debe ser actualizada cuando el dato modificado en la caché es remplazado o invalidado.

Proceso de migración y E/S

En la figura de abajo se observa la aparición de incoherencia después de un proceso que contiene una variable compartida X y migra del procesador 1 al procesador 2 usando el write-back caché en el procesador correctamente. En el medio, un proceso migra del procesador 2 al procesador 1 cuando usa write-through cachés.

En ambos casos, la incoherencia aparece entre las 2 copias de caché, etiquetadas con X y X'. Debemos ejercer precauciones especiales para evitar las incoherencias. Un protocolo coherente debe ser establecido antes de que los procesos de forma segura puedan migrar de un procesador a otro.

Los problemas de incoherencia suelen ocurrir durante las operaciones de E/S que omiten la caché.

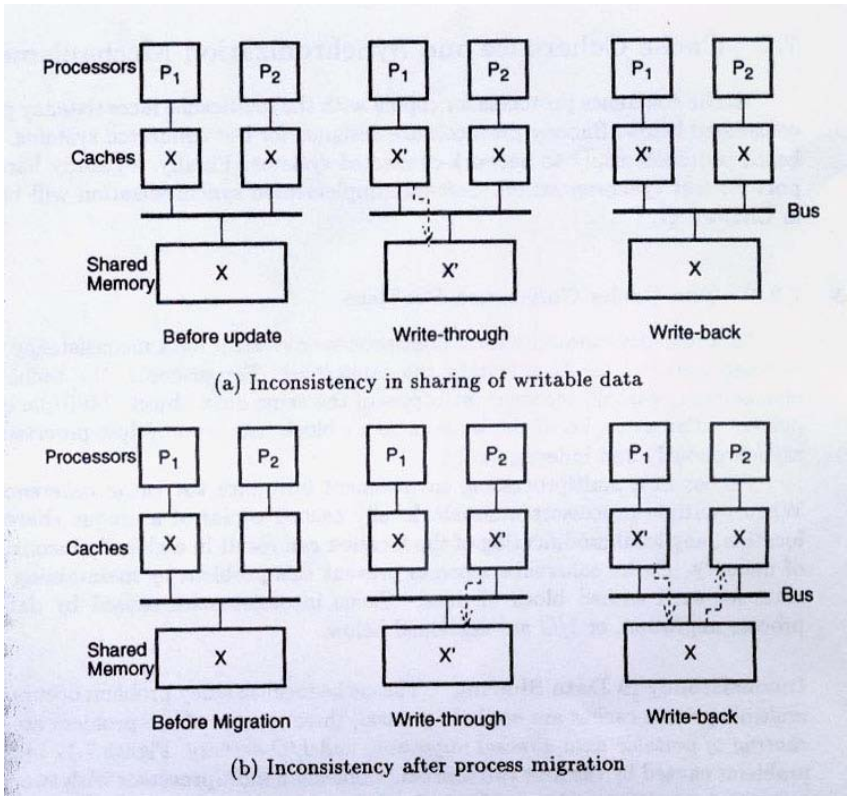


Figura 3: Problemas de coherencia de caché en la comparticion de datos y el proceso de migración.

Cuando el procesador de E/S carga un nuevo dato X' en la memoria principal, omitiendo el write-thorugh , la incoherencia ocurre entre la caché 1 y la memoria compartida.

Cuando produce directamente un dato desde la memoria compartida, las cachés write-back también crean la incoherencia.

Una posible solución al problema de incoherencia de E/S es agregar procesadores de E/S a las cachés privadas, respectivamente. De esta forma los procesadores de E/S comparten caches con la CPU. La coherencia de E/S puede mantenerse si la coherencia caché-a-caché es mantenida mediante un bus. Obviamente la deficiencia de este esquema es probable incrementar en las perturbaciones de la caché y la pobre localización de los datos de E/S, que pueden resultar en mayores proporciones de ratio.

Dos enfoques de protocolo

Muchos de los procesadores que se comercializan hoy en día usan sistemas de memoria basados en bus. Un bus es un dispositivo conveniente para garantizar la coherencia de las cachés porque este permite a todos los procesadores del sistema observar las transacciones de memoria en curso. Si una transacción amenaza el estado de coherencia de un objeto de la caché localizado, el controlador de caché puede tomar acciones apropiadas para invalidar la copia local. Los protocolos usan este mecanismo para garantizar la coherencia, esto se llama protocolos snoopy por los snoops que hay en cada transacción en las cachés.

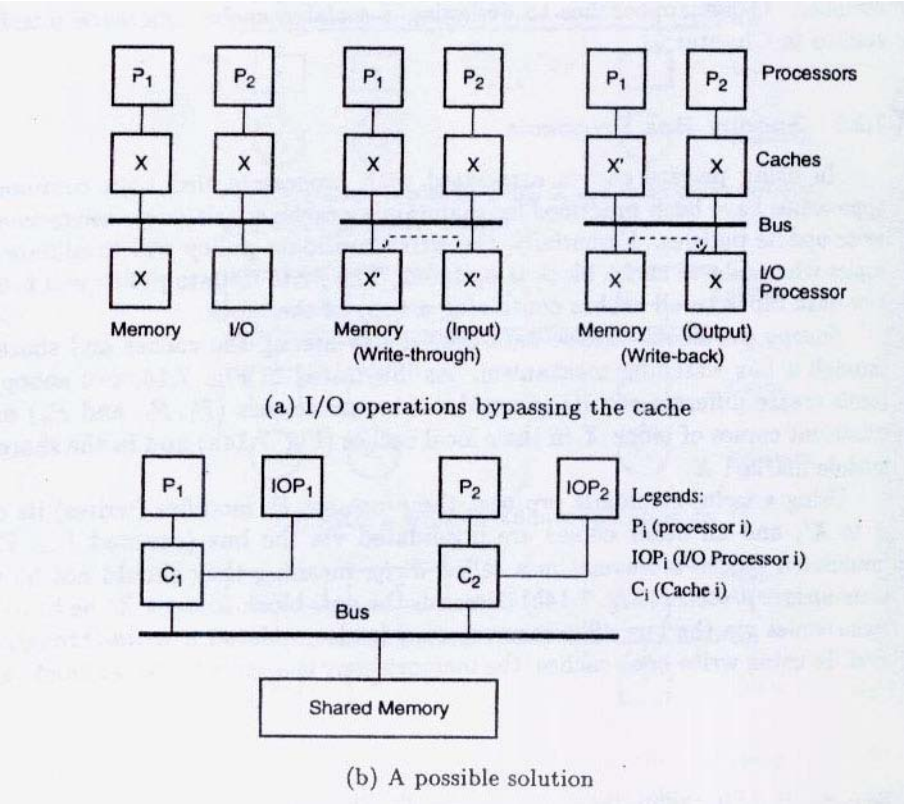


Figura 4: Incoherencia en la caché después de operaciones de E/S y posible solución.

Por otra parte, los sistemas multiprocesador escalables interconectan procesadores usando punto-a-punto en redes directas multiestación. A diferencia de la situación de los buses, el ancho de banda de estas redes se incrementa cuantos más procesadores sean añadidos al sistema. Sin embargo, tales redes no tienen un mecanismo conveniente de snoop y no proporcionan una emisión eficiente. En tales sistemas, el problema de coherencia de la caché puede ser solventado usando muchas variantes del directorio de planes.

En general, un protocolo de coherencia de caché consiste en establecer un posible estado en las cachés locales, el estado en la memoria compartida, y el estado en las transiciones causadas por el transporte de mensajes a través de la red de interconexión para mantener la coherencia de memoria. En lo que sigue, nosotros primero describiremos los protocolos snoopy y entonces el directorio de protocolos. Estos protocolos dependen del software, hardware o la combinación de ambos para implementarlos. La coherencia de caché puede ser aplicada en los TLB o asistida por un compilador.

3.Mecanismos de sincronización:

Protocolos snoopy:

En el uso de cachés privadas asociadas con los procesadores vinculados a un bus común, se han practicado dos enfoques para mantener la coherencia de la caché: políticas de write-invalidate y write-update. Esencialmente, la política write-invalidate invalidará todas las copias remotas cuando un bloque de la caché local sea actualizado. La política de write-update emitirá el nuevo bloque de datos a todas las cachés conteniendo una copia del bloque.

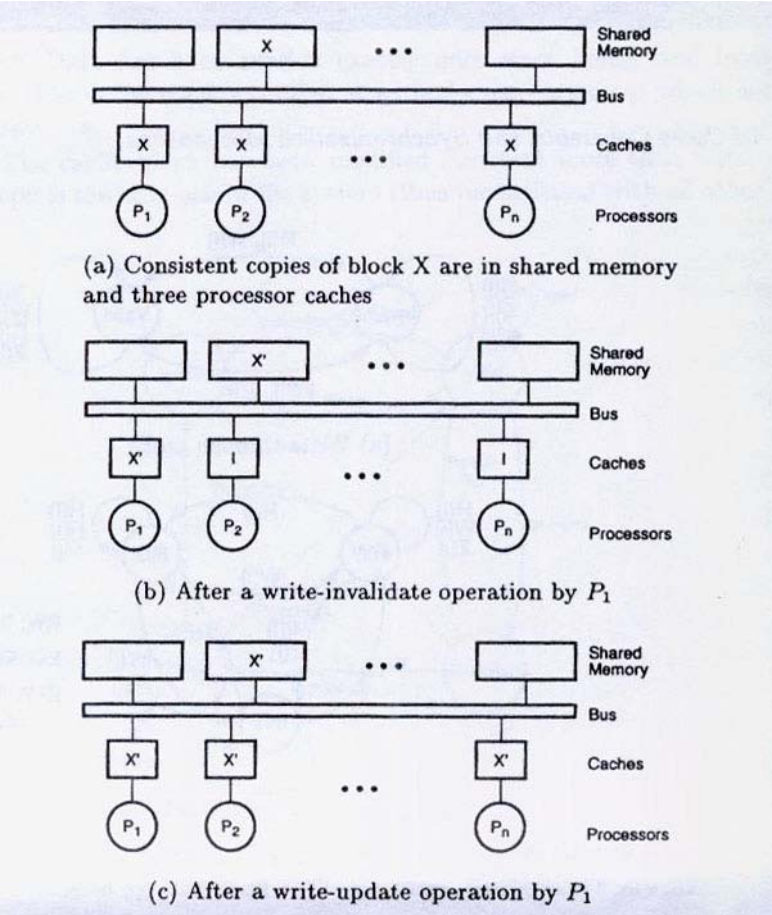


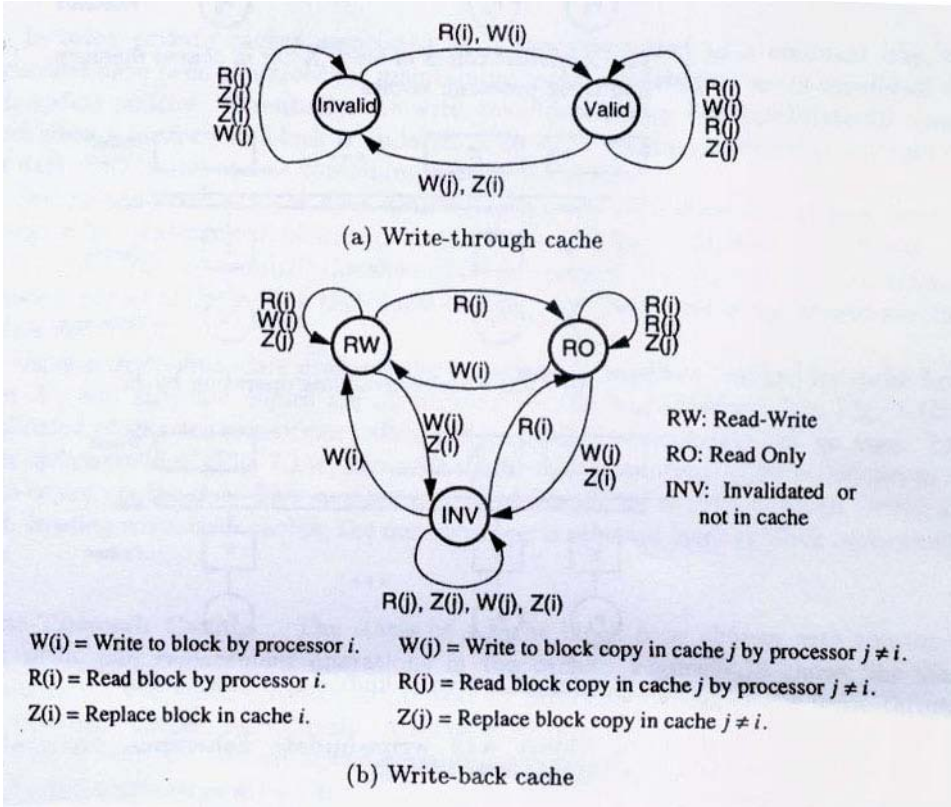
Figura 5: Protocolos write-invalidate y write-update para cachés write-through.

Los protocolos snoopy logran la coherencia de datos entre las cachés y memorias compartidas a través de un bus viendo el mecanismo. Como se ve en la **Figura 5**, dos buses de protocolos snoopy crean diferentes resultados. Consideramos 3 procesadores (P1,P2 y Pn) manteniendo copias consistentes del bloque X en las cachés locales y en los módulos de memoria compartida marcados con X.

Usando el protocolo write-invalidate, el procesador P1 modifica (escribe) sus cachés de X a X', y todas las otras copias son invalidadas mediante el bus (denotado como I en). Los bloques invalidados son a menudo llamados sucios, significando que ellos no son usados. El protocolo write-update (c) demanda un nuevo contenido de bloque X' será emitido a todas las copias de caché mediante un bus. La copia de memoria es también actualizada si las cachés write-through son usadas. En el uso de cachés write-back, la copia de la memoria es actualizada después del tiempo de reemplazamiento de bloque.

Cachés Write-through

Los estados de la copia de un bloque de caché cambia con respecto a leer, escribir y operaciones de reemplazamiento en la caché. La **Figura 6** muestra el estado de transiciones para dos protocolos snoopy write-invalidate diseñados por cachés write-through y write-back, respectivamente. Una copia de bloque de una caché write-through i agregado al procesador i puede asumir uno o dos posibles estados de caché: válido o inválido (**Figura 6 a**).



Un procesador remoto es denotado j, donde j es distinto de i. Para cada uno de los 2 estados de caché, son posibles 6 eventos que pueden llevarse a cabo. Notar que todas las copias de caché del mismo bloque usan el mismo gráfico de transición haciendo cambios de estado.

En un estado válido, todos los procesadores pueden leer (R(i),R(j)) de forma segura- El procesador local i puede también escribir (W(i)) de forma segura en un estado válido. El estado inválido corresponde al caso de que el bloque ya sea invalidado o reemplazado (Z(i) o Z(j)).

Donde un procesador remoto escribe (W(j)) en la copia de la caché, todas las demás copias de caché se convierten en invalidadas. El bloque de caché en la caché i se convierte en válido cuando lee correctamente (R(i)) o escribe correctamente (W(i)) es llevado fuera del procesador local i.

La fracción de ciclos de escritura en el bus es mayor que la fracción de ciclos de lectura en una caché write-through, debido a la necesidad de solicitudes de invalidación. El directorio caché (registro de los estados de la caché) puede ser hecho en dos copias o doble-portado a para filtrar fuera más invalidaciones. En este caso se cachean los bloqueos, y un Test&Set anónimo es aplicado.

Cachés write-back

El estado válido de una caché write-back puede ser más dividido en dos estados de caché, llamados RW(Read-Write) y RO (read-only) en la **Figura 6 b**. El estado INV (invalidated o not-in-cache) es equivalente al estado invalidado mencionado antes. El esquema de la coherencia de 3 estados corresponde a un protocolo propiedad.

Cuando la memoria posee un bloque, las cachés pueden contener solo copias RO del bloque. En otras palabras, múltiples copias pueden existir en el estado RO, y cada proceso tiene una copia (llamada un guardador de la copia) puede leer (R(i), R(j)) la copia segura.

El estado INV es insertado cuando un proceso remoto escribe (W(j)) su copia local o el procesador local reemplaza (Z(i)) su copia de bloque. El RW corresponde solo una caché existiendo en la propiedad del sistema del procesador local i. Leer (R(i)) y escribir (W(i)) puede ser de rendimiento seguro en el estado RW. Ya sea el estado RO o INV, el bloque de caché se convierte únicamente en propiedad cuando un local escribe (w(i)) y lo lleva acabo.

Antes de que un bloque sea modificado, la propiedad para el acceso exclusivo debe ser primero obtenida por un bus de transacciones read-only que es emitido a todas las cachés y memorias. Si es modificada una copia de un bloque de memoria existe en una caché remota, la memoria debe ser primero actualizada, la copia invalidada y la propiedad transferida a la caché solicitante.

Protocolo Write-once

James Goodman ha propuesto un protocolo de coherencia caché para multiprocesadores bus-based. Este esquema combina las ventajas de invalidaciones write-through y write-back. En orden para reducir el tráfico del bus, el primer write del bloque de caché usa el write-through.

Esto resultará en una copia de memoria coherente mientras todas las demás copias de caché son invalidadas. Después del primer write, la memoria compartida es actualizada usando una política de write-back. Este esquema puede ser descrito por la gráfica de 4 estados de la **Figura 7**. Los 4 estados de caché definidos son:

- Válido: El bloque de caché, que es coherente con la copia de la memoria, ha sido leído por la memoria compartida y no ha sido modificado.
- Inválido: el bloque no es encontrado en la caché o es incoherente con la copia de la memoria compartida.
- Reservado: Los datos han sido escritos exactamente una vez desde que se leyó de la memoria compartida. La copia de caché es coherente con la copia de la memoria, que es la única otra copia.
- Sucio: El bloque de caché ha sido modificado (escrito) más de una vez, y la copia de caché es la única en el sistema (por lo tanto incoherente con las otras copias).

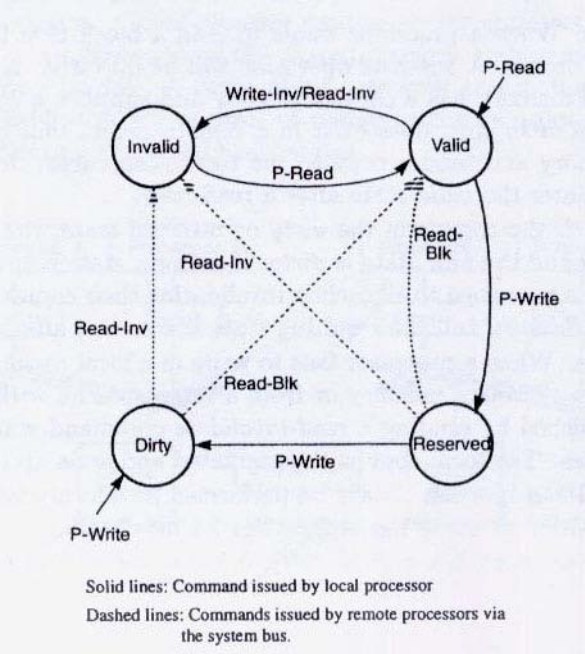


Figura 7:Protocolos de coherencia write-once de Goodman usando la política de write-invalidate en cachés write-back

Para mantener la coherencia, los protocolos requieren establecer dos comandos diferentes. Las líneas sólidas en la **Figura 7** corresponden a los comandos de acceso emitidos por un procesador local con la etiqueta read-miss, write-hit y write-miss. Cuando ocurre un read-miss, el estado válido es introducido.

El primer write-hit conduce al estado reserva. El segundo write-hit conduce al estado sucio y todos los futuros write-hits están en el estado sucio. Cuando un write-miss ocurre, el bloque de caché introduce el estado sucio.

Las líneas guiones corresponden a los comandos de invalidación emitidos por procesadores remotos mediante buses snoopy. El comando de read-invalidate, lee un bloque e invalida todos las otras copias. El write-invalidate invalida todas las copias de un bloque. El bus-read corresponde con una lectura de memoria normal por un procesador remoto mediante el bus.

Eventos y acciones de caché.

El acceso a memoria y los comandos de invalidación desencadenan los siguientes eventos y acciones:

- Read-miss: Cuando un procesador quiere leer un bloque que no está en caché, ocurre este tipo de errores. Una operación de lectura-bus será iniciada. Si no hay copias sucias, entonces la memoria principal tiene una copia consistente y suministra una copia a la caché que lo requiere. En una copia sucia existe una caché remota, que la caché inhibirá la memoria principal y envía una copia a la caché que lo solicita. En todos los casos, la copia de la caché introducirá el estado de válido después de un read-miss.
- Write-hit: Si la copia está en el estado de sucio o reservado, la escritura puede ser llevada a una salida localmente y el nuevo estado es sucio. Si el nuevo estado es válido, un write-invalidate es emitido a todas las cachés, invalidando sus copias. La memoria compartida es "written through", y el estado resultando es "reservado" después del primer write.
- Write-miss: Cuando un procesador falla al escribir en la caché local, la copia debe volver ya sea desde la memoria principal o desde la caché remota con un bloque sucio.
- Read-hit: Read-hits pueden siempre ser realizados en una caché local sin causar una transición de estado o usando el bus de snoopy para la invalidación.
- Block Replacement: Si una copia es sucia, ha sido "written back" a la memoria principal por el block replacement. Si la copia está limpia (ya sea válida, reservada o inválida), no se hará el reemplazo.

Los protocolos snoopy son populares en los multiprocesadores bus-based por su simplicidad a la hora de implementarlos.

El protocolo Futurebus+

Los protocolos paralelos Futurebus+ soportan ambos conectados y transacciones divididas. Generalmente hablando, las transacciones conectadas son baratas y fáciles de implementar en un segmento de bus simple. La transacciones de división son más complejas y caras, pero proporcionan gran concurrencia en las creaciones grandes de multiprocesadores de bus jerárquico. El futurebus+ es el mejor para los procesadores de memoria compartida. Los tipos de transacciones de bus son ajustados para conducir los estados de varios módulos de caché para conformar con casi cualquier protocolo de coherencia. Un modelo unificado de caché, llamado MESI, ha sido diseñado para mantener la coherencia de caché en un ambiente con múltiples Futurebuses lógicos conectados en una jerarquía.

Los repetidores de bus son usados para soportar transacciones de división, que insisten que una copia de una transacción debe ser repetida en buses adyacentes teniendo el mismo bloque de caché. Esto implica que el repetidor de buses contiene un buffer de caché que puede grabar el paso de un bloque de caché a un bus vecino. El protocolo de coherencia de caché propuesto por Futurebus es modificado del protocolo Goodman.

El protocolo snoopy trabaja para transacciones conectadas. Para transacciones de división sobre múltiples segmentos de bus, se necesitan módulos adicionales de interface. La **Figura 8** muestra una transacción de división, arquitectura de multiprocesador con memoria compartida.

Este sistema es una extensión de la arquitectura de memoria-compartida con módulos agente de caché y memoria. Un agente de caché usa transacciones de división para asumir todas las responsabilidades de un número de módulos de caché remotos. Los agentes de caché deben mantener etiquetas indicando el estado de los bloques de caché.

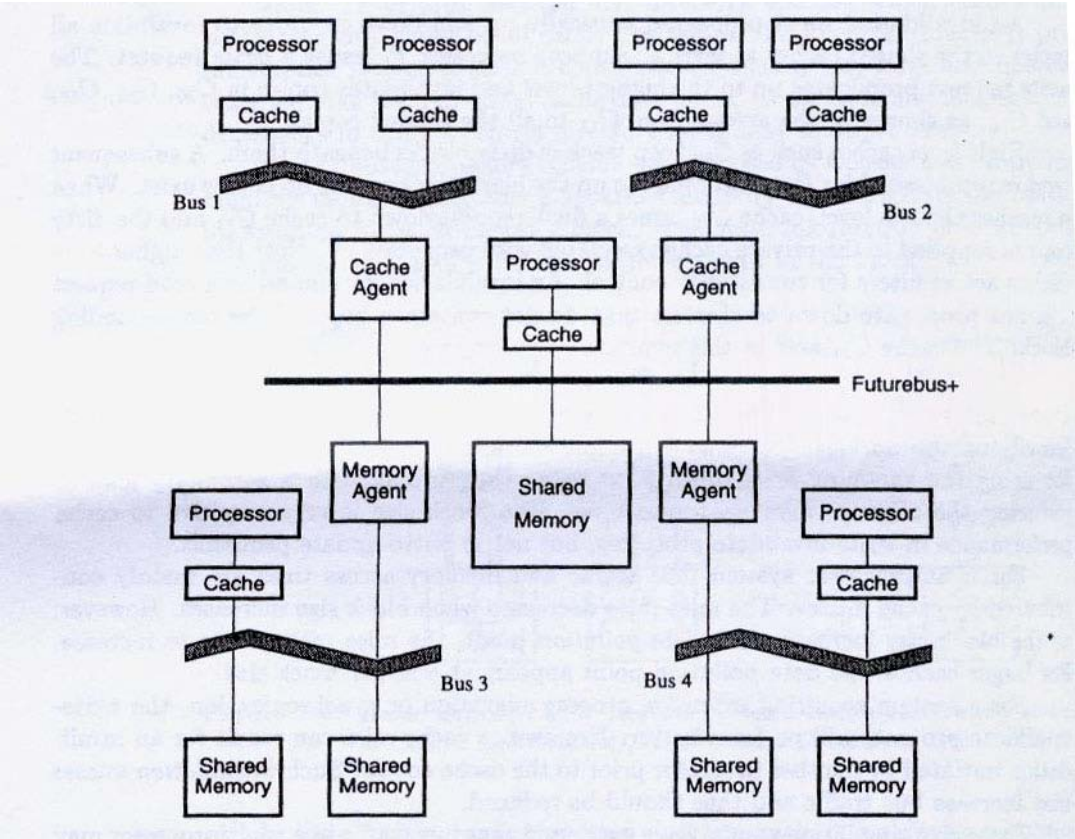


Figura 8: Transacciones de división en un multiprocesador.

Similar, un agente de memoria usa transacciones de división para asumir todas las responsabilidades de algunos módulos re memoria remota. Un agente de memoria divide una transacción para asignar permisos de lectura y escritura para la ubicación esto está manejado para que el segmento del bus se encienda. Esto también debe pasar invalidaciones o intervenciones al segmento del bus. Las transacciones de división han sido usadas para retrasar terminaciones de invalidaciones. Varios comandos de coherencia de caché son necesarios en los buses.

Coherencia de caché multinivel.

Para mantener la coherencia la coherencia entre las copias de caché a varios niveles, Wilson propuso una extensión para protocolos de write-invalidate usados en un bus simple. La coherencia entre las copias de caché al mismo nivel son mantenidas del mismo modo que se describió anteriormente. La coherencia de las cachés a diferentes niveles se ve en la figura 7.3.

Una invalidación debe ser propagada verticalmente arriba y abajo en orden para invalidar todas las copias en las cachés compartidas de nivel 2. Suponemos un procesador P1

pide un "write". El requerimiento de escritura propaga arriba para un nivel alto y invalida las copias en C20, C22, C16, y C18 como se ve en las flechas desde C11 a todas las copias sombreadas.

Las cachés de alto nivel como C20 mantienen un seguimiento de los bloques debajo de ellos. Una subsecuencia read requiere ser preguntado por P7 que propagará arriba la jerarquía porque no existen copias. Cuando llega al nivel superior, la caché C20 hace un requerimiento de vacío abajo a la caché C11 y la copia sucia es suministrada a las cachés privadas asociadas con el procesador P7. Notar que los actos de cachés de nivel alto como filtros para control de coherencia. Un comando de invalidación o requerimiento de lectura no propagará abajo a clusters que no contenga una copia de su correspondiente bloque. Los actos de la caché C21 actúan de esta manera.

Protocolo de problemas de rendimiento

El rendimiento de cualquier protocolo snoopy depende fuertemente en los patrones de carga de trabajo y eficiencia de implementación. La principal motivación para el uso del mecanismo de snooping es para reducir el tráfico del bus, con un objetivo secundario sobre reducir el tiempo de acceso a memoria de manera efectiva. El tamaño del bloque es muy sensible para el rendimiento de caché en los protocolos de write-invalidate pero no en los protocolos de write-update.

Para un sistema uniprocesador, el tráfico del bus y el tiempo de acceso a memoria son principalmente contribuidos por las pérdidas de caché. El ratio de pérdida se decrementa cuando el tamaño del bloque se incrementa. Sin embargo, como el tamaño del bloque se incrementa a un punto de polución de datos, el ratio de pérdida comienza a incrementarse. Para cachés grandes, el punto de polución de datos aparece como un bloque de gran tamaño.

Para un sistema que requieren un proceso extenso de migración o de sincronización, el protocolo write-invalidate mejora el rendimiento. Sin embargo, una pérdida de caché puede resultar para una invalidación iniciada por otro procesador prioritario a accesos de caché. Tales errores de invalidación pueden aumentar el tráfico de bus y por lo tanto, deberían reducirse.

Resultados de la simulación extensos han sugerido que el tráfico del bus en un multiprocesador debe incrementar cuando el tamaño de un bloque se incrementa. Write-invalidate también facilita la implementación o primitivas de sincronización. Típicamente, el

promedio de copias invalidadas de caché es más bien pequeña (1 o 2) en un multiprocesador pequeño.

El protocolo de write-update requiere que el bus tenga la capacidad de emitir. Este protocolo también puede evitar el efecto ping-pong en la compartición de datos entre múltiples cachés. Reduciendo la compartición de datos puede disminuir el tráfico en un multiprocesador write-update. Sin embargo, write-update no puede ser usado con ráfagas largas de escritura. Sólo a través de programas extensos de trazas pueden revelar el comportamiento de la caché, el ratio de hit (choque..), el tráfico del bus y el tiempo de acceso a memoria efectivo.

Protocolos basados en directorio.

Un protocolo write-invalidate puede conducir hasta tráfico pesado en el bus causado por read-misses, resultando desde el procesador actualizando una variable y otros procesadores intentando leer la misma variable. Por otra parte, el protocolo de write-update debe actualizar objetos de datos en cachés remotas que nunca serán usadas por otros procesadores. De hecho, estos problemas plantean limitaciones adicionales en el uso de buses para construir multiprocesadores grandes.

Cuando una red multietapas es usada para construir multiprocesadores grandes con cientos de procesadores, los protocolos de caché snoopy deben ser modificados para adaptarse a las capacidades de la red. Emitir es muy caro realizar en una red multietapas, los comandos de coherencia serán enviados sólamente a las cachés que guardan una copia del bloque. Esto llevan a protocolos basados en directorio para multiprocesadores de red-conectada.

Estructuras de directorios.

En una red multietapas, la coherencia de caché es soportada usando directorios caché para almacenar información en donde residen copias de bloques de caché. Varios protocolos basados en directorio difieren principalmente en cómo mantienen la información los directorios y qué información almacenan.

Tang propuso el primer esquema de directorio, que usó un directorio central que contenía una copia de todos los directorios caché. Este directorio central, proporcionaba toda la información necesaria para aplicar coherencia, es normalmente muy grande y deben

realizarse búsquedas asociativas, como directorios individuales de caché. Contención y a veces tiempos largos de búsqueda son dos inconvenientes en la utilización de un directorio central para un gran multiprocesador.

Un esquema de directorio distribuido fue propuesto por Censier y Feautrier. Cada módulo de memoria mantenía un directorio separado que guardaba el estado y la información presente por cada bloque de memoria. El estado de información es local, pero la información presente indica que las cachés tienen una copia del bloque.

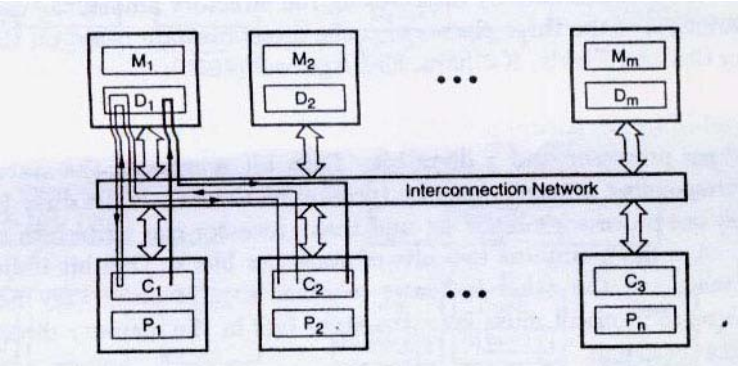


Figura 9: Concepto básico de un esquema de caché basado en directorio.

En la **Figura 9** un read-miss (líneas finas) en caché 2 resultados en un envío de solicitud al módulo de memoria. El controlador de memoria retransmite el pedido a una copia sucia en la caché 1. Esta caché 'write-back' su copia. El módulo de memoria puede suministrar una copia a el pedido de caché. En el caso de un write-hit en la caché 1 (líneas gruesas), un comando es enviado al controlador de memoria, que envía invalidaciones a todas las caché (cache2) marcadas en el vector presente residiendo en el directorio D.

Un protocolo de coherencia de caché que no usa emisiones debe almacenar las ubicaciones de todas las copias cacheadas de cada bloque de datos compartido. Esta lista de ubicaciones cacheadas, ya sea distribuida o centralizada, es llamada directorio caché. Una entrada de directorio para cada bloque de datos contiene un número de punteros para especificar las ubicaciones de las copias de cada bloque. Cada entrada de directorio contiene también un bit sucio para especificar si una caché única tiene permiso para escribir un bloque de datos asociado.

Diferentes tipos de protocolos de directorio caen debajo de tres categorías primarias: directorios full-map, directorios limitados y directorios encadenados. Los directorios full-map almacena suficientes datos asociados con cada bloque en la memoria global así que cada caché en el sistema puede almacenar simultáneamente una copia de cada bloque de datos. Esto es,

cada entrada de directorio contiene N punteros, donde N es el número de procesadores en el sistema.

Los directorios limitados difieren desde directorios full-map en los que ellos tienen un número fijo de punteros por entrada, a pesar de todo del tamaño del sistema. Los directorios encadenados emulan los esquemas de full-map por el directorio entre las cachés de distribución. Las siguientes descripciones de las 3 clases de directorios caché están basadas en la clasificación original por Chaiken, Fields, Kwihara y Agarwal:

Directorios Full-map

Los protocolos full-map implementan entradas de directorios con un bit por procesador y un bit sucio. Cada bit representa el estado del bloque en la correspondiente caché de procesador. Si el bit sucio está activo, entonces uno y sólo un bit de procesador es puesto y el procesador puede escribir en el bloque.

Una caché mantiene dos bits de estado por bloque. Un bit indica si un bloque es válido y el otro indica si un bloque válido puede ser escrito. El protocolo de coherencia de caché debe guardar los bits de estado en el directorio de memoria y entonces en la caché coherente.

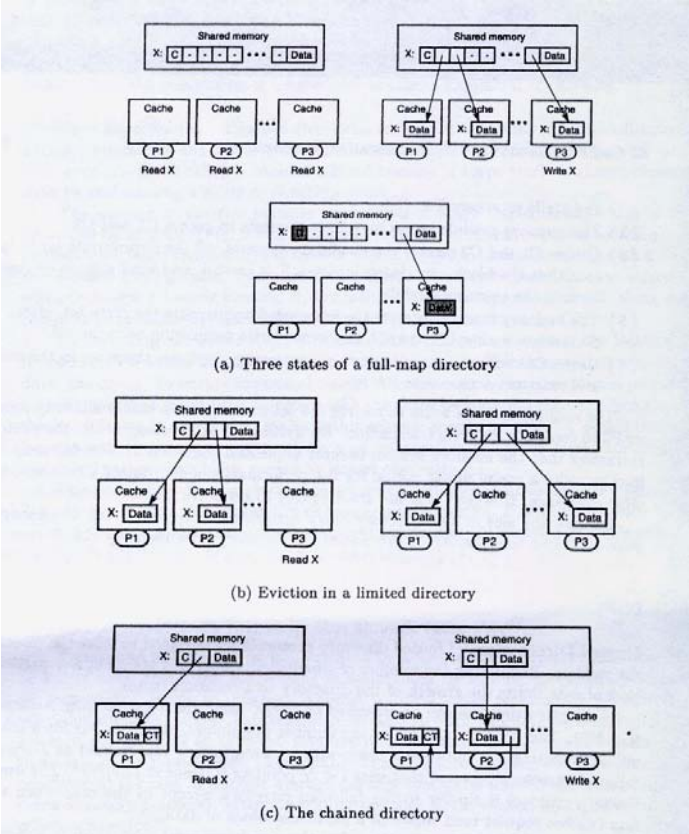


Figura 10: 3 tipos de protocolos de directorio de cachés.

La **Figura 10a** muestra 3 estados diferentes de un directorio full-map. En el primer estado, ubicada X es pérdida en todas las cachés en el sistema. El segundo estado resulta de 3 cachés (C1,C2 y C3) pidiendo copias de la ubicación de X. 3 Punteros (bits de procesador) son puesto en la entrada para indicar las cachés que tienen copias de cada bloque de datos. En los primeros dos estados, el bit sucio en la barra izquierda de la entrada de directorio están puestos para vaciar (C), indicando que el procesador no tiene permiso para escribir en el bloque de datos. El tercer estado resulta de la caché C3 pidiendo permisos de escritura para el bloque. En el estado final, el bit sucio es puesto a sucio (D), y hay un puntero simple para el bloque de datos en caché C3.

Veamos la transición desde el segundo estado al tercero con más detalle. Un procesador P3 cuestiona la escritura en la caché C3, los eventos siguientes tendrán:

- La caché C3 detecta que el bloque que contiene la ubicación X es válida pero que el procesador no tiene permisos de escritura en el bloque, indicado por el bit de permiso de escritura en la caché.
- La caché C3 realiza una petición de escritura al módulo de memoria que está en X y para el procesador P3.
- El módulo de memoria pide invalidación a las cachés C1 y C2.
- Las cachés C1 y C2 reciben la petición de invalidación, ponen el bit apropiado para indicar que el bloque contiene la ubicación X invalida y envían reconocimientos al módulo de memoria.
- El módulo de memoria recibe los reconocimientos, pone el bit sucio, borrar los punteros a las cachés C1 y C2, y envía permisos de escritura para la caché C3.
- La caché C3 recibe el mensaje de permisos de escritura, actualiza su estado en la caché y reactiva el procesador P3.

El módulo de memoria espera recibir los reconocimientos antes de permitir al procesador P3 completar su transacción de escritura. Esperando por los reconocimientos, el protocolo garantiza que el sistema de memoria garantiza coherencia secuencial. El protocolo de full-map proporciona un límite superior útil para el desempeño de la coherencia de caché centralizada basada en el directorio. Sin embargo, esto no es una copia escalable a la sobrecarga de memoria excesiva.

Porque el tamaño de la entrada de directorio asociada con cada bloque de memoria es proporcional al número de procesadores, la memoria consumida por el directorio es proporcional al tamaño de la memoria O(N) multiplicado por el tamaño del directorio O(N). La sobrecarga total de memoria escala como la raíz del número de procesadores O(N2).

Directorios Limitados

Los protocolos de directorio limitado son diseñados para resolver el problema de tamaño de directorio. Restringiendo el número de copias simultáneas de caché de un bloque de dato particular limitando el crecimiento de un directorio a un factor constante.

Un protocolo de directorio puede ser clasificado como *DiriX* usando la notación de Agarwal et al. El símbolo i se encuentra por cada número de punteros, y X es NB para un esquema que no emite. Un esquema full-map sin emisiones es representado como Dirn NB. Un protocolo de directorio limitado que usa i > N punteros es denotado *Diri N B*. El protocolo de directorio limitado es similar al directorio ful-map, excepto en el caso de que muchas más que i cachés realicen la petición de leer copias de un bloque de datos particular.

La **Figura 10b** muestra la situación cuando 3 cachés piden copias de lectura en un sistema de memoria con protocolo Dir2 N B. En este caso, podemos considerar el directorio de dos punteros como una caché de conjunto-asociativa bidireccional de punteros a copias copartidas. Cuando la caché C3 pide una copia de la ubicación X, el módulo de memoria debe invalidar la copia en cualquier caché C1 o caché C2. Este proceso de reemplazo de punteros es llamado "desalojo". Desde los actos de directorios una caché conjunto-asociativa, esto debe tener una política de reemplazo de punteros.

Si un multiprocesador exhibe un procesador localizado en el sentido de que cualquier intervalo de tiempo dado sólo un pequeño subset de todos los procesadores acceden a las palabras de memoria dadas, entonces un directorio limitado es suficiente para capturar un pequeño trabajador de procesadores.

Los punteros de directorio en un protocolo DiriNB codificand procesadores de identificadores binarios, así que cada puntero requiere log2 N bits de memoria, donde N es el número de procesadores en el sistema. Dada la misma hipótesis que para los protocolos full-map, la sobrecarga de memoria de los esquemas de directorio limitado crece coo O(N log2 N)

Estos protocolos son considerados escalables con respecto a la sobrecarga de memoria porque los recursos requieren implementarlos creciendo aproximadamente linealmente con el

número de procesadores en el sistema. Protocolos DiriB permiten más de i copias de cada bloque de datos que existe, pero recurren a un mecanismo de emisión cuando más de i copias cacheada de un bloque necesitan ser invalidadas. Sin embargo, redes de conexiones punto-a-punto no proporcionan una capacidad eficiente de emisión en todo el sistema. En esas redes, es dificil determinar la complejidad de un emisor para garantizar coherencia secuencial.

Directorios encadenados

Los directorios encadenados realizan la escalabilidad de directorios limitados sin restricción de número de copias compartidas de bloques de datos. Este tipo de esquemas de coherencia de caché son llamados esquemas encadenados, porque guardan pistas de copias compartidas de datos por mantener una cadena de punteros de directorio.

El más sencillo de los dos esquemas implementa un enlace simple encadenado. Suponemos que hay copias no compartidas en la ubicación X. Si el procesador P1 lee la ubicación X, la memoria envía una copia a la caché C1, alrededor con un punteo chain termination (CT). LA memoria también guarda un puntero a caché C1. Subsecuencialmente, cuando el procesador P2 lee la ubicación X, la memoria envía una copia a la caché C2, alrededor con el puntero a la caché C1. La memoria entonces guarda un puntero a la caché C2.

Repitiendo este paso, todas las cachés pueden cachear una copia de la ubicación X. Si el procesador P3 escribe la ubicación X, es necesaria para enviar un mensaje de invalidación de datos debajo de la cadena. Para garantizar coherencia secuencial, el módulo de memoria deniega al procesador P3 permisos de escritura hasta el procesador con el puntero de terminador de encadenado (CT). Quizá este esquema debe ser llamado un protocolo gossip (opuesto a los protocolos snoopy) porque la información es pasada desde individual a individual más bien entonces estando propagado por observación encubierta.

La posibilidad de reemplazar bloques caché complica los protocolos de directorio encadenado. Suponiendo que las cachés C1 a través de CN todas tienen copias de la localización X y esta localización X y localización Y del mapa a la misma línea de caché. Si el procesador Pi lee la ubicación Y, esta primero debe expulsar X desde su caché con las siguientes posibilidades de ubicación:

- Enviar un mensaje debajo de la cadena a la caché Ci-1 con un puntero a caché Ci+1 y empalme Ci fuera de la cadena o

- invalidar la ubicación X en caché Ci+1 a través de la caché C N.

El segundo esquema puede ser implementado por un protocolo de complejidad menor que el primero. En este caso la coherencia secuencial es mantenida observando la ubicación de memoria mientras las invalidaciones están en progreso. Otra solución para reemplazar el problema es usar una cadena doble de enlace. Este esquema mantiene adelante y atrás punteros de cadena para cada copia cacheada así que este protocolo no tiene que recorrer la cadena cuando la condición en el coste de un tamaño de bloque de mayor promedio de mensaje, dos veces el puntero de memoria en las cachés y un protocolo de coherencia más complejo.

Aunque los protocolos de encadenado son más complejos que los protocolos de directorio limitado, siguen escalables en términos de cantidad de memoria usada para los directorios. Los tamaños de los punteros crecen mediante un logaritmo de los números de procesadores, y el número de punteros por caché o bloque de memoria es independiente del número de procesadores.

Diseños de caché alternativos

Los méritos relativos de las direcciones físicas de cachés y direcciones virtuales de cachés para ser juzgadas se basan en el tiempo de acceso, problemas de alias, problemas de flujo, sobrecarga del núcleo del SO, marcado especial en el nivel de proceso, 3 diseños son los propuestos.

Cada diseño alternativo tiene sus propias ventajas y deficiencias. Existe insuficiente evidencia para determinar si una de las alternativas es mejor o peor que el uso de cachés privadas. Se necesitan más datos de investigación y seguimiento para aplicar estas arquitecturas de caché en el futuro diseño de multiprocesadores de alto rendimiento.

Cachés compartidas

Una enfoque alternativo para mantener la coherencia de caché es eliminar completamente el problema usando cachés compartidas adjuntas a módulos de memoria compartida. Cachés no privadas son permitidas en este caso. Este enfoque reducirá el tiempo de acceso a memoria principal pero contribuye muy poco a reducir el tiempo de acceso a memoria general y a resolver conflictos de acceso.

Cachés compartidas pueden crear cachés de segundo nivel. A veces, una puede hacer cachés de segundo nivel parcialmente compartidas por diferentes clusters de procesadores. Varias arquitecturas de caché son posibles si cachés privadas y compartidas son ambas usadas en una jerarquía de memoria. El uso de cachés privadas, cachés compartidas por clusters de multiprocesadores y memorias principales compartidas son tópicos interesantes para más investigaciones.

Datos no 'cacheables'

Otro enfoque no es caché de la escritura de datos compartidos. Datos compartidos son 'no caheables' ysólamente instrucciones o datos privados son cacheables en cachés locales. Los datos compartidos incluyen visión , colas de proceso, y algunas otras estructuras de datos protegidas por secciones críticas.

El compilador debe etiquetar datos como cacheables o no cacheables. Hardware especial para etiquetar debe ser usado para distinguirlos. Las cachés con bloques cacheables y no cacheables demandan más esfuerzo de programadores, además para soportar hardware y compiladores. Otra vez, esto abre nuevas cuestiones de investigación entre programabilidad y escalabilidad.

Vaciados de caché

Un tercer enfoque es usar vaciados de caché cada vez que se ejecuta una primitiva de sincronización de caché. Esto debe trabajar bien con transacciones procesando sistemas multiprocesador. Los vaciados de caché son lentos, a menos que se utilice un hardware especial. Este enfoque no resuelve los problemas de E/S y migración de procesos.

El vaciado puede hacerse muy selectivo por los programadores o por el compilador para aumentar la eficiencia. El vaciado de caché de sincronización , E/S y proceso de migración, puede realizarse incondicionalmente o de forma selectiva. Más a menudo se utiliza el vaciado de caché con cachés de dirección virtual.