Universidad
Europea de Madrid

LAUREATE INTERNATIONAL UNIVERSITIES

MULTIPROCESADORES

COHERENCIA DE CACHÉ

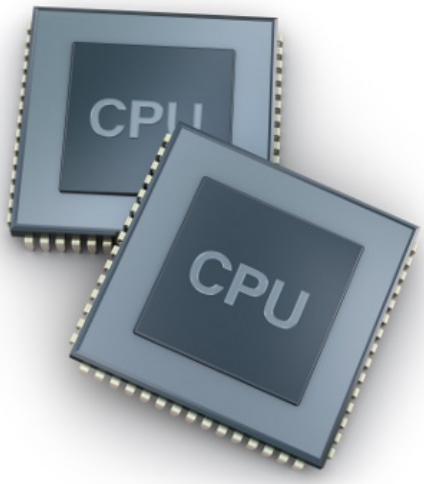
Índice

Presentación

En este tema veremos cómo solucionar el problema de la coherencia de caché cuando hay múltiples procesadores, tanto en sistemas monoprocesador con varios núcleos, como en sistemas con varios procesadores distribuidos.

Analizaremos cuáles son los problemas de la jerarquía de caché en sistemas multiprocesadores y diferentes métodos para evitar errores de caché, como:

- Protocolo Snooping (o fisgoneo).
  - Snooping para write through.
  - Snooping para write back.
- Coherencia basada en directorios.
  - Directorios full-map.
  - Directorios limitados.
- Directorios encadenados.



Acceso a caché en escritura

Ya hemos visto que la caché contiene **copia** de datos en memoria, nunca datos propios. Se considera un acierto de caché cuando el procesador intenta acceder a un dato, un dato que siempre está en memoria pero que, a veces, está replicado en memoria caché.

Y aquí es donde está la ventaja de la caché, si este dato se encuentra en caché, es la caché la que responde al procesador simulando que ya se ha producido el acceso al dato en memoria.

Si los accesos son en lectura todo es perfecto, ¿pero qué pasa cuando un acceso a un dato es en escritura y la caché tiene una réplica del mismo? El acceso, una vez más, se para en la caché, por lo que la memoria nunca se entera de que ha habido una modificación del dato. Como consecuencia, el dato almacenado en caché y en memoria no son el mismo, lo que se conoce como **incoherencia de caché**.

Ya hemos visto también que esto tiene una solución, bastante sencilla, las políticas de escritura. Hemos visto que existen básicamente dos métodos que funcionan bien pero que, como siempre, no son perfectos.

Escritura inmediata: write through	La escritura inmediata es segura pero poco eficiente, ya que todos los accesos a memoria en escritura provocan fallo de caché, lo que reduce a la mitad el índice de aciertos por parte de la caché. Hay técnicas que mejoran este problema, dejando la responsabilidad a la caché de actualizar el dato en memoria, dejando que el procesador continúe con la ejecución de código antes de que la memoria se actualice por completo. El inconveniente es que carga de trabajo a la caché, haciéndola sustancialmente más lenta.
Escritura retardada: write back	La escritura retardada, por el contrario, penaliza más los fallos de caché, pues si hay un fallo de caché y hay que reemplazar un bloque de datos de la misma, si el bloque tiene activada la señal de bloque modificado, antes de proceder al reemplazo hay que escribir el bloque en memoria y, después, proceder a la carga del nuevo bloque. Esto añade retrasos en la caché cada vez que hay que reemplazar un bloque que esta modificado (prácticamente todos). Al igual que en el caso de la escritura inmediata, hay técnicas que mejoran este comportamiento. Una de ellas es hacer que la caché, en sus momentos ociosos y de manera autónoma, revise qué bloques se han modificado y vaya actualizando la memoria para que, si hay reemplazo sobre ellos, ya se haya realizado la mitad del trabajo.

Una gran ventaja

La ventaja no es otra que el tiempo de espera que el procesador ha tenido que esperar es muy inferior al que hubiera tenido que hacerlo si el dato no se encontrara en caché.

Escritura inmediata: write through

Los accesos en escritura siempre producen fallo de caché, por lo que la petición de escritura siempre llega a la memoria principal. No se pueden dar problemas de caché porque el dato en memoria no se queda desactualizado en ningún momento.

Escritura retardada: write back

La memoria caché, mantiene información interna de que datos han sido actualizados por una escritura, y estos datos antes de ser eliminados de caché, son escritos en memoria. Los datos permanecen desactualizados solo mientras hay una copia de los mismos en caché, pero nunca se pierden datos, porque la caché se guarda de actualizar la memoria antes de eliminar los datos actualizados

Multiprocesadores o multinúcleos

Como hemos visto en temas anteriores, el desarrollo de sistemas multiprocesadores o multinúcleos está muy en boga.

¿Pero que pasa en estos sistemas con la caché?

Lo primero que tenemos que pensar es cómo se organiza la jerarquía de memoria en estos sistemas, diferenciemos multiprocesadores y multinúcleos.

Multiprocesador	Máquina con dos a más procesadores completos. Normalmente podemos hablar de una sola maquina con más de un procesador o incluso en dos máquinas diferentes: UMA vs NUMA.
Multinúcleo	Procesador con dos o más núcleos en un solo chip. Una máquina que tiene un solo procesador, internamente construido por más de un núcleo funcional, instalado.

En ambas máquinas la jerarquía de memoria es muy diferente, pero tienen un problema, aunque sea a niveles diferentes.

¿Qué pasa con un dato que se necesite en más de un procesador?

Una variable compartida por más de un procesador está replicada en los sistemas de memoria de cada una. Recordemos que esta es una máxima a cumplir, para agilizar los accesos a esa variable. Por ejemplo, este es el principio de existencia de la caché.

Si la variable está replicada en la memoria común a todos los procesadores, ya que ambos deben poder acceder a ella, por lo tanto no es un dato local. Además, está replicada en cada una de las memorias locales ya el acceso a los datos debe ser lo más rápido posible, consiguiendo reducir latencia en los accesos.

¿Qué pasa si al acceso a esta variable, no es solo en lectura?

Tras una serie de accesos al dato en escritura, las copias de la variable en las cachés de los procesadores tendrán datos diferentes, ya que no es suficiente con que cada procesador se encargue de mantener el valor correcto de la variable en la memoria compartida cada vez que se actualiza, sino que es necesario que el valor correcto de esta se chequee antes de acceder a él en lectura.

Multiprocesadores o multinúcleos: ejemplo

Para simplificar el ejemplo, quitemos las cachés de nuestro problema. Para ello supongamos una maquina UMA con dos procesadores. El sistema completo tiene una memoria privada para cada procesador y una memoria compartida a todos los procesadores para el intercambio de información.

Supongamos que ambos procesadores acceden al *dato compartido i* donde guardan información de sincronización sobre las operaciones que están realizando conjuntamente. Supongamos la siguiente secuencia de accesos a la *variable i*.

Pasos	Procesador	Acción	Memoria compartida	Memoria local procesador 1	Memoria local procesador 2
		Valor Inicial	5	¿?	¿?
Paso 1	Procesador 1	Lee i	5	5	¿¿
Paso 2	Procesador 2	Lee i	5	5	5
Paso 3	Procesador 1	Incrementa i en 3	8	8	5
Paso 4	Procesador 2	Incrementa i en 1	6	8	6

En detalle Interpretación de los datos

¿Cuál de ellas tiene el valor correcto de la variable i?

Nadie. El valor correcto debería ser 9 (5+3+1), y dicho valor que no está almacenado en ninguna de las memorias.

Este problema, ocurre siempre que haya una memoria más rápida enmascarando a otra memoria mayor y compartida, pero más lenta, y haya datos compartidos entre todos los procesos. Esto significa que cuando hay más de una memoria caché de una memoria y hay datos compartidos entre ellas, estos datos pueden no ser coherentes, por eso se le llama *incoherencia de caché*. La incoherencia de caché es complicada de solucionar, aunque no imposible, y solo se da en sistemas multiprocesador o multinúcleo.

Dato en más de un procesador

La memoria caché, mantiene información interna de que datos han sido actualizados por una escritura, y estos datos antes de ser eliminados de caché, son escritos en Memoria. Los datos permanecen desactualizados, solo mientras hay una copia de los mismos en caché, pero nunca se pierden datos, porque la caché se guarda de actualizar la memoria antes de eliminar los datos actualizados

Interpretación de los datos

- Cuando los procesadores leen la variable compartida en los pasos 1 y 2, todo va bien.
- En el paso 3, el procesador 1 accede a la variable en escritura guardando un 8 (5+3), por lo que su copia local almacena un 8 y se encarga de actualizar la copia en la memoria compartida para futuros accesos. Sin embargo, la copia local del procesador 2 no se actualiza, provocando el problema: esta guarda un 6 en la misma variable. Después de calcular 5+1, en su copia local queda un 6 y en la memoria compartida otro 6.

Jerarquía de caché

Actualmente, el problema de la incoherencia de caché está muy extendido a todos los niveles, ya que la mayoría de los procesadores que se comercializan son multinúcleo. Probablemente, habremos escuchado hablar de que nuestro sistema tiene caché L1, L2 o incluso L3.

Ya hemos hablado de que cada nivel de caché es una caché de la caché anterior. Por lo tanto, estas cachés pueden incluirse como etapas en nuestra jerarquía de memoria, o podemos hablar de que la caché se encuentra jerarquizada dentro de la propia jerarquía de memoria.

En ambos casos, la idea es la misma, el problema es cómo se organiza esta caché en los procesadores actuales. Tomemos como ejemplo dos procesadores más o menos actuales.

Especificaciones del procesador Intel Core i7

- Caché L1 dedicada a cada núcleo, además dividida en datos e instrucciones.
- Caché L2, caché clásica, pero una para cada procesador.
- Caché L3 compartida para todos los núcleos.

Como podemos comprobar, este esquema apenas presenta diferencias con el que vimos en el ejemplo del tema anterior, la única diferencia existente se encuentra en los nombres que hemos dado a cada una de las memorias.

Por lo tanto, las políticas de coherencia de caché toman una gran importancia actualmente, ya que su utilidad se aplica tanto en los súper computadores con multitud de procesadores y configuraciones UMA vs NUMA como en los ordenadores personales, con múltiples procesadores en un chip.

Para terminar, veamos qué indica AMD respecto a su procesador AMD PhenomTM II.

Especificaciones del procesador AMD PhenomTM II

Especificaciones del procesador Intel Core i7

Intel, respecto a la caché de uno de los microprocesadores Intel Core i7, en su hoja de características, especifica:

- Up to 6 Execution Cores, each core supports two threads (Intel(r) Hyper-Threading Technology) for up to 12 threads.
- A 32-KB instruction and 32-KB data first-level caché (L1) for each core.
- A 256-KB shared instruction/data mid-level (L2) caché for each core.
- Up to 15 MB last level caché (LLC): up to 2.5 MB per core instruction/data last level caché (LLC), shared among all cores.

Especificaciones del procesador AMD PhenomTM II

- L1 Cache (Instruction + Data) 64K of L1 instruction and 64K of L1 data cache per core.
- L2 Cache (total dedicated) 2MB per core.
- L3 Cache (total dedicated) 4MB to 6Mb shared to all cores.

Esquemas básicos de coherencia de caché

La primera manera de asegurar la coherencia de caché en sistemas de memoria compartida, es mediante la diferencia entre migración y replicación.

Migración	Un dato puede **moverse** a una caché local y utilizarse ahí de forma transparente. Mientras un dato está migrado, ese dato no puede estar en ninguna otra caché. La migración reduce la latencia de acceso a los datos y la demanda de ritmo de transferencia de los datos compartidos con la memoria compartida.
Replicación	Los datos compartidos se pueden encontrar en varias cachés simultáneamente. La replicación reduce la latencia de acceso y la disputa de la lectura de los datos compartidos.

Estos dos métodos son críticos en las prestaciones de sistemas multiprocesador, por lo que los sistemas suelen emplear diferentes sistemas para asegurar la coherencia de caché. Vamos a distinguir dos tipos de problemas.

Los multinúcleos	Asegurar la coherencia de caché entre procesadores cercanos, normalmente dentro de un mismo chip mediante el uso de algoritmos de fisgoneo (snooping) y de memoria *no cacheable*.
Los multiprocesadores	Sistemas con varios procesados lejanos normalmente situados en maquina diferentes y unidos por sistemas de transferencia de información tipo Ethernet Gigabit. Estos no tienen un bus que los comunique a todos, por lo que las técnicas de snooping no funcionan y se sustituyen por protocolos basados en directorios.

El sistema de memoria no cacheable es uno de los más simples y de los menos eficientes, por lo que su uso no está muy extendido.

Se basa en que los datos compartidos pueden ser conocidos por el sistema operativo ya que, en tiempo de creación del programa, el programador tiene que determinar qué variables son compartidas para que estas puedan ser conocidas por más de un proceso simultáneamente.

Es un método simple, pero obliga a los sistemas operativos a tener una porción de memoria asignada a una tarea, que no siempre es utilizada. Además, aumenta considerablemente la latencia en acceso a datos compartidos aunque estos sean accedidos siempre en lectura, con lo que nunca habría problemas de coherencia de caché.

Sistema de memoria no cachéable

Se basa en bloquear una porción de memoria compartida para que los datos almacenados en esta, nunca puedan estar replicados en cachés, de manera que siempre que el procesador solicita un dato de memoria perteneciente a este bloque, la caché siempre produce un fallo de caché.

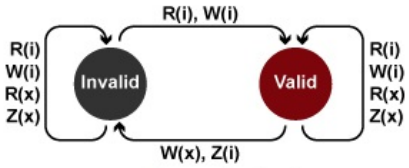
Variables compartidas

Al ser estos datos conocidos, el sistema operativo puede asignarle una posición en memoria determinada y perteneciente a la memoria no cacheable.

Protocolo snooping (o fisgoneo)

El protocolo snooping se basa en el principio de invalidación de copia de datos en cachés una vez que un procesador accede en modo escritura a un dato compartido. Se conoce como snooping porque está constantemente observando el bus de comunicación entre cachés y memoria compartida, para detectar cuando hay datos duplicados por las cachés de los procesos.
Hay dos variantes de este algoritmo, una para cachés con política de escritura con escritura inmediata y otra para cachés con escritura retardada.

Snooping para write through
Este protocolo es simple. La mejor manera de verlo, es como una máquina de estados con solo dos estados: válido e inválido.



Snooping para write through

Leyenda:
Acción:
- R=lectura
- W=escritura
- Z=reemplazo

Procesador:
- i=procesador actual
- x=demás procesadores

La caché almacena internamente un bit por línea que almacena este estado, y el algoritmo se ejecuta internamente en la caché. Cada caché tiene por tanto su propio algoritmo en ejecución, de manera que es ella misma la que escanea el bus, para actualizar el estado de cada línea.
Si analizamos el diagrama, vemos que el principio de funcionamiento es:
Una línea pasará de valida a inválido cuando un procesador distinto del propio accede a un dato en escritura.
Una línea pasara de inválida a válida cuando se accede a ella, bien en escritura o lectura.
Para los demás casos, en estado no variará.

1/3

Ejemplo

Supongamos un procesador multinúcleo con tres núcleos, con su correspondiente caché asociada a cada núcleo y una memoria compartida.

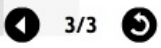
Supongamos además que dichos núcleos acceden de manera compartida a una dirección de memoria que llamaremos X según la siguiente secuencia.

Etapa	Acción	Memoria compartida	Caché procesador 1	Caché procesador 2	Caché procesador 3
1	V. iniciales	x=3	x=?, estado=I	x=?, estado=I	x=?, estado=I
2	P1:lee X	x=3	x=3, estado=V	x=?, estado=I	x=?, estado=I
3	P2:lee X	x=3	x=3, estado=V	x=3, estado=V	x=?, estado=I
4	P3:lee X	x=3	x=3, estado=V	x=3, estado=V	x=3, estado=V
5	P1:escribe X=9	x=9	x=9, estado=V	x=3, estado=I	x=3, estado=I
6	P3:escribe X=5	x=5	x=9, estado=I	x=3, estado=I	x=5, estado=V
7	P2:lee X	x=5	x=9, estado=I	x=5, estado=V	x=5, estado=V

2/3

Análisis

Etapa	Análisis
1	Partimos de una situación en la que solo hay un dato en memoria que es compartido por todos los procesadores.
2 3 4	En las etapas 2, 3 y 4 los tres procesadores replican dicho valor en sus respectivas cachés. Como todos los accesos son en lectura el valor de X queda en estado válido en todas ellas.
5	En la etapa 5, el procesador P1 accede a dicho valor en escritura. Como es una caché write through, esta actualización se refleja en la memoria principal. Al mismo tiempo las demás cachés detectan la actualización de este dato, por lo que ponen sus respectivas líneas en inválida.
6	En la etapa 6, es el procesador P3 el que actualiza el valor de X. Al igual que en el caso anterior, se actualiza la memoria principal y las cachés que tienen el dato como válido lo pasan a inválido.
7	Por último, en la fase 7, el procesador P2 lee el contenido de la variable X. Su caché detecta que ese dato esta invalidado, por lo que genera un fallo de caché y provoca que se traiga de nuevo el dato desde memoria, pasándose este a válido. En este caso, al ser un acceso en lectura, no invalida las demás cachés.



Snooping para write back

Para cachés con política write back, al no estar la memoria actualizada, este algoritmo tiene algunos inconvenientes. Por este motivo se utilizan otros métodos, el más utilizado es el construido a partir del autómata que puedes consultar en este enlace.

El principio de funcionamiento de este algoritmo es muy similar al anterior, diferenciándose en que cuando un proceso tiene una línea de caché en modo escritura, esta estará en estado RW, obligando a que todas las demás estén en inválido. Por el contrario, si ninguna línea tiene el dato en modo escritura, entonces todas las copias pueden estar en modo de RO.

Por ejemplo, supongamos un procesador multinúcleo con tres núcleos, con su correspondiente caché asociada a cada núcleo y una memoria compartida y supongamos que estos núcleos acceden de manera compartida a una dirección de memoria que llamaremos X, según la siguiente secuencia.

Haz clic sobre los enlaces en las diferentes etapas para acceder a un análisis del ejemplo.

Etapa	Acción	Memoria compartida	Caché procesador 1	Caché procesador 2	Caché procesador 3
Etapa 1	V. Iniciales	X=3	X=?, estado=INV	X=?, estado=INV	X=?, estado=INV
Etapa 2	P1: lee X	X=3	**X=3, estado=RO**	X=?, estado= INV	X=?, estado= INV
Etapa 3	P2: lee X	X=3	X=3, estado= RO	**X=3, estado=RO**	X=?, estado= INV
Etapa 4	P3: lee X	X=3	X=3, estado=RO	X=3, estado=RO	**X=3, estado=RO**
Etapa 5	P1: escribe X=9	X=3	**X=9, estado=RW**	**X=3, estado=INV**	**X=3, estado=INV**
Etapa 6	P3: escribe X=5	X=9	**X=9, estado=INV**	X=3, estado=INV	**X=5, estado=RW**
Etapa 7	P2: lee X	X=5	X=9, estado=INV	**X=5, estado=RO**	X=5, estado=RO

Etapa 1

Partimos de una situación en la que solo hay un dato en memoria que es compartido por todos los procesadores.

Etapas 2, 3 y 4

En las etapas 2, 3 y 4 los tres procesadores replican dicho valor en sus respectivas cachés. Como todos los accesos son de lectura, el valor de X queda en estado RO en todas ellas.

Etapa 5

En la etapa 5, el procesador P1 accede a dicho valor en escritura. Como es una caché write back, esta actualización no se refleja en la memoria principal, pero provoca que todas las cachés detecten el cambio de estado en el dato, por lo que ponen sus respectivas líneas en INV.

Etapa 6

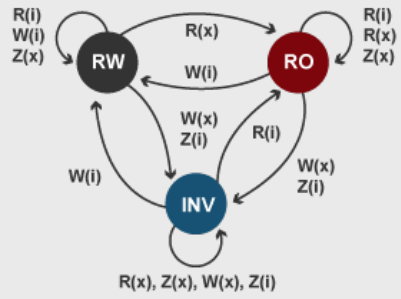
En la etapa 6 es el procesador P3 el que quiere actualizar el valor de X, al tenerlo en modo INV, genera previamente un fallo de caché para que la caché que se encuentre en estado RW actualice la memoria y el procesador P3 pueda disponer del valor correcto.

Una vez la memoria se ha actualizado, el procesador P3 actualiza su copia de caché, por lo que su estado pasa a RW, pero las cachés que tienen el dato en RO o RW lo pasan a INV.

Etapa 7

Por último, en la etapa 7, el procesador P2 lee el contenido de la variable X. Su caché detecta que ese dato esta INV, por lo que genera un fallo de caché y, posteriormente, notifica que se desea el valor del dato.
Esto provoca que la caché en estado RW actualice la memoria principal, causando que el dato se traiga de nuevo desde memoria y se pase a RO. En este caso, al ser una acceso en lectura, las cachés en estado RW pasan a RO, no a INV.

Autómata



Como se puede apreciar, ahora el autómata tiene tres estados distintos, nombrados:

- RO = ReadOnly
- RW = ReadWrite
- INV = INValido

El resto de la nomenclatura coincide con la del ejemplo del caso anterior.

Coherencia basada en directorios

Los protocolos de coherencia caché basados en directorios se utilizan en los **sistemas multiprocesadores lejanos**, construidos utilizando redes punto a punto o redes multietapa.

En tales arquitecturas, los procesadores están comunicados por redes, por lo que no existe un mecanismo adecuado de observación de todas las transacciones con la memoria. Además, no hay capacidad de broadcast, por lo que sería imposible implementar un protocolo snoopy.

Asimismo, en estas redes, los procedimientos de comunicación a todos los nodos es muy caro de implementar y mantener, por lo que los comandos de consistencia deberán ser sólo enviados a aquellas cachés que guarden una copia del bloque, lo que conduce a la necesidad de que exista información, sobre qué es lo que contienen otros nodos, almacenada en determinados nodos.

Esos almacenes son los directorios de información caché que utilizan estos protocolos y que les dan su nombre. Podemos distinguir dos tipos de esquemas.

Sistemas basados en directorio centralizado	Este contiene toda la información necesaria para mantener la consistencia. Es sencillo de implementar pero tiene el inconveniente de que el directorio central actúa como cuello de botella para los accesos.
Sistemas basados en directorios distribuidos	Cada módulo de memoria contiene un directorio separado que informa sobre dónde hay copias de los bloques de memoria que contiene, y sobre cuál es el estado de esas copias.

Los directorios de caché están formados por entradas, la memoria está dividida en bloques y cada bloque es lo que se puede compartir entre procesos. Por cada bloque de la memoria, hay una entrada en el directorio que contiene varios punteros que especifican en qué cachés locales están las copias de ese bloque. Además, cada directorio contiene un *bit de sucio* que especifica cuándo un solo procesador puede escribir sobre su copia local de ese bloque.

Independientemente de la información almacenada en los directorios caché, en cada caché se almacena información sobre el estado de los bloques en ella depositados.

Directorios de información caché

Los directorios caché se utilizan para almacenar información sobre dónde residen copias de los bloques caché. Los diferentes protocolos de este tipo difieren en qué información contienen esos directorios y cómo se mantiene la misma.

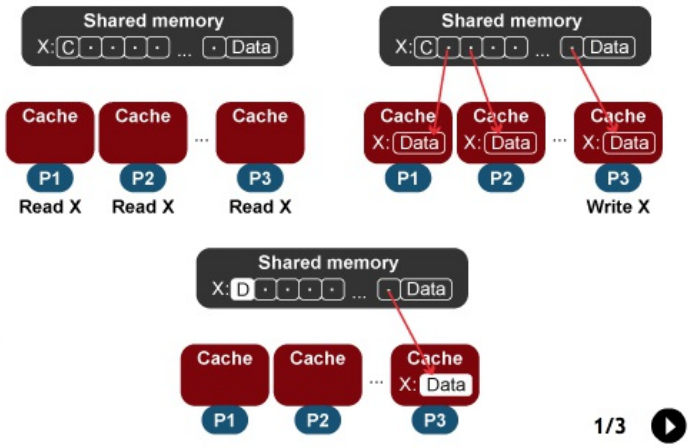
Modelos de directorios

Atendiendo a cómo se organizan internamente los directorios, hay tres grupos de protocolos basados en directorio: los basados en directorios full-map, los basados en directorios limitados y los basados en directorios encadenados.

Directorios full-map

Los directorios full-map se caracterizan porque cada entrada de directorio contiene N bits, siendo N el número de procesadores del sistema más un bit de sucio.

Así, para cada entrada hay N bits de presencia en procesador, uno por cada procesador del sistema, en los que se indica en qué procesadores hay copias válidas del bloque. Si el bit de sucio está activo, solo un bit de procesador estará en *presente*, lo que quiere decir que la posición de memoria no contiene información actualizada. Obsérvese que la cantidad de memoria consumida por el directorio es proporcional al tamaño de la memoria y al tamaño de cada entrada.



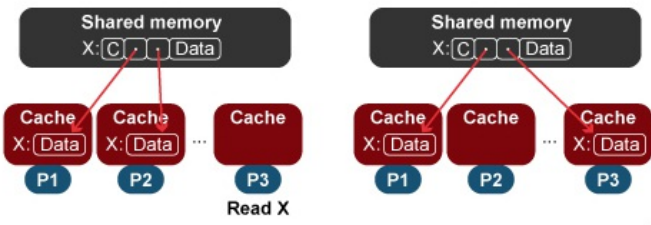
1/3

Directorios limitados

Los directorios limitados se diferencian de los full-map en que tienen un número fijo de bits por entrada, según el tamaño del sistema. Son una alternativa a los full-map para resolver el problema del tamaño de los directorios.

Puede surgir el problema de que se tengan más copias que bits para un bloque y, en ese caso, habrá que invalidar alguna de las copias, simplemente porque no se puede reflejar su presencia en el directorio.

Para proceder a ese reemplazamiento se establece una estrategia para determinar qué bloque debe ser candidato al desplazamiento.

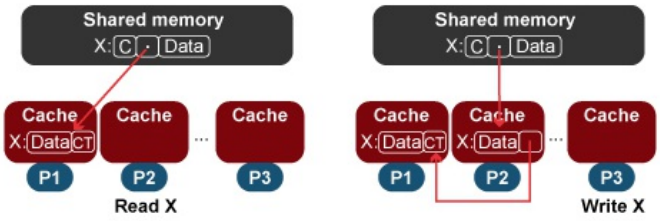


2/3

Directorios encadenados

Los directorios encadenados mantienen información para un número no limitado de copias por bloque, pero la mayor parte del directorio se distribuye dentro de las propias cachés locales de los procesadores. Las entradas de directorio se organizan en forma de lista enlazada.

La primera entrada de esta lista se encuentra en memoria principal y contiene un puntero que indica en qué caché está la primera copia. Esa copia, a su vez, contiene un puntero que indica en qué caché está la segunda copia y así sucesivamente.



3/3

Resumen

En este tema, hemos visto cuáles son los problemas de la jerarquía de memoria en sistemas multiprocesador o multinúcleo.

Hemos empezado exponiendo los problemas, para terminar revisando y estudiando cómo funcionan los algoritmos más utilizados para asegurar la coherencia de caché mediante el algoritmo de snoopy para cachés write through y write back.

Además hemos esbozado métodos para arquitecturas con mapas de memoria distribuidos, conocidos como protocolos basados en directorios.