Tema 9 - Cachés

Felipe Ortega, Gorka Guardiola

GSyC, ETSIT. URJC.

Sistemas Distribuidos (SD)

22 de diciembre, 2020

Contenidos

9.1 Coherencia

Coherencia de cachés

- Cachés HW (distribuidas, tejido de coherencia).
- Localidad.
- Problema de la coherencia.
- Referencia principal: Hennessy y Patterson [1].

Modelo de coherencia

- Garantías sobre cómo se sirven accesos concurrentes de la caché.
- Recordar que el compilador mete variables en registros, reordena y cambia operaciones (incluso elimina).

Consistencia sequencial (sequential consistency)

- Cada procesador manda peticiones en el orden de sus programas.
- Las peticiones de memoria se sirven de una cola FIFO global (para cada celda).
- Demasiado estricta (i.e. lenta).
- Se suele relajar (i.e. ir más rápido dando menos garantías).

Consistencia sequencial: relajación

- **Write-to-read**: *reads* se saltan *writes* (Sucede porque hay un *store buffer* o *write buffer*).
- *Writes* tienen que esperar por la línea de cache.
- Los *reads* pueden saltarse el *store buffer*.
- Esconde latencia de los *writes*.

Consistencia sequencial: relajación

- **Write-to-write**, ejecutan fuera de orden.
- Cache no bloqueante, *coalescing* escritura.
- *Writes* tienen que esperar por la línea de cache.
- La espera depende de cuantos haya esperando.

Coherencia

Memory Consistency in Modern Architectures

Reordered Memory Accesses	Read-to-Read	Read-to-Write	Write-to-Write	Write-to-Read	Atomic OPs and Reads	Atomic OPs and Writes
Alpha	Y	Y	Y	Y	Y	Y
AMD64	Y			Y		
IA64	Y	Y	Y	Y	Y	Y
PA-RISC	(Y)	(Y)	(Y)	(Y)		
POWER*	Y	Y	Y	Y	Y	Y
SPARC RMO	Y	Y	Y	Y	Y	Y
SPARC PSO			Y	Y		Y
SPARC TSO				Y		
IA32*	Y	Y		Y		

*write atomicity relaxed

Consistencia sequencial: relajación

- Mejor no depender de eso.
- Si no comparto memoria, no me importa. Pero si la comparto, entonces:
  - Utilizar barreras de memoria (`lfence` , `sfence`, `mfence`).
  - Todas las operaciones implicadas (`load`, `store`, todas), se ven globalmente después de la barrera.
  - Utilizar instrucciones atómicas que dan garantías (XCHL) o a más alto nivel: cierres, semáforos, canales.

9.2 Protocolos de coherencia

Protocolos

- *Directory-based*, directorio central.
- *Snooping* (requiere bus compartido o radiado) vigila direcciones y tipo de operación para invalidar.
- *Snarfing* (requiere bus compartido o radiado) , vigilar datos y direcciones para actualizarse.

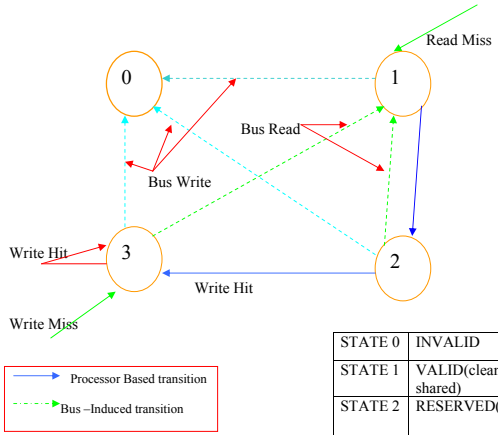
Snooping

- Consiste en espiar el bus, vigilando las operaciones de unas cachés que afectan al contenido de otras.
- Dos opciones:
  - *Write invalidate*.
  - *Write update*.
- Veamos cada una de ellas.

Write invalidate

- Es el protocolo que ofrece mejor rendimiento y según [1] el utilizado por *todos* los multiprocesadores modernos.
- La cache radia la orden *invalidar* para obtener la propiedad del bloque.
- Cualquier copia del bloque en otras cachés deja de tener validez (no está actualizada).
- Posibles estados de un bloque:
  - `INVALID`: la cache no tiene los datos actuales;
  - `VALID`: la cache tiene datos pero puede estar en otras caches, en memoria;
  - `RESERVED`: la cache tiene los datos y está sólo en cache y en mem;
  - `DIRTY`: sólo esta cache, no en mem. tiene datos válidos.

Coherencia



STATE 0	INVALID
STATE 1	VALID(clean, potentially shared)
STATE 2	RESERVED(clean, only copy)
STATE 3	DIRTY(modified, only copy)

Write invalidate

- **Read hit**: se devuelven datos.
- **Read miss**: se carga el bloque en estado VALID si no hay ninguna otra copia a `DIRTY`. De lo contrario, la caché que tiene la copia actualizada se lo da, escribe en memoria y todos pasan a VALID.
- Depende de si implementa *write back* o *write through*.
  - En *write back*, se mira (***snoop***) directamente en la caché. Si hay que enviar copia actualizada desde otra caché, se hace directamente sin pasar por memoria.
  - En *write through*, la prioridad es que la memoria se mantiene actualizada. Cualquier caché solo puede leer de memoria para resolver un fallo.

Write invalidate

- **Write hit**:
  - `RESERVED/DIRTY`, se escribe directamente;
  - `VALID`, se escribe y cambia a RESERVED, los demás cambian a `INVALID`, escribir de forma repetida, una escritura (*write once protocol*).
- **Write miss**:
  - Si no hay *dirty bit* activado, el bloque se carga de memoria y queda `DIRTY`.
  - Si el bloque está a `DIRTY` en otra caché, se lo da a esta caché (la que ha solicitado *write*) y se pone en la otra caché a `INVALID`. El procesador que pide el bloque toma su propiedad. Una vez cargado, se pone a `DIRTY` tras el *write* en esta caché.
- **Reemplazo**: Sólo hay que escribir los `DIRTY`; si no, se reemplazan.

Write update

- Ref: https://en.wikipedia.org/wiki/Firefly_(cache_coherence_protocol).
- Radia los updates. Posibles estados de un bloque:
  - `VALID-EXCLUSIVE`: en caché y en memoria;
  - `SHARED`: en caché, en otras cachés y en memoria;
  - `DIRTY`: solo en cache y no en memoria;
- Bus, línea compartida (*shared line*) indica si otra cache tiene copia del bloque.
- La cache activa la línea compartida en *read* o *write* si tiene copia para avisar.

Comparación

- Se suele usar *write invalidate*.
- *Write-invalidate* realiza un solo *write* para varios; con *write update* necesito hacer varios *broadcasts*.
- *Write-invalidate* se aprovecha de los principios de localidad.
- *Write-update* tiene menos latencia entre *read* y *write*.
- *Write-update* usa más ancho de banda.

9.3 Protocolos avanzados

MESI

- Usado por **Intel**. Acrónimo formado por posibles estados.
- Combinación de varias cosas, un poco más complicado.
- **Modificado**: bloque en esta caché solamente.
- **Exclusive**: bloque en esta cache y en memoria.
- **Shared**: bloque en varias caches y en memoria.
- **Invalid**: bloque no está en memoria.
- *Snoop* + Request for Ownership (RFO).

MOESI

- Usado por AMD64 (AMD, no Intel).
- Añade estado **owned**: en cache y no en memoria (como *shared*, pero sin memoria).
- Mejor para que las cachés se pasen datos unas a otras.

MOESI

- Barreras de memoria, operaciones atómicas, etc.
- Recordar: pocas garantías para cosas compartidas.

Sistemas de memoria vs. Sistemas distribuidos

- Comunicaciones fiables vs. no fiables.
- *Snoop* implica *broadcast* (no siempre posible o eficiente).
- Bloquearse esperando (*lock*), es factible (aunque ineficiente) en un procesador, no en un sistema distribuido.
- Poco a poco convergen, aunque la latencia limita más siempre los sistemas distribuidos (velocidad luz).
- El sistema distribuido con cachés más común es un sistema de ficheros o similar (HDFS en Hadoop, etc.).

Semántica de ficheros compartidos en sistemas distribuidos

- Ya lo hemos visto, es similar a un modelo de coherencia (garantías).
- Semántica UNIX: cada operación se ve inmediatamente.
- Semántica de sesión: cuando se cierra el fichero.
- Ficheros inmutables: cada cambio crea una versión (no se puede modificar).
  - Útil para sistemas de procesamiento de datos *streaming*.
- Transacciones: cada operación es una transacción, pueden agruparse.

Bibliografía I


[1] Hennessy, John L., and Patterson, David A.
Computer architecture: A quantitative approach. 5th Ed.
Morgan Kaufmann, 2011.


[2] van Steen, M., Tanenbaum, A. S.
Distributed Systems.
Third Edition, version 01. 2017.