Tema 9 - Cachés

Felipe Ortega, Gorka Guardiola

GSyC, ETSIT. URJC.

Sistemas Distribuidos (SD)

3 de diciembre, 2019

Contenidos

9.1 Coherencia

Coherencia de cachés

- Cachés HW (distribuidas, tejido de coherencia).
- Localidad.
- Problema de la coherencia.
- Referencia principal: Henessy 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, si comparto entonces:
  - Utilizar barreras de memoria (lfence, sfence, mfence).
  - Todas las operaciones implicadas (load, store, todas), se ven globalmente despues de la barrera.
  - Utilizar instrucciones atómicas que dan garantías (XCHL) o 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.

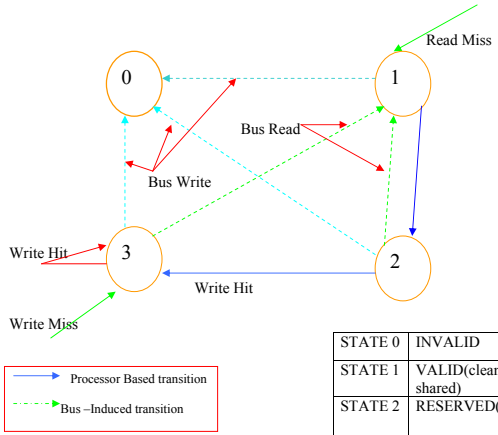
Snooping

- Espiar el bus; dos opciones:
- *write invalidate*;
- *write update*.
- Veamos cada una de ellas.

Write invalidate

- La cache radia invalidar para obtener la propiedad del bloque.
- 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)

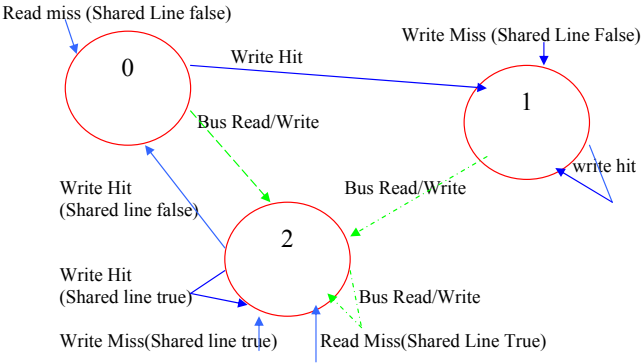
Write invalidate

- **Read hit**: se devuelven datos.
- **Read miss**: se carga en VALID si no hay ningún otro `DIRTY`, si no, se lo da, escribe en memoria y todos VALID, depende de write through, write back. Write through, la memoria se mantiene actualizada. En write back, se mira (***snoop***) directamente en la cache.

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**: no hay dirty bit, el bloque se carga de memoria, `DIRTY`. Otro bloque `DIRTY`, se lo da a esta caché y se pone `INVALID`. El procesador que lo pide, toma propiedad. Una vez cargado se pone a `DIRTY`.
- **Reemplazo**: Sólo hay que escribir los `DIRTY`; si no, se reemplazan.

Write update (firefly)



Write update

- Radia los updates. Posibles estados de un bloque:
- `VALID-EXCLUSIVE` cache y memoria;
- `SHARED` cache, otras caches, en memoria;
- `DIRTY` cache sólo y no memoria;
- Bus, línea compartida (Shared line) diciendo si la cache tienen copia.
- La cache levanta la línea compartida en read o write si tiene copia.

Write update

- **Read hit**: datos sin esperar.
- **Read miss**: se carga en la cache como VALID-EXCLUSIVE o SHARED dependiendo de si la Shared-line está arriba. Si otra cache tiene el bloque, se lo da y levanta SL. Todas las otras caches paran su intento de dar los datos. Si el bloque se lo da otra cache, todas se actualizan.

Write update

- **Write hit**: VALID-EXCLUSIVE, pasa a DIRTY sin esperar. SHARED y SL down, pasa a VALID-EXCLUSIVE y coge los datos, SHARED y SL up, se cambia y se radian los cambios. DIRTY, carga datos y no sincroniza la memoria.
- **Write miss**: SHARED, se escribe a memoria, si SL, sigue igual despues de escribir a memoria (y otra cache). Si DIRTY y SL es falso, entonces cambia a VALID-EXCLUSIVE.

Comparación

- Se suele usar *write invalidate*.
- Write-invalidate, un write para varios, write update varios broadcasts.
- Write-invalidate se aprovecha de la 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**.
- 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).
- Owned: en cache y no en memoria (como shared pero sin memoria).
- Mejor para que las caches 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 S. Dist.
- Poco a poco convergen aunque la latencia limita más siempre los S. Dist. (velocidad luz).
- El sistema distribuido con cachés más común es un sistema de ficheros o similar (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).
- Transacciones: cada operación es una transacción, pueden agruparse.

Bibliografía I


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


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