Protocolos de caché para sistemas Symmetric MultiProcessor

Melina Belén Jáuregui, Lucas Ruiz Abelleira, Dalmiro Vilaplana,
estudiantes, Facultad de Ingeniería de la Universidad de Buenos Aires,

Abstract—El presente informe se enfoca en el estudio del funcionamiento de las cachés privadas en sistemas Symmetric MultiProcessor (SMP). Se realiza un análisis detallado de los diversos protocolos de coherencia caché, con el objetivo de evaluar cómo cada uno de ellos aborda los desafíos inherentes a la consistencia de la memoria en diversos escenarios. Además, se lleva a cabo una investigación sobre el impacto de estos protocolos en términos de eficiencia y rendimiento, con el propósito de examinar detalladamente las implicaciones que cada uno de ellos tiene en el desempeño global del sistema.

Index Terms—Protocolos caché, Protocolo Snoop, Write-Update, Write-Invalidate, Firefly Protocol, Dragon Protocol, MSI Protocol, MESI Protocol, MOSI Protocol, MOESI Protocol.

1 Introducción

En un panorama tecnológico en constante evolución, los desafíos para el avance de la memoria se vuelven cada vez más significativos, lo que a su vez limita el progreso de la velocidad de los procesadores y la capacidad de almacenamiento.

Según estimaciones, el rendimiento de los procesadores se ha venido incrementando aproximadamente un 60% cada año debido a la reducción del tiempo de ciclo de reloj y al incremento del número de instrucciones ejecutadas por ciclo (IPC). Sin embargo, el tiempo de acceso a las memorias DRAMS sólo mejora un 10% por año, aunque la capacidad se duplica cada año y medio, según la Ley de Moore [1].

Sin embargo, ha surgido una solución que impulsa el rendimiento de los sistemas informáticos: la implementación de sistemas multiprocesadores con caches privadas.

Mientras los avances en velocidad de la memoria se han vuelto cada vez más desafiantes, estos sistemas aprovechan al máximo los recursos disponibles al introducir una capa de almacenamiento temporal y privada entre los procesadores y la memoria principal.

No obstante, esta solución presenta un inconveniente significativo: la posibilidad de que múltiples copias del mismo dato coexistan en diferentes cachés de forma simultánea. Si cada procesador actualiza su propia copia, y este suceso no es informado hacia los demás, se genera una visión inconsistente de la memoria [4].

Por consiguiente, debe existir un mecanismo que evite la existencia de múltiples réplicas diferentes del mismo bloque de datos en diversas cachés privadas. En el presente informe, se presenta un análisis específico sobre el funcionamiento y los efectos de la coherencia de caché en sistemas multiprocesador.

2 Coherencia de caché

La coherencia de caché es el término que se utiliza para referirse a la consistencia y sincronización de los datos almacenados en diferentes cachés dentro de un sistema multiprocesador o multicore. Esta asegura que todos los procesadores observen una vista consistente de la memoria compartida, evitando inconsistencias de datos y garantizando una ejecución confiable del programa [2].

Cuando se realizan modificaciones en un bloque de memoria compartido por varias cachés, surge la necesidad de propagar esta alteración a cada caché que contenía el mismo dato. Si no se la mantiene, puede producirse corrupción de datos y un comportamiento incorrecto del programa.

Para abordar el problema de coherencia de caché, existen diversas soluciones tanto a nivel de hardware como a nivel de software. Las soluciones más ampliamente utilizadas son las de carácter hardware ya que son transparentes para el programador y el compilador. Esto es beneficioso porque no se requiere preocuparse por aspectos específicos de la caché ni modificar el código para mantener la coherencia.

En los esquemas de hardware, se pueden identificar dos categorías principales [4]:

- Protocolos Snoop
- Protocolos Directorio

En este informe se desarrolla en profundidad los protocolos Snoop que proveen reconocimiento en tiempo de ejecución de potenciales inconsistencias [2].

3 Protocolos Snoop

Estos protocolos distribuyen la responsabilidad de mantener la coherencia de caché entre todos los controladores de caché del sistema. Una caché debe reconocer cuando una

Ingeniería en Informática Universidad de Buenos Aires, mail: mjauregui@fi.uba.ar
Licenciatura en Análisis de Sistemas Universidad de Buenos Aires, mail: lruiz@fi.uba.ar
Licenciatura en Análisis de Sistemas Universidad de Buenos Aires, mail: dvilaplana@fi.uba.ar

línea que posee es compartida con otras caches. Para ello, estos protocolos marcan el estado de cada línea de caché utilizando 2 o 3 bits del tag dependiendo de la cantidad de estados que cada protocolo disponga. Cuando se actualiza una línea, se lo anuncia a todas las demás caches mediante un mecanismo de broadcasting [2].

El mecanismo de broadcasting es una técnica utilizada para mantener la coherencia de la caché en sistemas multiprocesador o multi-núcleo. Se basa en la propagación de mensajes a través del bus de comunicación hacia todas las cachés, con el propósito de informar el estado actualizado de un bloque de memoria [4].

A partir de esta estrategia, cada controlador de caché puede "hurgar" (snoop) en la red para observar estas notificaciones y actuar en consecuencia. Se han explorado dos enfoques:

- Write Update
- Write Invalidate

A continuación, se presentarán en detalle cada protocolo Snoop con sus respectivas variantes.

3.1 Write Update

El enfoque "Write-Update" se basa en la idea de actualizar el estado de un bloque de caché por cada actualización del mismo. Por ello mismo, pueden haber múltiples escritores y lectores. Cuando un procesador quiere actualizar una línea compartida, la palabra a ser actualizada es distribuída hacia las demás cachés. Esto implica que estas últimas únicamente modifiquen las palabras individuales alteradas por el procesador, sin requerir la actualización de todo el bloque de memoria compartida. [13].

Cuando dos cachés intentan escribir simultáneamente en el mismo bloque de memoria, puede producirse una condición conocida como "escritura simultánea" o "escritura en paralelo". Esta situación puede generar problemas de coherencia de caché si no se maneja adecuadamente [14].

Un enfoque común utilizado para manejar la escritura simultánea en sistemas multiprocesador es el protocolo de escritura serializada, el cual utiliza algún mecanismo de arbitraje para determinar cuál de las cachés tiene el derecho de realizar la escritura. Esto puede hacerse asignando prioridades a las cachés o utilizando algún otro esquema de arbitraje justo [3].

A continuación se explicarán dos variantes más utilizadas de este enfoque, Protocolo Firefly y Protocolo Dragon.

3.1.1 Protocolo Firefly

El protocolo Firefly es un mecanismo de coherencia caché que fue desarrollado por DEC Systems Research Center. Este esquema presenta tres posibles estados los cuales pueden ser asignados a cada linea [10]. Estos son:

- Exclusivo (E): solo una caché tiene la línea y la memoria está actualizada.
- Compartido-Limpio (Sc): múltiples cachés pueden tener la línea y la memoria está actualizada.
- Modificado (M): solo una caché tiene la línea y esta es dirty. Cuando una linea de caché tiene asignado este estado, la caché local es propietaria de los datos por lo que debe actualizar la memoria al reemplazar.

Con este protocolo, cuando un procesador desea realizar una operación de **lectura** en una dirección de memoria específica, se siguen los siguientes pasos:

1) Bloque se encuentra en la caché local: En este escenario el procesador puede realizar una lectura directamente desde su caché.
2) Bloque no se encuentra en la caché local: En este caso, se debe proceder como:
    a) Si alguna otra caché contiene el bloque y su estado es Compartido-Limpio, luego se realiza una copia del bloque en la caché actual y se marca como estado Compartido-Limpio.
    b) Si alguna otra caché contiene el bloque y su estado es Exclusivo, luego se realiza una copia del bloque en la caché actual y se marcan ambas cachés como estado "Compartido-Limpio".
    c) Si alguna otra caché contiene el bloque y su estado es Modificado, luego se actualiza el bloque en la memoria principal y se establece la línea en el estado Exclusivo. Cuando se haya completado la actualización, recién ahí la caché local vuelve a solicitar el bloque por lo que estará frente al escenario 2b.
    d) Si ninguna caché contiene el dato se lo debe buscar en memoria.

Por otro lado, cuando un procesador desea realizar una operación de **escritura** en una dirección de memoria específica, se llevan a cabo los siguientes pasos:

1) Bloque se encuentra en la caché:
    a) Bloque se encuentra en estado "Modificado" o "Exclusivo": En este escenario el procesador tiene la autoridad para escribir directamente en la caché y marcar el bloque como Modificado.
    b) Bloque se encuentra en estado Compartido-Limpio: En este caso se va a requerir actualizar la memoria principal. Como se mencionó en la sección III, las demás cachés van a estar monitoreando el bus, por lo que esta actualización va a ser percibida por las mismas (mecanismo broadcasting). Cada una va a inspeccionar su estado interno y, en caso de que almacenen el mismo bloque, procederán a actualizarlo.
2) Bloque no se encuentra en la caché:
    a) Ninguna otra caché lo tiene: En este escenario, primero se obtendrá el bloque desde la memoria principal. Concretado este paso (y habiéndolo actualizado), se establecerá el estado de la línea local en Modificado.
    b) Otras cachés lo tienen en estado Compartido-Limpio: En este caso se debe actualizar el estado de la línea local en Compartido-Limpio.

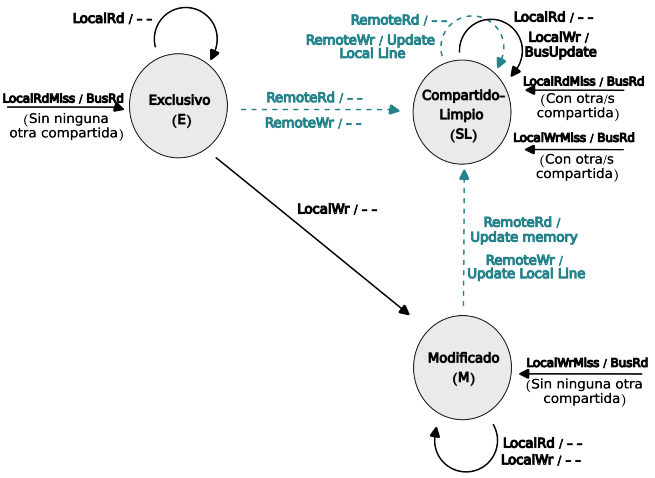

Fig. 1. Diagrama de transición de estados protocolo Firefly [10].

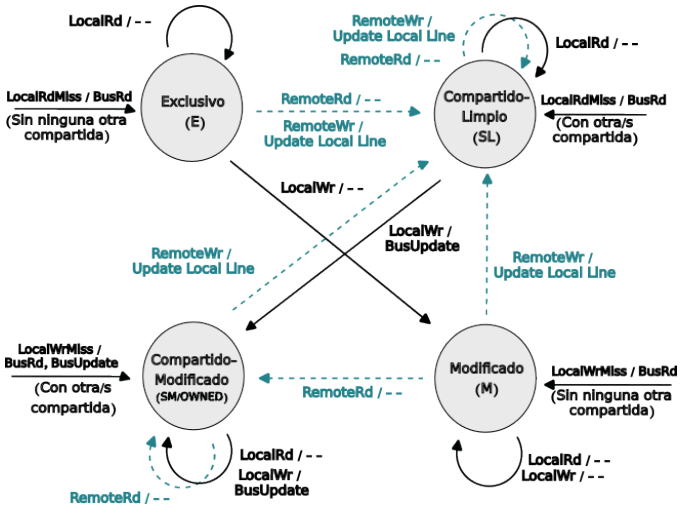

Fig. 2. Diagrama de transición de estados protocolo Dragon. [12]

Consecuentemente, se procederá como el escenario Escritura 1b.

En la Figura 1 se puede ver detalladamente la transición de estados dependiendo de cada evento.

3.1.2 Protocolo Dragon

Este esquema se distingue como una variante de mayor complejidad del protocolo Firefly, con la adición de un estado denominado Compartido-Modificado.

Con esta implementación, la definición de los estados Compartido se modifican a:

- Compartido-Limpio (Sc): múltiples cachés pueden tener la línea y la memoria puede o no estar actualizada. No es la propietaria de los datos, por lo que al desalojar el bloque, no se debe actualizar la memoria principal.
- Compartido-Modificado (Sm): múltiples cachés pueden tener la línea y la memoria no está actualizada. Solo una caché puede estar en este estado para una línea dada (pero otras pueden estar en estado Sc). En este estado, la caché local es propietaria de los datos por lo que debe actualizar la memoria al reemplazar.

Con la adición de este nuevo estado, **no es necesario actualizar la memoria cada vez que se escribe un bloque que está compartido en otras cachés**. Solamente se debe establecer el estado de la caché local en Compartido-Modificado y, en las demás cachés que comparten el bloque de memoria, en Compartido-Limpio [11].

En la Figura 2 se puede ver detalladamente la transición de estados dependiendo de cada evento.

3.2 Write Invalidate

En este enfoque, a diferencia de Write-Update, pueden haber varios lectores pero un solo escritor en un momento dado. Cuando una caché quiere realizar una escritura a cualquiera de las palabras en un bloque específico, primero envía un mensaje que invalida ese bloque en las demás caches, haciendo que la línea sea exclusiva para la caché de escritura [13].

A continuacion se explicarán cuatro variantes más utilizadas de este enfoque: MESI, MSI, MOESI y MOSI.

3.2.1 Protocolo MESI

El protocolo MESI es un enfoque ampliamente utilizado para mantener la coherencia de caché en sistemas multiprocesador. El término MESI es un acrónimo que representa los cuatro posibles estados en los que puede encontrarse un bloque de memoria en una caché: Modificado (M), Exclusivo (E), Compartido-Limpio (Sc) e Invalido (I) [9].

Tanto el estado ”Modificado” como el ”Exclusivo” comparten el mismo concepto explicado previamente en los protocolos de Write-Update, por lo tanto, nos enfocaremos en explicar de manera detallada los estados ”Compartido-Limpio” (Sc) e ”Inválido” (I).

- Compartido-Limpio (Sc): indica que esta línea de caché podría estar presente en otras cachés del sistema y ambas copias están en su versión actual. La memoria principal está actualizada.
- Inválido: El estado ”Inválido” indica que los datos de la línea de caché no son válidos. Esto significa que los datos buscados no se encuentran en la caché, o que la copia local de la caché no es correcta, debido a que otro procesador ha actualizado la posición de memoria correspondiente.

El propósito de este último estado en los protocolos Write-Invalidate radica en evitar la necesidad de actualizar las demás cachés que contienen el mismo bloque en cada operación de escritura, a diferencia de lo que ocurre en los protocolos Snoop Write-Update. En ciertos aspectos, el protocolo MESI es la versión de Write-Invalidate del protocolo Firefly. Esta afirmación se sustenta en la existencia de similitudes en los procesos de lectura y escritura entre ambos protocolos. Sin embargo, a pesar de estas similitudes, el proceso de escritura presenta diferencias significativas debido a la adopción del enfoque de Write-Invalidate. Por lo tanto, es pertinente detallar su funcionamiento de manera precisa [3].

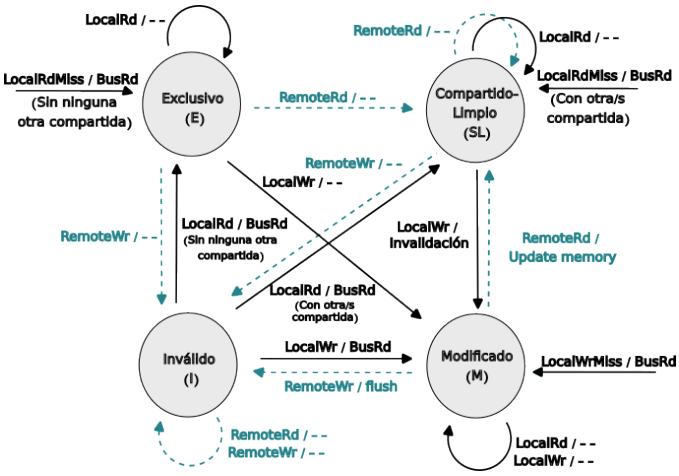

Fig. 3. Diagrama de transición de estados protocolo MESI [11].

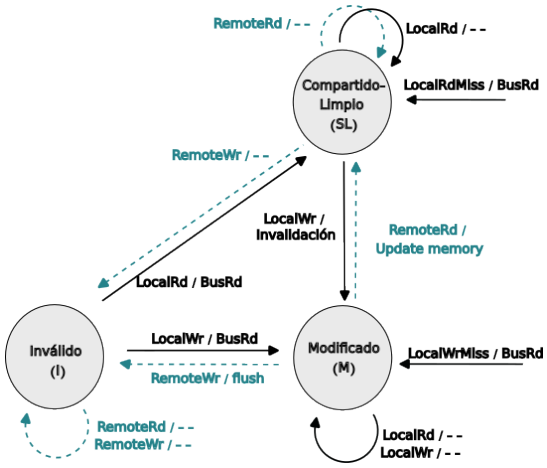

Fig. 4. Diagrama de transición de estados protocolo MSI [6]

.

Cuando un procesador desea realizar una operación de **escritura** en una dirección de memoria específica, se llevan a cabo los siguientes pasos:

1) Bloque se encuentra en la caché:
   a) El bloque se encuentra en estado Modificado o Exclusivo: En este escenario el procesador tiene la autoridad para escribir directamente en la caché y marcar el bloque como Modificado.
   b) El bloque se encuentra en estado Compartido-Limpio: En contraste con los protocolos Write-Update, donde se actualizaban todas las cachés que compartían el mismo bloque, el enfoque de Write-Invalidate se basa en notificar la invalidación del bloque mediante el bus. Dado que cada caché monitorea constantemente el bus, estas invalidaciones son percibidas por todas las cachés. En consecuencia, cada caché realizará una inspección de su estado interno y, en caso de almacenar el mismo bloque, se le establecerá el estado de Inválido. Además, a la caché local se le asigna el estado Modificado.
2) Bloque no se encuentra en la caché:
   a) El bloque no se encuentra en ninguna otra caché: En este escenario se actualiza la caché local y se establece el estado de la línea en Modificado.
   b) Otras cachés lo tienen en estado Compartido-Limpio: En este caso se debe actualizar el estado de esta línea local en Compartido-Limpio. Consecuentemente, se procederá como escenario Escritura 1b.

En la Figura 3 se puede ver detalladamente la transición de estados dependiendo de cada evento.

3.2.2 Protocolo MSI

El protocolo MSI se presenta como una versión simplificada del protocolo MESI, ya que consta de tres estados: Modificado, Compartido-Limpio e Inválido [4]. A diferencia del protocolo MESI, el estado Compartido-Limpio en MSI no garantiza la presencia de múltiples copias del bloque, pudiendo existir solamente una copia [5] . En este sentido, se puede considerar que el estado Compartido-Limpio engloba la funcionalidad del estado Exclusivo del protocolo MESI.

En la Figura 4 se puede ver detalladamente la transición de estados dependiendo de cada evento.

3.2.3 Protocolo MOESI

El protocolo MOESI es un enfoque de Write-Invalidate con cinco estados: Modificado, Exclusivo, Inválido, Compartido-Modificado, Compartido-Limpio [10]. Al comparar el protocolo MOESI con el protocolos anteriormente mencionados, se observa que es una versión más sofisticada del protocolo MESI, incorporando el estado de Compartido-Modificado (también conocido como Owned en inglés) [4]. Como resultado, MOESI combina elementos tanto del protocolo MESI como el de Dragon. Por una parte, al igual que en el protocolo de Write-Update, no se hace necesaria la actualización de la memoria en cada operación de escritura. Por otra parte, siguiendo el enfoque de Write-Invalidate, se evita la necesidad de actualizar todas las cachés en cada operación de actualización, manteniendo así la línea de acción del protocolo MESI.

En la Figura 5 se puede ver detalladamente la transición de estados dependiendo de cada evento.

3.2.4 Protocolo MOSI

El protocolo MOSI se presenta como una versión que combina los protocolos MOESI y MSI. Este consta de cuatro estados: Modificado (M), Compartido-Modificado (Owned) Compartido-Limpio (Shared) e Inválido (Invalid) [4]. Al igual que el protocolo MSI, el estado Compartido-Limpio no garantiza la presencia de múltiples copias del bloque, pudiendo existir solamente una copia. Por otra parte, debido a la inclusión del estado Compartido-Modificado (Owned), no se hace necesaria la actualización de la memoria en cada operación de escritura.

En la Figura 6 se puede ver detalladamente la transición de estados dependiendo de cada evento.

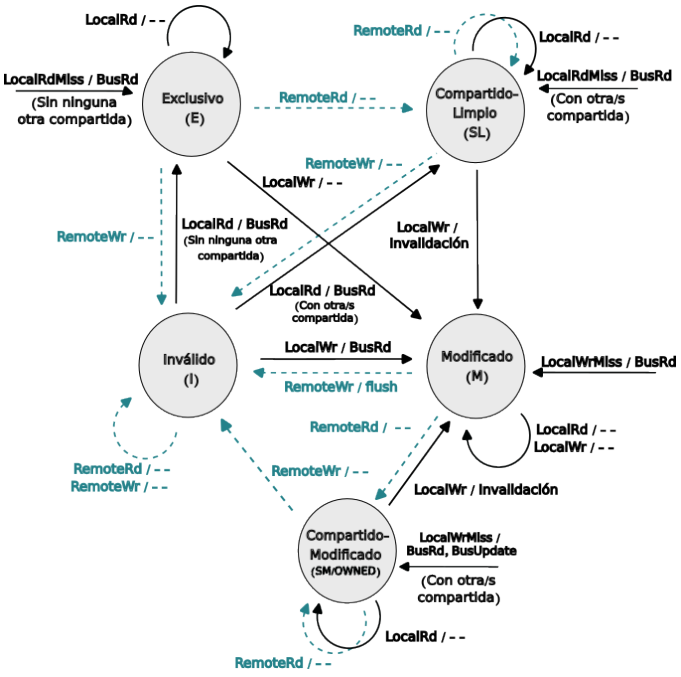

Fig. 5. Diagrama de transición de estados protocolo MOESI [12].

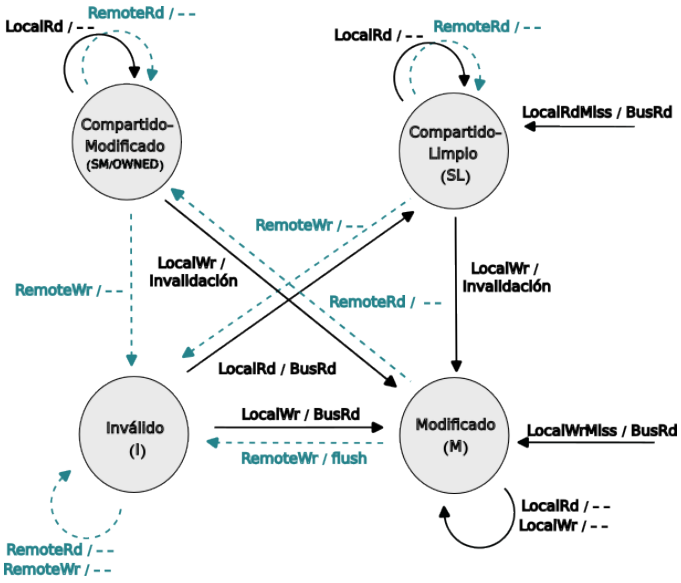

Fig. 6. Diagrama de transición de estados protocolo MOSI [12].

4 Influencia de los estados

4.1 Incorporación del estado Exclusivo (E)

El estado Exclusivo en la caché brinda beneficios significativos en las operaciones de escritura. Al encontrarse en este estado, un procesador puede escribir directamente en el bloque dentro de su propia caché, sin necesidad de invalidar o notificar a otras cachés, lo que evita la sobrecarga de mensajes de invalidación y garantiza un proceso de escritura más eficiente y rápido. En consecuencia, el estado Exclusivo se destaca por reducir la cantidad de transacciones de monitoreo de bus requeridas, lo que se traduce en una mejora notable en la eficiencia del sistema de caché [12].

Los beneficios del estado Exclusivo toman relevancia cuando se lo utiliza en aplicaciones que tienen un alto grado de lectura y acceso secuencial a datos compartidos [5].

A modo de ilustración, consideremos un sistema multiprocesador en el cual múltiples procesadores trabajan en una tarea de procesamiento de imágenes y cada uno tiene asignada una región específica de la imagen para procesar.

Si la coherencia caché está implementada con el estado Exclusivo, luego, cuando un procesador lee un bloque de memoria por primera vez, se lo debe establecer en ese mismo estado. Esto significa que el procesador tiene la única copia válida y no modificada del bloque en su caché.

A medida que el procesador trabaja en su región de la imagen, puede acceder repetidamente a los datos compartidos adyacentes, como los píxeles vecinos.

Dado que el procesador tiene el estado Exclusivo, puede leer o modificar estos datos directamente desde su caché sin necesidad de invalidar o comunicarse con otras cachés.

Sin este estado, tal como se aplica en el protocolo MSI o MOSI, implica cargar inicialmente la línea de caché utilizando un BusRd, pasar al estado Sc y luego realizar un BusRdX (verificación de exclusividad del bloque) para pasar al estado M. Esto requiere dos transacciones de monitoreo de bus. Por otro lado, en el caso de MESI, esto se puede lograr en una sola transacción. La caché realizaría un BusRd, pasaría al estado E y, dado que ninguna otra caché tiene la línea de caché, no sería necesario realizar una transacción BusRdX para pasar al estado M. Simplemente se podría cambiar discretamente el estado de la línea de caché a Modificado.

4.2 Incorporación del estado Compartido-Modificado (Sm/Owned)

La intención de agregar el estado Compartido-Modificado (Sm/Owned) es reducir la cantidad de escrituras de memoria requeridas por el protocolo.

Consideremos el escenario en donde una caché realiza una operación de escritura en un bloque que no es compartido por otras cachés. En todos los protocolos, su estado se actualiza a Modificado.

Luego, tanto en MSI, MESI y Firefly, cuando otra caché necesita acceder al dato actualizado, se requiere realizar una escritura en memoria y, una vez completada, ambas cachés establecen el estado del bloque como Compartido-Limpio (Sc).

En contraste, al considerar los protocolos MOESI o Dragon, la necesidad de actualizar la memoria principal para permitir el acceso simultáneo en múltiples cachés a un bloque de datos que fue modificado, no es requerida. Esto se debe a la inclusión del estado Compartido-Modificado (Owned), el cual asegura la consistencia adecuada de la memoria en todas las cachés [12].

Únicamente cuando se desee reemplazar una línea de caché en los estados Modificado o Compartido-Modificado, con el fin de garantizar la corrección, se debe realizar la escritura correspondiente en la memoria [5].

Esta estrategia reduce la latencia de acceso a la memoria y toma relevancia cuando el mismo bloque es modificado y accedido por diferentes cachés constantemente, haciendo estas operaciones de lectura y escritura de manera mucho más eficiente.

5 Write-Invalidate vs Write-Update

Ambos enfoques tienen sus ventajas y desventajas dependiendo de la forma en que se establezca la interacción entre las cachés y de cómo se gestione el manejo de datos en memoria.

Hay que tener en cuenta que, en términos de nivel de operación, Write Invalidate trabaja a nivel de bloques de caché. Esto implica que la escritura a cualquiera de las palabras en un bloque específico necesita generar una invalidación total del bloque en las demás cachés. Por otro lado, Write Update opera a nivel de palabras individuales en bloques de caché con múltiples palabras. Esto implica que la actualización se realiza en el nivel de palabra modificada en lugar de actualizar todo el bloque de memoria compartida. De esta manera, solo se envían y actualizan los valores de las palabras del bloque que han sido modificadas por el procesador.

Por consiguiente, el enfoque Write-Invalidate se destaca por su mayor eficiencia en las operaciones de escritura. Entre las ventajas asociadas a este enfoque se pueden mencionar:

- Menor tráfico en el bus: En comparación con el enfoque Write-Update, Write-Invalidate genera menos tráfico en el bus de interconexión. Esto se debe a que solo se envían señales de invalidación a las cachés que tienen copias válidas del bloque, en lugar de enviar los nuevos valores del bloque a todas las cachés. Esto puede ser beneficioso en sistemas con limitaciones de ancho de banda del bus [10].
- Menor latencia de escritura: En el enfoque Write-Invalidate, después de enviar una señal de invalidación, el procesador puede realizar la escritura directamente en el bloque de memoria. Para poder continuar con la siguiente operación luego de hacer la escritura, no es necesario esperar a que todas las demás cachés actualicen sus copias del bloque (solo la tienen que invalidar) [8].
- Solución al problema de coherencia caché cuando se presenta: En el caso del Write-Invalidate, se proporcionan los datos actualizados al procesador recién cuando los solicita, resultando en una ventaja si la tasa de solicitud del mismo bloque de memoria en otras cachés es baja. En contraste, el enfoque Write-Update actualiza todas las copias del bloque en las cachés, incluso si no son utilizadas posteriormente por otros procesadores, lo que puede resultar en actualizaciones innecesarias y redundantes en el sistema de caché [12].

Por otro lado, el protocolo Write-Update presenta la ventaja de permitir una lectura más rápida de los datos, debido a que las actualizaciones realizadas por un procesador son propagadas inmediatamente a otras cachés. Esto significa que los datos modificados se encuentran actualizados y disponibles para su lectura de manera inmediata en todas las cachés que comparten el mismo bloque de memoria.

A diferencia de este último, el protocolo Write Invalidate tiende a requerir más tiempo para leer los datos. Esto se debe a que cualquier lectura a un bloque de memoria inválido, requiere obtener una nueva copia de los datos del procesador que realizó la actualización.

6 Aplicaciones en procesadores comerciales

Es importante considerar que varios de los protocolos mencionados se utilizan principalmente en arquitecturas específicas y/o con fines académicos. Sin embargo, los protocolos más ampliamente adoptados en el ámbito comercial son MESI y MOESI debido a su alta eficiencia y rendimiento. A continuación, se presentarán ejemplos donde cada protocolo tiene aplicación comercial:

- **Protocolo Firefly**: utilizado en la arquitectura DEC Firefly multiprocessor workstation, desarrollado por DEC Systems Research Center [14].
- **Protocolo Dragon**: utilizado en el multiprocesador Xerox Dragon, desarrollado por Xerox PARC [15].
- **Protocolo MESI**: utilizado ampliamente en microprocesadores comerciales como Intel's IA32 Pentium processor, Intel Xeon, AMD K6 y PowerPC 601 [16] [18].
- **Protocolo MSI**: utilizado en la arquitectura Silicon Graphics Professional 4D (SGI 4D) [17].
- **Protocolo MOESI**: utilizado en varios multiprocesadores comerciales modernos como la arquitectura AMD64 y SUN Microsystems UltraSPARC [18].
- **Protocolo MOSI**: utilizado en el procesador ruso Elbrus-4C, el cual es diseñado y desarrollado en los laboratorios de MCST en Moscú [19].

7 Conclusión

A lo largo de este ensayo de investigación hemos explorado en detalle los protocolos Snoop, analizando tanto el enfoque Write-Update representado por Firefly y Dragon, como el enfoque Write-Invalidate representado por MSI, MESI, MOSI y MOESI.

Cada uno de estos protocolos presenta características y funcionalidades específicas que pueden resultar beneficiosas en determinados escenarios, pero también pueden conllevar costos en otros contextos.

La elección adecuada del protocolo dependerá en gran medida de las necesidades y requisitos particulares del sistema en cuestión. Será crucial considerar cuidadosamente las características del sistema, tales como la frecuencia de escritura, el tráfico en el bus de interconexión, la latencia y el rendimiento general, a fin de determinar cuál de los protocolos se ajusta mejor a las demandas específicas.

Tras haber analizado las ventajas y desventajas de Write-Update y Write-Invalidate, se recomienda el uso del primer enfoque en casos donde la tasa de lectura de un mismo bloque en diferentes cachés es alta, mientras que la tasa de escritura es baja. Por el contrario, en situaciones donde la tasa de escritura es alta y las lecturas se realizan principalmente en la caché que modifica el bloque, se considera que el enfoque Write-Invalidate es la opción más adecuada.

En relación a estas situaciones, se enfatiza que la utilización del estado Exclusive puede mejorar de manera significativa el rendimiento del sistema cuando las operaciones de lectura y escritura se llevan a cabo principalmente en una única caché. Por otro lado, en escenarios donde se requiere un acceso compartido a un bloque de datos modificado

por múltiples cachés, se determina que incluir el estado Compartido-Modificado (Owned) es la elección óptima.

En última instancia, al comprender las características y las implicaciones de cada protocolo, los diseñadores y desarrolladores pueden tomar decisiones informadas para seleccionar el protocolo más adecuado para sus sistemas, optimizando así el rendimiento y asegurando la correcta sincronización de los datos compartidos en la memoria caché.

AGRADECIMIENTOS

Ing. Marchi Edgardo e Ing. Cervetto Marcos, docentes de cátedra Organización del Computador, Facultad de Ingeniería, Universidad de Buenos Aires.

REFERENCES

[1] García, A. C., Pindado, S., Gómez, F. *"Programa de simulación del protocolo de coherencia MESI"*. Boletín UPIITA, revista de ciencia, tecnología e innovación, número 35. [Online]. Available: https://www.boletin.upiita.ipn.mx/index.php/ciencia/217-cyt-numero-35/60-programa-de-simulacion-del-protocolo-de-coherencia-mesi.

[2] Straminsky, G. *Resumen de Organización del Computador 2: Cache Coherence*. CubaWiki, [Online]. Available: https://www.boletin.upiita.ipn.mx/index.php/ciencia/217-cyt-numero-35/60-programa-de-simulacion-del-protocolo-de-coherencia-mesi.

[3] Pingali, K. *"MESI (Modified, Exclusive, Shared, Invalid) Protocol"*. The University of Texas at Austin, [Online]. Available: https://www.cs.utexas.edu/~pingali/CS377P/2018sp/lectures/mesi.pdf.

[4] Redis. *"Cache Coherence"*. Redis Glossary. [Online]. Available: https://redis.com/glossary/cache-coherence/.

[5] Li, Y. *"Lecture 5: Cache Coherence"*. Carnegie Mellon University [Online]. Available: http://www.cs.cmu.edu/afs/cs/academic/class/15740-f19/www/lectures/05-coherence.pdf

[6] Georgia Tech, *MSI Coherence - Georgia Tech - HPCA: Part 5*, en: Udacity [Online]. Available: https://www.youtube.com/watch?v=gAUVAel-2Fg

[7] Georgia Tech, *Write Update Snooping Coherence - Georgia Tech - HPCA: Part 5* en: Udacity [Online]. Available: https://www.youtube.com/watch?v=rLmS02gsEAk

[8] Sundararaman Nakshatra, *Computer Architecture(EECC551), Cache Coherence Protocols* [Online]. Available: http://meseec.ce.rit.edu/551-projects/fall2010/1-3.pdf

[9] F. J. JIMÉNEZ1, J. GÓMEZ1, A. MESONES1, E. HERRUZO1, J. I. BENAVIDES1 Y F. J. SÁNCHEZ2, *Teaching the cache memory coherence with the MESI protocol simulator* Dpto. Electrotecnia y Electrónica. Escuela Politécnica Superior. Universidad de Córdoba. Av. Menéndez Pidal s/n. 14081. Córdoba. Spain. 2 I.E. S. Emilio Canalejo Olmeda. Av. Constitución, 18. 14550. Montilla. Córdoba. Spain. [Online]. Available: http://e-spacio.uned.es/fez/eserv/taee:congreso-2006-1112/S3E04.pdf

[10] Nima Honarmand, *Cache Coherence*, Stony Brook University, [Online]. Available: https://compas.cs.stonybrook.edu/~nhonarmand/courses/sp18/cse502/res/610/04-coherence.pdf

[11] *Lecture 11: Snooping Cache Coherence: Part II* CMU 15-418: Parallel Computer Architecture and Programming (Spring 2012) [Online]. Available: https://www.cs.cmu.edu/afs/cs/academic/class/15418-s12/www/lectures/11_coherence2.pdf

[12] Kshitiz Dange, Yash Tibrewal, *Cache Coherence Protocols Analyzer* 15-618 spring 2017 final [Online]. Available: https://kshitizdange.github.io/418CacheSim/final-report

[13] *Capítulo 5. Multiprocesadores*. Universidad de Valencia. [Online]. Available: https://www.uv.es/varnau/OC_T5_1.pdf

[14] *Firefly protocol*,Academic Directionaries [Online]. Available: https://en-academic.com/dic.nsf/enwiki/5951550

[15] *Protocolo DRAGON*, Wiipedia, La enciclopedia Libre. [Online]. Available: https://es.wikipedia.org/wiki/Protocolo_DRAGON

[16] *Second Generation Intel® Xeon® Scalable Processors* Datasheet, Volume One: Electrical. Intel. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/2nd-gen-xeon-scalable-datasheet-vol-1.pdf

[17] *Lecture 24: Parallelism* UTCS 352, Lecture 24 [Online]. Available: https://www.cs.utexas.edu/users/mckinley/352/lectures/24.pdf

[18] Taeweon Suh, The Academic Faculty, *INTEGRATION AND EVALUATION OF CACHE COHERENCE PROTOCOLS FOR MULTIPROCESSOR SOCS* [Online]. Available: https://repository.gatech.edu/server/api/core/bitstreams/378a706d-9be8-4f9f-93ad-d684005933ad/content

[19] V.S. Burenkov, *A Technique for Parameterized Verification of Cache Coherence Protocols*, JSC MCST, 24 Vavilov str., Moscow, 119334, Russian Federation, [Online]. Available: https://ispranproceedings.elpub.ru/jour/article/viewFile/324/170