Proyecto 1: Coherencia de cache

Emanuel Esquivel López
ema11412@estudiantec.cr
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Abstract*—Cache coherence is a very important part for the development of a processor, as well as for the correct processing of the generated instructions, it is also very important to take into account that the processor is not only in a multiprocessor system, so many times the Data exchange can be corrupted since they are updated frequently, in addition to this it is clear that the data is important within the system and any type of error can be fatal, so we know that there are various protocols that control this, which are MSI, MESI, MOESI among others, for this implementation the MSI protocol of directories between more caches is also used.**

***Palabras clave*—MSI, protocol, cache, coherence, direct**

I. Introducción

Los sistemas actuales son extremadamente complejos a nivel de hardware ya que llevaron muchos años a su elaboración, por lo que un elemento muy importante para su estudio, el cual es el procesador, este es el encargado de las operaciones elementales y lógica interna del computador, además de esto el cache es un elemento también el cual interactúa directamente con el procesador, por lo que lo hace una memoria extremadamente rápida comparándola por ejemplo con la memoria principal.

El echo de que esta memoria sea una memoria de alta velocidad nos da la facilidad de respuesta inmediata en algunos casos donde esta es utilizada y ayuda a explotar ciertas capacidad, como la localidad de distintos datos y así poder aumentar la eficiencia de algunos programas los cuales sus datos últimamente accesados depende entre si, pero que pasa cuando tenemos mas de un programa o aplicación, llamado *cliente*, los cuales acceden a datos en la cache, los cuales pueden ser datos compartidos o no, llega un conflicto el cual consiste en la incoherencia de cache, o inconsistencia.

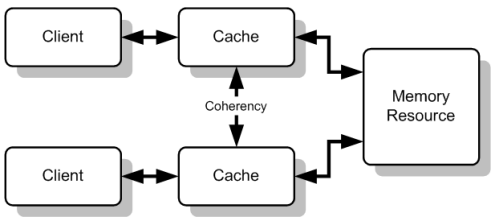

Figure 1. Coherencia de cache

Este problema es generado ya que hay casos donde se debe de leer y/o escribir en cache por distintos procesadores a distintas memoria cache, si solo fuera lectura esta inconsistencia no existiría, por lo que debe existir métodos o protocolos los cuales permitan erradicar estos problemas o bien poder tratarlos de la mejor manera para ya sea el caso algún procesador no accese a datos des actualizados o no esperados por este,estos métodos utilizados son MSI, MESI y MOESI, además de estos hay un protocolo también muy conocido para esto llamado protocolo de directorio.

En el presente documento se hablara del protocolo MSI, el cual fue implementado en el proyecto de manera practica al acceso de múltiples CPU a su respectiva memoria cache L1, y además protocolo directorio con L2 común.

II. Marco teórico

II-A. Protocolo MSI

Este protocolo se basa en los estados M, S, I los cuales son definidos a continuación:

- M (Modificado): Este estado es colocado cuando se modifica cierta dirección en la memoria.
- S (shared) : Este es el estado compartido, es cuando se lee la dirección de memoria por cualquier otro core.
- I (Invalido) : Es un estado determinado para la escritura, si se escribe en un I tenemos un WM lo que lo cambia al estado modificado.

La cache como esta en constante cambio se ve reflejado de esta manera, ya que si hay muchos procesadores accediendo a la memoria, con esto si un procesador accede a una dirección de memoria y la modifica, y casualmente esta esta en la cache L1 de otro core, el problema es que ya este dato no seria el actual lo que debe de se actualizado.

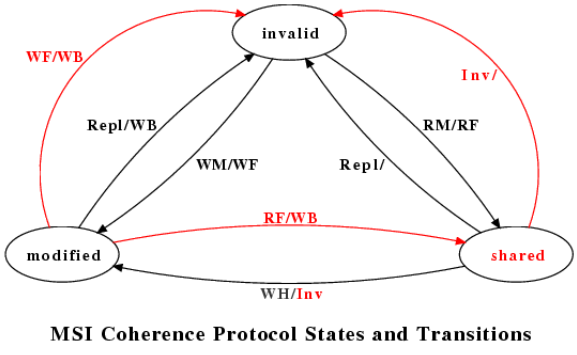

Figure 2. Estados MSI

II-B. Protocolo de directorio

Como fue visto anteriormente es un protocolo de coherencia de cache muy utilizado, este básicamente funciona como los otros ya que permite controlar ese problema solo que lo hace mediante el uso de un bus, también llamado bus compartido, este básicamente pregunta al bus el estado de los directorios para así no ser saturado. [1]

Podemos ver unas ventajas y desventajas a continuación:

- Escalabilidad: Es una parte muy importante por la cual muchos se decantan por este método, por lo que tiene gran flexibilidad al aumento creciente de trabajo, o nodos de trabajo, pero a su vez si este numero crece desmesuradamente podría ser fatal y surgir muchos problemas
- Simplicidad: Es prácticamente el punto mas importante en este protocolo ya que este nos permite organizar todo el trafico que pasa en el sistema, asegura la atomicidad de todas las señales además de que no habrá que esforzarse para garantizar orden del trafico.

Podemos ver el diagrama en la figura 4.

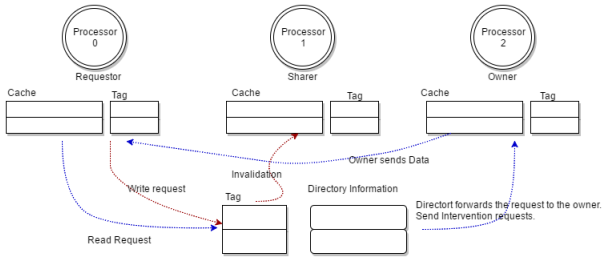

Figure 3. Protocolo directorio

Este tipo de protocolo separa el bus en los siguientes nodos:

- Nodo: El solicitante ya sea de escritura o lectura en bloque de memoria.
- Nodo directorio: Es el que mantiene el estado de cada bloque del cache del sistema, el nodo solicitante se comunica con este.
- Nodo propietario: Posee el estado mas reciente de memoria.
- Nodo compartido: Nodos que comparten copias del bloque de memoria.

Para este protocolo existen una serie de estados los cuales son muy importantes de concer, los cuales nos dicen los estados iniciales y finales, solicitud de petición al bus y la acción a tomar en cuenta [1].

Inicial	Solicitud	Respuesta	Nuevo
U	Rd RdX	Obtener bloque de memoria actualizada Envio de bloque usando mensaje (ReplyD) Si no hay cliente el directorio pasa a EM	EM
EM	Rd	Responde intervención al owner (Int)	S
	RdX	Envié la invalidación al owner (Inv)	
S	Rd	Responde con bloque de memoria (ReplyD)	
	RdX	Responde con bloque de memoria (ReplyD) Invalida a los clientes (Inv)	EM
	Upgr	Invalida a los clientes (Inv) Notifica que puede actualizar	EM

Para el caso del proyecto se simplifico a solo los estados DM, DS, DI, respectivamente con la funcionalidad del MSI tradicional explicado anteriormente.

III. Sistema desarrollado

El sistema desarrollado se puede ver en la figura 4, el cual nos muestra como interactúan los principales entres en el sistema, como lo son las memorias cache con la memoria principal, el cual sigue los principios establecidos en el marco teórico.

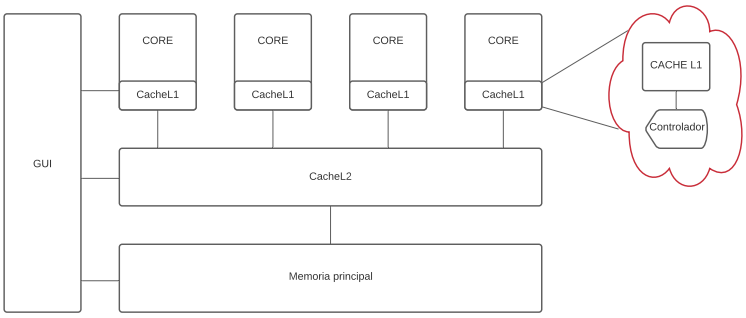

Figure 4. Diagrama de contexto de arquitectura

Se logra ver que cada core principal tiene su propia memoria cache L1, se ve en la parte derecha que esta en su interior tiene un controlador propio, el cual es el encargado de estar actualizando la memoria antes de cada operación.

El sistema de cache L1 se encarga de escribir a L2 y Memoria principal si es necesario, por lo que en general es la mente principal de todo el programa, cada vez que se realiza una acción en la memoria por parte del procesador, menos por *CALC*, L1 tiene ese método el cual es el encargado de consultar a L2 los estados de las direcciones de memoria, así como también los estados y propietarios, con esto podemos ver si el dato en L1 es o no valido para la lectura, o bien es valido y puede ser leído desde L2. En caso de que no sea valido este deberá ir a la memoria principal y actualizar.

La memoria principal no tienen nada mas direcciones y datos.

El sistema al inicio tiene todo invalido y en blanco los valores en memoria, esto para forzar a escribir o a leer directamente a memoria.

Los valores en L2 son actualizados con lecturas o con escrituras a la memoria principal.

IV. Resultados y análisis

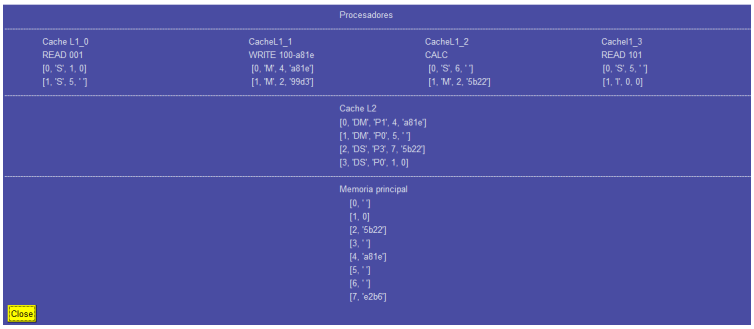

Figure 5. Resultados

Como se ve en la figura aca podemos verificar las acciones realizadas en las iteraciones del procesador, como se ve el procesador 0 hace una lectura en 001, el lo hace de manera efectiva, lo coloca en L2 como se ve en el bloque 3, y así en el bloque 0 de su memoria L1, se logra ver en L2 que lo pone como propietario, ademas de poner S y DS en las memorias respectivas.

Esta funcionalidad se puede ver también con el core 3, el cual lee en 5.

El core 1 hace un read en 4 como se puede ver, va a memoria principal y escribe en esta, a su vez coloca los valores en L2 y L1 de su cache, y también pone su propietario y por ultimo deja en estado M, y DM respectivamente.

Cuando estos datos son editados nuevamente se produce una actualización en las cache L1 de cada procesador.

V. Conclusiones

El protocolo MSI es de gran utilidad para los protocolos de coherencia de cache, ya que este permite de mejor manera el manejo de estas memorias y es mas simple que los demás protocolos de coherencia de cache.

Ir a memoria principal es un método mas costoso que realizar operaciones básicas en la memoria cache ya sea L1 o L2, estas en ese orden tienen menor tiempo de acción con los datos y operaciones sobre la memoria.

References

[1] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," *Computer*, vol. 23, no. 6, pp. 49–58, 1990.

[2] X. Zhang, "Verification strategy of cache coherence for opensparc t 2 multi-processor systems under the direction of dr," 2013.

[3] T. Suh, D. M. Blough, and H.-H. Lee, "Supporting cache coherence in heterogeneous multiprocessor systems," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 2. IEEE, 2004, pp. 1150–1155.

[4] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 423–434.

[5] H. Altwaijry and D. S. Alzahrani, "Improved-moesi cache coherence protocol," *Arabian Journal for Science and Engineering*, vol. 39, no. 4, pp. 2739–2748, 2014.