Proyecto 1. Protocolos de coherencia en sistemas multiprocesador

Franciso J. Murillo Morgan
fmurillom@estudiantec.cr
Área académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—Durante la realización de este proyecto se utilizaron los conceptos de coherencia de memoria, protocolo basado en directorios, y el protocolo MSI para la realización de un simulador de un sistema multiprocesador en el que se eliminen e identifiquen los problemas de coherencia que podrían manifestarse en las memorias cache de los procesadores.**

Para la implementación, se utilizó el lenguaje de programación Python,para desarrollar una abstracción de los elementos principales de un sistema multiprrrocesador, asi como un controlador de memorias cache, cuyo protocolo de monitoreo sea capaz de prevenir cualquier tipo de incoherencia de cache que pueda darse dureante la ejecución. Además de eso se utilizaron 4 hilos distintos para la simulación de 4 nucleos de procesamiento simultaneo.

***Palabras clave*—Memoria Cache, Procesador, Multiprocesador, Python, Coherencia, Memoria Principal, Protocolo MSI**

I. Sistema Desarrollado

Para el desarrollo de este primer proyecto, se desarrollo un simulador de un sistema multiprocesador. Este sistema multiprocesador consistirá de dos procesadores. Cada procesador contará con dos núcleos,una memoria caché L2, y un controlador para la coherencia de caches. Cabe destacar que cada uno de los núcleos dentro del procesador tendrá su propia memoria caché L1.

I-A. Estructura por bloques de cada chip de memoria

En la figura 1 se puede apreciar un diagrama de la composición que tendrá cada uno de los procesadores en el sistema.

En las figuras 2 y 3 se puede observar la estructura de cada una de las memorias cache L1 y L2 respectivamente.

Como se puede observar en la figura 2, cada bloque de la memoria L1 tendrá almacenado:

- Estado de coherencia: Este bloque informará cual es el estado del bloque de cache, ya sea Modificado (M), Compartido (S), o Invalido (I). Cada uno de estos estados será definido más adelante.
- Direccion de memoria: Este bloque informara a que direccion de la memoria principal esta haciendo referencia el bloque.
- Dato: Este bloque almacenara el dato del bloque, ya sea leido directamente de memoria, o si este ya fue modificado por el nucleo.

Como se puede observar en la figura 3, cada bloque de la memoria L2 tendrá almacenado:

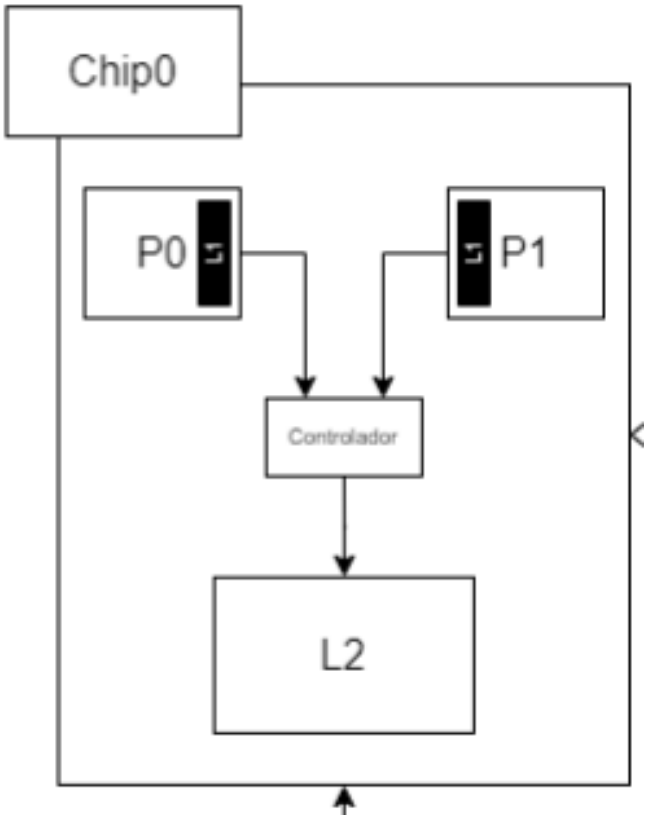

Figura 1. Ejemplificación de la estructura interna de cada chip.

P0, 0

	Estado de coherencia	Dirección de memoria	Dato
0	M	1011	FFFF
1	S	0110	FFFF

Figura 2. Estructura por bloques de la memoria cache L1.

L2, 0

	Estado	Dueño	Dirección de memoria	Dato
0	DM	P0,1	1011	FFFF
1	DS	P0,0;E	0110	FFFF
2	DI	P1,0	0111	FFFF
3	DS	P1,0	1000	FFFF

Figura 3. Estructura por bloques de la memoria cache L2.

- Estado de coherencia: Este bloque informará cual es el estado del bloque de cache, ya sea Modificado (M), Compartido (S), o Invalido (I). Cada uno de estos estados será definido más adelante.
- Dueño: Este bloque nos indicará que caches L1 son dueños del dato.
- Direccion de memoria: Este bloque informara a que direccion de la memoria principal esta haciendo referencia el bloque.
- Dato: Este bloque almacenara el dato del bloque, ya sea leido directamente de memoria, o si este ya fue modificado por el nucleo.

Cada uno de los procesadores independientes,será conectado a una memoria principal,la cual será la encargada de almacenar todos los valores que los chips necesitan para lectura o escritura. En la figura 4 se puede observar la estructura por bloques que tendrá dicha memoria principal.

M

	Estado	Dueño(s)	Dato
1011	DM	C0	FFFF
0110	DS	C0, C1	FFFF
0111	DI	C1	FFFF
...	...	...	...
1000	DS	C0	FFFF

Figura 4. Estructura por bloques de la memoria principal.

Como se puede observar en la figura 4, los bloques de memoria principal contienen:

- Estado: Este bloque informará cual es el estado del bloque de cache, ya sea Modificado (M), Compartido (S), o Invalido (I). Cada uno de estos estados será definido más adelante.
- Dueño(s): Este bloque nos indicará que chips son dueños del dato.
- Dato: Este bloque mostrará el valor almacenado en dicha dirección de memoria.

I-B. Protocolo de Coherencia

Para este proyecto se utilizo el protocolo de coherencia de cache MSI. Este protocolo de coherencia se basa en cambiar el estado de cada uno delos bloques de cache, para identificar si un bloque se encuentra invalido, modificado o compartido. La definición de cada estado será:

- Modificado (M): EL bloque ha sido modificado en cache, esto significa que el bloque es inconsistente con el dato en memoria principal, por lo que el dato mas actualizado será el que se encuentra en memoria cache.
- Compartido (S): Este bloque no ha sido modificado y ha sido leido y almacenado en más de una memoria cache.
- Invalido (I): Este bloque actualmente no se encuentra cargado en caché, o ha sido invalidado por el bus de datos, y este debe ser leido de memoria principal o de otra cache antes de poder ser utilizado.

Una manera sencilla de entender el protocolo, y los cambios de estado presentes en este, es visualizar el protocolo como si este fuese una maquina de estados. En la figura 5 se puede observar una representación de dicha maquina de estados. En esta se incluye las causas de cada transición ya sea por alguna instrucción del procesador o del bus de memoria.

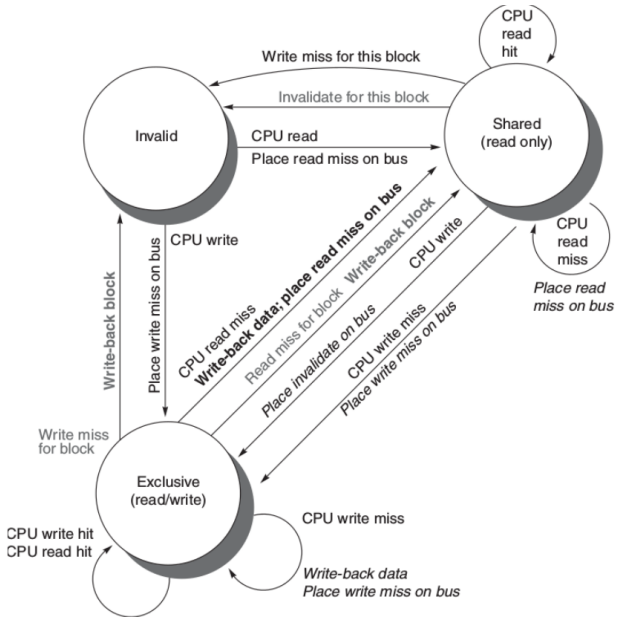

Figura 5. Estructura por bloques de la memoria principal.

I-C. Método de Escritura

En los sistemas multiprocesadores, existen dos métodos para la escritura de datos en memoria. Estos dos métodos son:

- Write Through: Para este método de escritura, cada vez que el procesador genere una instrucción de escritura, este se encargará de escribir el dato tanto en memoria cache como en la memoria principal. De esta manera cada vez que otro procesador necesite del dato actualizado, este puede buscarlo directamente en memoria, y al leer el dato, este será el más nuevo.

- Write Back: Para este método de escritura, cuando el procesador necesite actualizar el dato de una posición de memoria, este solamente realiza la escritura en la memoria cache correspondiente. En el caso de que algún otro procesador necesite el dato, este debe buscar el dato respectivo en las otra memorias cache para así poder cargar el dato. El dato luego será escrito a memoria principal en el momento en que este ya no sea necesitado y el bloque que lo contiene sea remplazado por la política de reemplazo de bloque de cache respectiva.

Para la realización de este proyecto, se utilizo write back, esto debido a que como se debe utilizar el protocolo basado en directorios (este será especificado más adelante) apoyandose en la memoria cache L2, se tiene un buen conocimiento la ubicación de todos los datos en las memorias cache del sistema multiprocesador, por lo que buscar datos entre caches se vuelve más sencillo. Además de esto, se agiliza la escritura y lectura de datos ya que se disminuye la cantidad de veces que se debe acceder a memoria principal para buscar algún dato.

I-D. Protocolo basado en directorios

Este protocolo se basa en la utilización de una memoria cache extra, la cual contendrá la información de cada uno de los bloques en memoria principal que se encuentran distribuidos a través de las distintas cache. Para este proyecto, las caches L2 en cada uno de los chips serán las memorias dedicadas para este protocolo. Como se observar en la figura 3 las cache L2 contendrán la información general de la ubicación de cada bloque de memoria en cada una de las cache.

I-E. Implementación

Para la implementación de este simulador,se utilizó el lenguaje de programación Python, esto debido a que es un lenguaje de programación que permite la programación orientada a objetos. El poder abstraer cada uno de los bloques del sistema como un objeto, facilita la implementación y la comprensión del sistema. Además de esto, Python facilita la creación de hilos de procesamiento, con el fin de acercarse lo mejor posible a un sistema multiprocesador, el cual tendrá varios núcleos generando instrucciones al mismo tiempo.

El diagrama UML de la figura 6 permite ver la relación e interacción que existe entre las distintas clases implementadas para la realización de este proyecto.

A continuación se detallará la implementación de cada clase, y el papel que cada una de estas juega en la implementación del sistema.

I-F. Bloque_L1

Esta clase representará un bloque de memoria dentro de la cache L1.

I-G. Memoria L1

Esta clase se encarga de realizar la abstracción de una cache L1. Esta clase esta compuesta por dos instancias de la clase Bloque_L1.

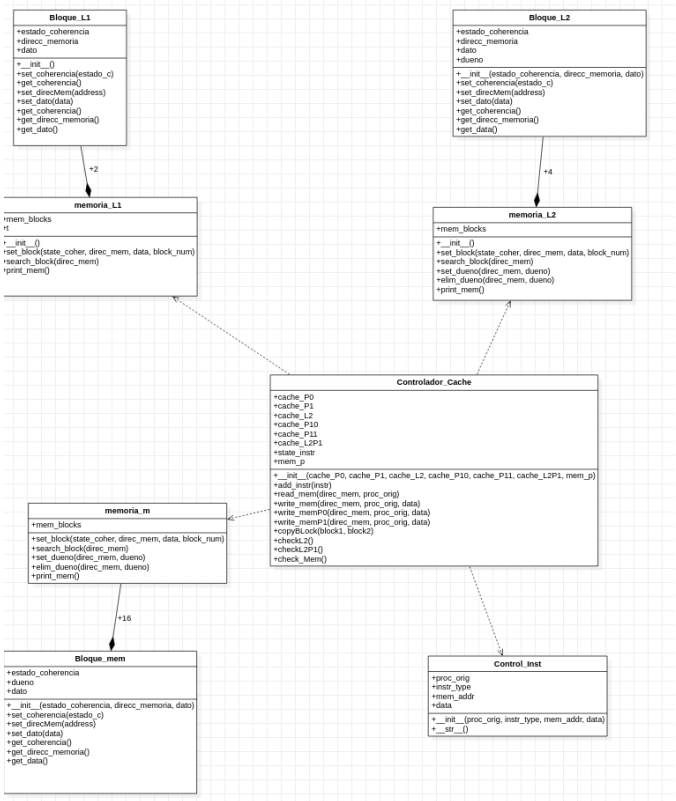

Figura 6. Estructura por bloques de la memoria principal.

I-H. Bloque_L2

Esta clase representará un bloque de memoria dentro de la cache L2.

I-I. Memoria L2

Esta clase se encarga de realizar la abstracción de una cache L1. Esta clase esta compuesta por dos instancias de la clase Bloque_L2.

I-J. Bloque_mem

Esta clase representará un bloque de memoria dentro de la memoria principal del sistema.

I-K. memoria_m

Esta clase representará la memoria principal del sistema. Esta clase esta compuesta por 16 instancias de la clase Bloque_mem

I-L. Control_inst

Esta clase es la encargada de representar las instrucciones generadas por cada uno de los núcleos de procesamiento. Cada instancia de esta clase incluira el procesador que genero la instruccion, la accion a realizar, la direccion de memoria a la cual realizar la accion, y en caso de ser necesario, el dato que se debe escribir en la direccion de memoria seleecionada. Para la generacion de instrucciones automaticamente, se utilizó la distribución de probabilidad de Poisson. De esta manera

nos asegurmaos que las instrucciones no son instrucciones aleatorias, si no que existirá una relación entre ellas y de esta manera se simula de mejor manera el comportamiento real de un sistema multiprocesador.

I-M. Controlador_Cache

Esta clase es la encargada de aplicar el protocolo de monitoreo diseñado para la implementación de este simulador. Cada vez que alguno de los núcleos genera una instrucción de lectura, esta clase es la encargada de leer la instrucción e identificar la direccion de memoria que se desea leer. Al haber identificado dicha direccion, se encarga de revisar en ambas cache L2 y buscar que bloques de cache L1 en cada nucleo hace alguna referencia a la direccion de memoria deseada. En caso que algun bloque de cache L1 tenga una copia del dato, el controlador procede a realizar la lectura del valor de dicho dato directamente de la cache en donde se encontró la copia. Una vez hecho esto, se procede a marcar la direccion de memoria en las caches como un bloque en estado Shared (S). Para el caso en que el controlador no encuentre una referencia a la direccion de memoria, dentro de alguna de las cache L2, el controlador procede a leer el dato directamente de la memoria principal, y procedea escribir dato en el bloque de cache seleccionado, poniendo este en un estado de Modified (M). En el caso que no exista un bloque de memoria L1 disponible para almacenar referenciar el bloque de memoria, el controlador procede a seleccionar un bloque a reemplazar con este nuevo dato, realizar el proceso de escritura en memoria principal (proceso que será descrito más adelante) y a reemplazar el bloque seleccionado con el bloque que se desea referenciar.

Para el caso de una instrucción de escritura en memoria, el controlador primero verificara en ambas cache L2, si alguno de las caches L1 de los núcleos posee una referencia a dicha dirección de memoria. De ser el caso, el controlador procede a poner los bloques compartidos en estado invalido, de esta manera el único bloque que contendrá el dato mas nuevo será el que se acaba de escribir en alguna de las cache L1. Luego de hacer la escritura y la invalidación de los bloques compartidos, el controlador procederá a actualizar los bloques activos y compartidos para las memorias cache L2 en cada uno de los chips.

II. Resultados

A la hora de realizar la implementación descrita en la sección anterior, se obtuvo un simulador de un sistema multiprocesador, en cuya simulación es posible verificar la implementación del protocolo MSI a travez del protocolo de monitoreo implementado con la clases Cont_Cache en cada uno de los bloques de caches L1. Además a lo largo de toda la simulación, se pueden observar los estados de cada uno de los bloques de memoria cache L1 y L2 asi como los bloques ya en memoria principal. Al poder observar el contenido de las cache L2, se puede verificar la aplicación del protocolo basado en directorios en la cache L2.

III. Conclusiones

A travez de la realización de este proyecto, se logró entender como se manejan y se eliminan los problemas de coherencia que pueden aparecer en las distintas caches de cada uno de los procesadores. Con la implementación realizada, se demuestra que es más eficiente (a nivel de control de flujo de datos) utilizar un solo controlador para las memorias cache L1 de ambos chips, en lugar de tener un controlador de caches L1 aislado para cada uno de los chips. Además de esto, al utilizar un solo controlador de caché, se puede notar que el uso del bus de datos es mucho mayor, en comparación a tener dos controladores de cache por separado. Esto podría llegar a ser un problema dependiendo de la aplicación que se le vaya a dar al sistema multiprocesador. Al aplicar el protocolo de escritura write-back, se agiliza el tiempo de escritura de datos en las cache, esto debido a que en write-back, solo se escribirá el dato a memoria principal en el caso de que este dato ya no sea necesitado en el flujo de instrucciones del procesador. Por último, al aplicar el protocolo basado en directorios, se tiene una mejor noción de los datos que se encuentran en cada una de los bloques en las cache de cada nucleo. De esta manera se facilita la busqueda de datos compartidos entre caches.

Referencias

[1] T. Suh, ÏNTEGRATION AND EVALUATION OF CACHE COHERENCE PROTOCOLS FOR MULTIPROCESSOR SOCS", Smartech.gatech.edu, 2006. [Online]. Available: https://smartech.gatech.edu/bitstream/handle/1853/14065/suh_taeweon_200612_phd.pdf.

[2] "threading — Thread-based parallelism — Python 3.8.3 documentation", Docs.python.org, 2020. [Online]. Available: https://docs.python.org/3/library/threading.html.