SW Cache Coherence modeling and evaluation

Juan Peña-Rostran, Leonardo Guillen-Fernandez, Brandon Gómez-Gómez, Josue Cubero-Montero
Email: juanpenarostran@estudiantec.cr , Leoguillen@estudiantec.cr , brandonmgg@estudiantec.cr , jdcubero@estudiantec.cr
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—El presente paper centra su atención en el como mantener la integridad de los datos en un procesador multicore; para esto, se expone dos protocolos para mantener la coherencia de cache: MESI y MOESI. Ademas, se exponen algunas de sus ventajas, desventajas y evaluamos el desempeño de estos protocolos. Además, se exponen las principales ideas sobre el desarrollo de sistema multicore con la capacidad de soportar los protocolos de Coherencia de memoria Cache**

***Keywords*—Coherencia de cache, MOESI, MESI, Cola de mensajes.**

I. Introduccion

El siguiente proyecto se basa en el diseño e implementación de un sistema multiprocesador con la ejecución de 3 PEs que comparten una única memoria compartida, funcionando como Memoria Principal y cada PEs tendrá una memoria de caché privada. Este sistema se desarrollará utilizando threads para poder ejemplificar y simular la ejecución paralela y la utilización de recursos, implementando dos protocolos de coherencia de caché (MOESI y MESI) que garantizan la consistencia entre datos de los distintos PEs

Los sistemas multiprocesador son fundamentales en la actualidad debido a su capacidad para mejorar el rendimiento en la ejecución de tareas computacionales. Estos sistemas permiten la ejecución simultánea de múltiples hilos de ejecución, lo que lleva a una mayor eficiencia en la utilización de recursos de hardware. La implementación de protocolos de coherencia de caché en sistemas MP es esencial para garantizar que los datos almacenados en caché sean coherentes entre los distintos núcleos de procesamiento, lo que evita resultados incorrectos y problemas de sincronización en las aplicaciones.

El proyecto tiene como objetivo diseñar y desarrollar un sistema MP con al menos tres PEs y una memoria compartida. Esto permitirá a los usuarios comprender y explorar los conceptos clave de la programación paralela, la gestión de la memoria compartida y los protocolos de coherencia de caché. Además, la implementación de una interfaz gráfica proporcionará una manera visual de comprender los estados de las líneas de caché y las transacciones realizadas, lo que facilitará el aprendizaje y la depuración de programas paralelos. Con la llegada de los sitemas multicore, la consistencia de los datos experimenta un nuevo problema: Un mismo dato puede estar presente en multiples caches de manera simultanea. Si un core 0 accesa una direccion especifica y el valor correcto en esta direccion se encuentra en la cache del core 1, entonces el core 0 estaría utilizando un valor desactualizado del dato, incidiendo en resultados erróneos y aplicaciones no funcionales.

Si estos problemas ligados a la integridad de los datos no son solucionados, se puede caer en problemas importantes en la correcta ejecución de un programa. Es aqui donde los protocolos para la coherencia de cache, entran al juego. Estos protocolos definen reglas y mecanismos para asegurar la coherencia de las caches.

Hoy en dia, no solo existe un protocolo de coherencia de cache, esto pues, no solo se busca solucionar el problema de coherencia sino que tambien buscan hacerlo de la manera mas eficiente en términos de tiempo, trafico de datos y accesos remotos. Es debido a esto que, en este paper, se busca analizar dos protocolos de coherencia de cache.

II. Propuesta de Diseño y Evaluación de Ventajas/Desventajas

Diseño conceptual del sistema.

Con el objetivo de demostrar el comportamiento del sistema y la interacción entre las distintas partes de este, en la Figura 1 se muestra un diagrama a alto nivel del sistema.

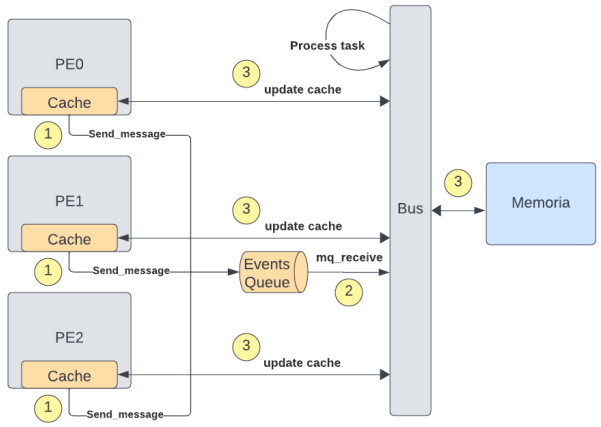

Figura 1: Secuencia en la interacción de todo el sistema.

La secuencia de la Figura 1, muestra:

1. Flujo de la cache a la cola
2. Flujo de la cola hacia el bus
3. Flujo del bus hacia las caches y la memoria

Ventajas del enfoque de coherencia de caché por medio de una cola:

- FIFO: Gracias a la estructura de una cola (*Fisrt In-Firts Out*), se puede forzar el orden de las operaciones sobre la memoria y cache, asegurando que dichas operaciones se realicen en el orden en el que fueron recibidas.
- Sincronización: Las colas, tambien funcionan como un mecanismo de sincronizacion, de modo tal que nos ayuda a solucionar el problema de condiciones de carrera en casos en los que el las caches quieran acceder mismas direcciones de, por ejemplo, la memoria al mismo tiempo.
- Desacople: Las colas pueden desacoplar el manejo de la coherencia de cache por parte del CPU, permitiendo que este pueda continuar sus operaciones sin tener que esperar por la actualización de la cache.

Desventajas del enfoque de coherencia de caché por medio de una cola:

- Buffering: Por la naturaleza de FIFO de las colas, estas actuan como un buffer y aunque esto resulta util para la sincronizacion de las peticiones, ante mucho trafico de datos, se pueden tener bottleneckes.
- Blocking: Otro problema ligado al buffering es que, si la cola se llena, esto puede ocacionar a un bloque del sistema, lo cual ocasiona delays en el sistema.

Como se expuso en la Figura 1, donde se muestra todo el sistema multi-procesador, podemos analizar cada uno de los puntos **1**, **2**, **3** y que es lo que sucede en cada uno de estas interacciones.

Inicialmente, de la Figura 2 se puede ver la interacción entre la cache con la cola de mensajes. Tal y como se ha mencionado, esta interacción se da cada que el CPU realiza una petición a la cache y esta no tiene la información necesaria para procesarla.

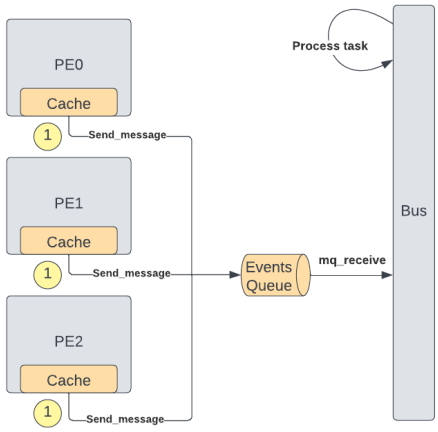

Figura 2: Comunicación de la caché con el bus por medio de la cola de mensajes.

Uno de los objetos mas importantes en todo el sistema es el bus. El flujo del funcionamiento interno de este es lo suficientemente importante como para analizarlo. En la Figura 3 se tiene el diagrama asociado a la interacción del bus con la cola de mensajes.

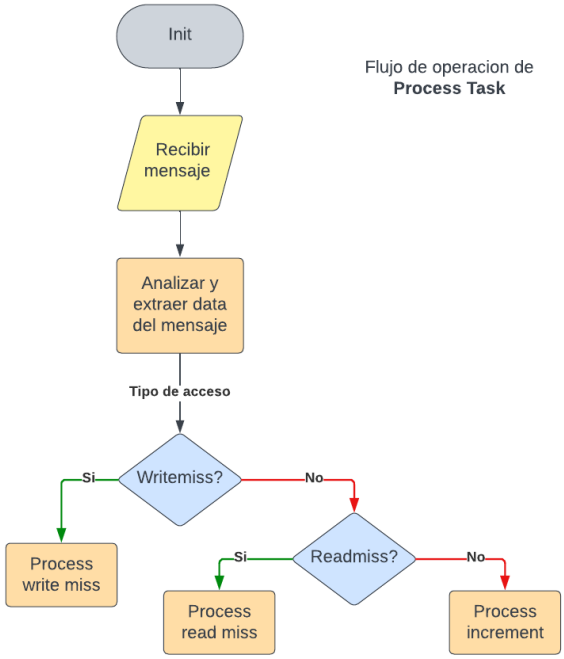

Figura 3: Flujo de operación interno del bus.

Del diagrama de la Figura 3, se puede observar la relevancia en el tipo de acceso por el cual se envía el mensaje al bus. En ese punto, la operación del bus se divide en tres partes:

- Procesamiento del readmiss.
- Procesamiento del writemiss.
- Procesamiento del increment.

Finalmente, se tiene la parte de la interacción entre el bus, las memorias cache y la memoria principal. Esta interacción no es siempre la misma, pues depende de la operación o mensaje que se este procesando. Tal y como se mencionó con anterioridad, se puede tomar uno de tres posibles caminos. En términos generales, el bus lee la información dentro del mensaje y función de esto, realiza write back a la memoria, actualiza el estado de las caches y envía los datos correspondientes a cada uno de estos.

III. Especificación de Componentes del Sistema

Partiendo de lo mostrado en la Figura 1 donde se muestra un diagrama con las principales componentes del sistema, en la Figura 4, se muestra la relación entre los objetos y los atributos y métodos principales de los mismos.

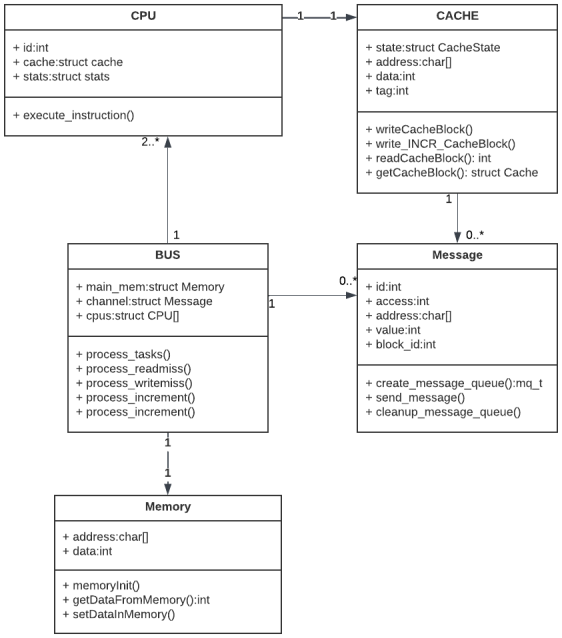

Figura 4: Flujo de operación interno del bus.

Los objetos principales que forman el sistema multi-procesador del diseño realizado son:

- CPU
- Cache
- Memoria
- Mensaje
- Bus

III-A. CPU

Se encarga de la ejecución de instrucciones La instrucción a ejecutarse contiene:

- La operación a realizarse:
  - READ
  - INCR
  - WRITE
- La dirección de memoria
- El dato

Indistintamente de la operación a realizarse, el CPU accede a la caché ya sea para una lectura o escritura por medio de los métodos **readCacheBlock** y **writeCacheBlock** de la memoria cache.

III-B. Cache

Contiene un conjunto de bloques de caché e información sobre el estado de cada bloque de caché:

- State
- Address
- tag
- data

En la sección anterior se dijo que el CPU hace las escrituras y lecturas a la cache. Debido a que; como bien sabemos, los valores dentro de esta puede que no estén en el estado adecuado para su posterior operacion. Por tanto, en aquellos donde deba de actualizarse de las cache, esta envía un mensaje a la cola del bus. El único caso en el que no se realiza ningun llamado al bus es cuando se da un write hit, de modo tal que se puede escribir el dato y cambiar su estado a **Modified**. En la Figura 5 se encuentra los metodos que funcionan como interfaz entre la comunicacion de la cache y el CPU.

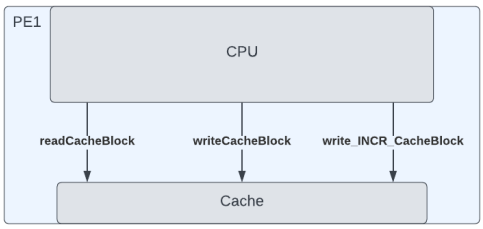

Figura 5: Interacción CPU - Cache

III-C. Mensaje

Para poder comunicar las caches y la memoria por medio del bus, se usa la abstracción de un mensaje. La idea de este objeto es la de proporcionar a las caches el mecanismo mediante el cual pueden colocar la información que debe de ser actualizada en la cola del bus. Debido a que se tienen múltiples caches ejecutándose simultáneamente, el mensaje también busca volver transparente el proceso de sincronización del acceso a la cola del bus.

III-D. Bus

El bus tiene dos funciones principales:

- Manejo de mensajes.
- Interacción con los CPUs y la memoria.

En este caso, el bus se comporta como un mediador.

El objetivo de este es el de reducir dependencias entre objetos. Este patrón restringe la comunicación directa entre objetos forzandolos a colaborar solo a través de un mediador; en este caso, el bus. De este modo, el bus interactúa con los CPUs/cachés y la memoria. El modo en el que el bus procesa las solicitudes de las cachés es por medio de una cola. Esta cola recibe un mensaje(una estructura con toda la información ligada al mensaje). A partir de este mensaje se puede obtener información sobre la caché del CPU que envía el mensaje, la dirección, el valor y el tipo de acceso por el cual entró al bus: **WRITEMISS o READMISS.**

III-E. Memoria

Consiste en una estructura que contiene bloques de memoria. Estos bloques de memoria están formados por una dirección y por el dato que están pueden contener. El conjunto de bloques de memoria se encuentran en un array formando así la memoria. La memoria proporciona una “interfaz” para realizar escrituras y lecturas sobre esta.

III-F. Interfaz

La interfaz consiste en un sistema construido en C con la libreria GTK la cual consiste en un sistema gráfico para la creación de interfaces. En este caso se desarrollo un sistema sincronizado con el backend el cual actualiza de manera dinamica los datos de la interfaz segun el contenido proveniente del bus de datos del sistema. Para la interfaz se uso principalmente CSS, lo cual permitio obtener un sistema mas ordenado y detallado. Si bien la creacion de interfaz es complicada en C, el uso de esta herramienta junto con CSS hizo mas facil el desarrollo de la misma. En cuanto a los componentes que conforman la interfaz, se tiene un diseño simple basado en matrices, con labels y botones, asi como desplegables para el ingreso de datos. Los botones cumplen la funcion de ejecutar internamente funciones, mientras que los desplegables y campos de texto el de introducir datos. Y por su parte los labels cumplen la importante tarea de mostrar los datos al usuario segun la ejecucion del programa.

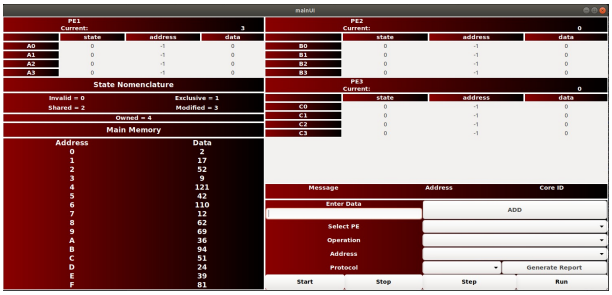

Figura 6: Enter Caption

IV. Proceso de Diseño

Para llevar a cabo este proyecto, se seguirá una metodología de diseño iterativa y colaborativa. Esta metodología se ha elegido debido a la complejidad de la implementación y la necesidad de adaptarse a medida que se avanzan en las etapas del proyecto. Cada etapa del proceso de diseño se centrará en la planificación, desarrollo, prueba y mejora de componentes clave, con un enfoque en la modularidad y la eficiencia.

- Etapa 1: Primero se deben establecer los objetivos y visualizar el alcance del proyecto, tomando la metodología de desarrollo iterativo para permitir adaptaciones a medida que se avanza en el proyecto. En primera instancia se definen componentes principales como lo son PEs, cachés privados, memoria compartida o memoria principal, protocolos de coherencia y la interfaz gráfica
- Etapa 2: Se procede al desarrollo de cada componente de manera modular para lograr una fácil integración y para la fácil interacción con pruebas de funcionamiento, además se debe adaptar cada módulo para poder tener la capacidad de poder elegir cual protocolo utilizar y que el sistema funcione correctamente.
- Etapa 3: Conforme se desarrollen los componentes, se deben hacer pruebas unitarias y de interacción para poder identificar errores y solucionar problemas de compatibilidad con los demás módulos, garantizando la funcionalidad correcta desde cualquier función de los componentes.
- Etapa 4: Se procede a realizar iteraciones y optimizaciones de los componentes, esto con el hecho de que si se necesita un nuevo parámetro en las funciones, sea fácil de agregar nuevas funcionalidades o métodos, y además realizar sus funcionalidades de la manera más efectiva posible, ya que al ser un simulador, un proceso mal diseñado puede proceder a resultados erróneos.
- Etapa 5: Implementación de la interfaz gráfica para visualizar todos los estados de las cachés y las transacciones entre PEs, además de una generación de código con la cual poder realizar pruebas especializadas desde la interfaz, además esta debe ser intuitiva y fácil de manejar

A nivel de pruebas se deben realizar varias conexiones entre los componentes tal y como se muestra en la siguiente Figura 7.

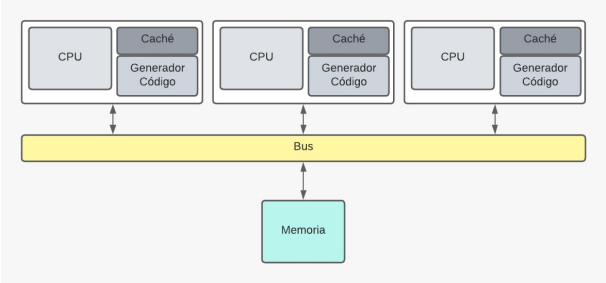

Figura 7: Conexión a alto nivel de todo el sistema.

Como se observa, se deben realizar 3 conexiones, la primera debe ser entre CPU, caché y el generador de código, en base a esto se darán asignaciones a líneas de caché, dependiendo de los estados y las direcciones que contengan estas, además deben conceder el mensaje en ciertas situaciones y condiciones con las que debe ir a memoria, luego teniendo esta primera parte funcional , se procede desarrollar la interacción del bus con memoria principal, basado en la recepción de mensajes de los diferentes CPUs se debe interpretar el mensaje y con esa información poder elegir que se debe hacer dependiendo del protocolo elegido y dependiendo de esas acciones, escribir o leer en memoria principal, y finalmente la comunicación entre estos dos bloques, la comunicación de varios CPUs y un bus por medio de una cola de mensajes y ejecuciones paralelas.

V. Evaluación de Desempeño de los dos protocolos.

Resultados de pruebas y comparación de los protocolos. A continuación se presentan los diferentes datos estadísticos obtenidos de dos experimentos simulados. Uno para el protocolo MOESI en la figura 9 y otro para 8. Donde se registran las distintas transacciones de WRITE, READ, INCR y el tiempo promedio que dura el procesador,

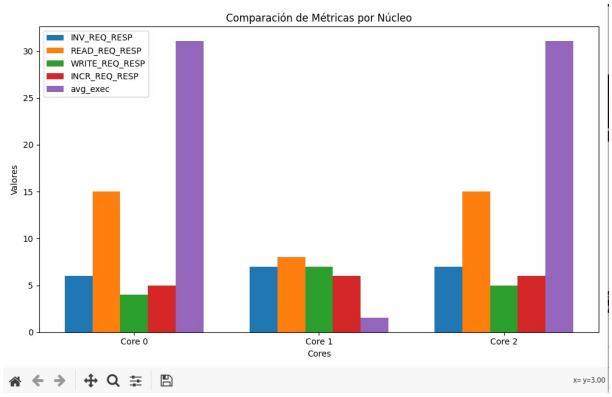

Figura 8: Grafica de resultados de Protocolo MESI

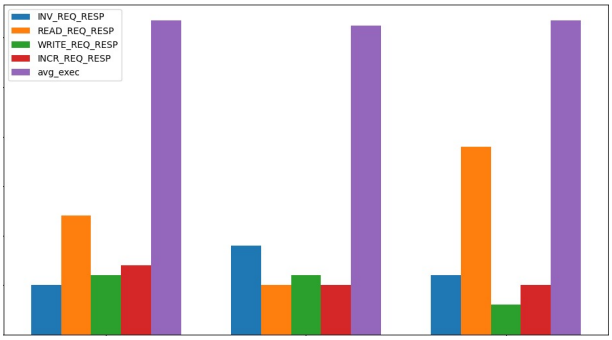

Figura 9: Grafica de resultados de Protocolo MOESI

Análisis de los hallazgos y conclusiones. Principalmente se encuentra que el tiempo promedio de ejecución en cada core sin importar el protocolo se mantiene aldededor de 30s al finalizar este experimento.

VI. Conclusiones y Recomendaciones

- En el protocolo MOESI la carga de instrucciones por CPU no fue simétrica.
- Por otro lado, en el protocolo MESI se ejecutaron las instrucciones de una forma mas homogenea o en una misma cantidad.

VII. Bibliografía

Referencias

[1] Zexin Fu, Yihai Zhang, Mingzi Wang, Zhangxi Tan. *Cache Coherent Framework for RISC-V Many-core Systems*
[2] Quentin L. Meunier, Yann Thierry-Mieg, Emmanuelle Encrenaz. *Modeling a Cache Coherence Protocol with the Guarded.*
[3] Hennessy, J., & Patterson, D. (2017). *Computer Architecture: A Quantitative Approach (6th ed.).*
[4] William Stallings. (2010). *Computer Organization and Architecture (10 ed.).*