WHITE PAPER

TEC | Tecnológico de Costa Rica

Cache Coherence Modelling and Evaluation

Autores

Sebastián Mora Godínez
Computer Engineering Student

Alejandro Chavarría Madriz
Computer Engineering Student

Alejandro Campos Abarca
Computer Engineering Student

Tabla de contenidos

Introducción

La caché, un componente esencial en la jerarquía de memoria de los procesadores modernos, desempeña un papel fundamental en la mejora del rendimiento del procesador al almacenar temporalmente datos y programas utilizados con frecuencia. Su importancia radica en su capacidad para proporcionar acceso rápido a la información, reduciendo significativamente el tiempo de acceso a la memoria principal. Sin embargo, a medida que los sistemas multinúcleo se vuelven cada vez más comunes, surgen problemas de coherencia en el manejo de la caché que deben abordarse.

La coherencia de caché se convierte en un desafío en sistemas multinúcleo debido a la naturaleza distribuida de las cachés individuales en cada núcleo del procesador. Cuando múltiples núcleos acceden y modifican los mismos datos almacenados en caché, es crucial mantener la coherencia para evitar resultados inesperados y errores en la ejecución de programas. Aquí es donde entran en juego protocolos como MESI y MOESI.

Estos protocolos de coherencia de caché proporcionan un marco sólido para garantizar que todos los núcleos vean una imagen coherente de la memoria compartida. MESI, por ejemplo, define cuatro estados principales (Modificado, Exclusivo, Compartido e Inválido) para realizar un seguimiento del estado de los datos en caché y determinar cómo deben gestionarse las operaciones de lectura y escritura. MOESI amplía este conjunto con un estado adicional, "Poseído"(*Owned*), para mejorar aún más la eficiencia y reducir el tráfico de coherencia de caché.

Por lo tanto, dado que la caché es una parte esencial de la arquitectura de computadoras modernas, y su importancia en la aceleración del rendimiento es innegable y los problemas de coherencia presentan una complejidad sustancial, en este documento se plantea y describe un software que permite explorar a fondo los protocolos de MESI y MOESI en una simulación realista un de sistema multinúcleo logrado mediante el aprovechamiento de hilos del procesador. Además es desarrollada una interfaz gráfica que asista a la comprensión de dichos protocolos con un enfoque interactivo.

Proceso de diseño

Frontend

En el contexto del diseño de la interfaz destinada a interactuar con un simulador de protocolos de caché MESI y MOESI, el proceso de toma de decisiones se llevó a cabo con un enfoque cuidadoso y deliberado. El principal objetivo era establecer la plataforma adecuada para la comunicación efectiva con el simulador, considerando los requisitos específicos del proyecto.

La primera y fundamental elección fue determinar si se debía desarrollar la interfaz como una aplicación web o como una aplicación de escritorio. Esta decisión se basó en una evaluación de los requisitos y las capacidades de ambas opciones. La opción de una aplicación web se descartó debido a la naturaleza de las necesidades del proyecto. El simulador requería una cantidad significativa de animaciones para representar de manera efectiva los procesos y datos involucrados en los protocolos MESI y MOESI. Se reconoció que las aplicaciones web, aunque son excelentes para muchos propósitos, no estaban inherentemente diseñadas para manejar animaciones complejas de manera eficiente. En este sentido, Unity, con su enfoque en el desarrollo de juegos y aplicaciones interactivas, emergió como la elección preferida.

Además de su capacidad para manejar animaciones de manera fluida, el equipo ya contaba con experiencia en el uso de Unity, lo que permitía un inicio más rápido y una curva de aprendizaje más suave. Esto fue un factor determinante en la elección, ya que la familiaridad con la plataforma aceleraría el desarrollo y garantizaría una mayor eficiencia en la implementación.

Unity también superó alternativas más rudimentarias, como la utilización de bibliotecas como Tkinter, que, aunque podrían haber cumplido con los requisitos básicos, carecían de las capacidades necesarias para crear una interfaz impactante y llamativa. La sofisticación de Unity y su riqueza de herramientas proporcionaron al equipo las herramientas necesarias para transmitir de manera efectiva el mensaje y los conceptos clave relacionados con los protocolos de caché MESI y MOESI de una manera visualmente atractiva.

También se consideraron dos enfoques para la comunicación entre la aplicación frontend y el backend. Inicialmente, se exploró la posibilidad de mantener una conexión activa mediante un socket TCP/IP para lograr una comunicación en tiempo real. Sin embargo, esta idea se descartó debido a que las especificaciones del proyecto requerían tanto una ejecución paso a paso como una ejecución seguida, lo que complicaba la implementación de una comunicación en tiempo real.

Además, se evaluó la opción de separar por completo el backend y el frontend, estableciendo una comunicación a través de archivos. No obstante, esta alternativa también se descartó, ya que se necesitaba la capacidad de ejecutar múltiples instancias del backend sin interferir con el funcionamiento del frontend. La decisión final se centró en encontrar un equilibrio que permitiera una comunicación eficiente entre ambas partes sin agregar complejidad innecesaria al sistema.

Backend

Para el desarrollo del backend, se llevaron a cabo evaluaciones iniciales de diversas opciones de lenguajes de programación. En un principio, se consideraron Python, C y Rust como candidatos viables. Sin embargo, la decisión final se basó en la necesidad de manejar eficientemente el paralelismo, un aspecto crucial de la especificación del sistema. Inicialmente, Python se descartó debido a la limitación del GIL (*Global Interpreter Lock*) [1], que impide el aprovechamiento de un paralelismo real. Esto dejó a C y Rust como las últimas dos opciones viables. No obstante, se tuvo en cuenta que el equipo de desarrollo carecía de experiencia previa en Rust, lo que implicaría una curva de aprendizaje y un riesgo asociado. Por lo tanto, tras una evaluación cuidadosa, se tomó la decisión de optar por C como lenguaje de programación para el desarrollo del backend.

Para diseñar el sistema MP (*Multiprocessing*), donde se simula el modelo de coherencia de caché, se inició identificando las diferentes componentes que constituyen dicho sistema. Entre las partes identificadas se incluyen los elementos de procesamiento (PE), las cachés individuales de cada PE, el bus de interconexión y la memoria compartida. Durante el análisis de los PEs y las cachés, se notó que ambas entidades están compuestas por conjuntos de variables y constantes, lo que las hace candidatas ideales para ser modeladas mediante estructuras (structs).

Durante el diseño de las cachés, se optó por omitir elementos como el índice y el offset, los cuales son comunes en las cachés modernas. Esto se debió a que tanto la memoria principal como las cachés de los PEs almacenan datos de 32 bits, lo que elimina la necesidad de realizar búsquedas a través de líneas de caché para encontrar los datos.

Además, en un principio se consideró la posibilidad de que el interconnect pudiera gestionar las transacciones de los PEs de manera paralela. Sin embargo, esta opción conllevaría un aumento en la complejidad del diseño del bus. En respuesta a este desafío, se llevaron a cabo análisis adicionales en colaboración con el cliente. Como resultado de este análisis, se optó por una estrategia diferente, que consiste en serializar las transacciones de todos los PEs, simulando una topología de tipo estrella lo cual reduce la complejidad del bus de interconexión.

En el caso de los protocolos, tanto MESI como MOESI se comenzó a diseñar como una máquina de estados finitos, sin embargo, esta máquina solamente contemplaba las transiciones que podían suceden localmente en un PE y no aquellas que podían provenir de otros PEs, es por esto que se introdujo una serie de variables que permiten conocer el tipo de request, si el dato se encuentra en la caché local y si existen otros PEs con el dato de la transacción, de esta manera, es más se pueden abarcar tanto las transiciones que se deben realizar de manera local como aquellas que son causadas otros PEs, lo que permitió agrupar todas las transiciones en 8 casos.

Propuesta de diseño

Frontend

La función principal de la interfaz se centra en la simulación de la ejecución de protocolos de caché MESI y MOESI. Para lograr esto, la interfaz utiliza una conexión de socket TCP/IP para comunicarse con el backend, que es responsable de proporcionar la información necesaria para llevar a cabo las animaciones y la simulación.

El proceso comienza cuando el usuario establece la conexión. Una vez tomada esta decisión, la interfaz envía una solicitud al backend a través del socket TCP/IP para obtener los datos relevantes para la simulación.

El backend responde con un JSON que contiene información detallada sobre la ejecución del programa como el estado de los tres CPU, las tres cachés, las comunicaciones entre ellas, el bus y la memoria RAM. Esta información se utiliza para iniciar las animaciones y permitir al usuario visualizar cómo se ejecutan las instrucciones y cómo cambia el estado de las cachés a lo largo del tiempo. Las animaciones generadas por la interfaz muestran claramente los cambios en las cachés, qué procesador está ejecutando una instrucción en un momento dado y cómo viajan los datos a través del bus. Además, las animaciones representan los cambios de estado de las cachés, lo que ayuda al usuario a comprender el funcionamiento de los protocolos MESI y MOESI de manera visual y efectiva.

En la interfaz también existe un botón permite al usuario reiniciar la ejecución de la simulación en cualquier momento. Esto es útil para volver a observar la simulación desde el principio o para cambiar de protocolo y ver cómo se comportaría en un escenario diferente. También existe un segundo botón permite al usuario realizar una solicitud al backend para generar una nueva ejecución con instrucciones autogeneradas. Esta característica brinda la flexibilidad de explorar diferentes escenarios y comportamientos de protocolo, lo que enriquece la experiencia de aprendizaje y análisis.

Ventajas:

- Representación Visual Clara: La interfaz utiliza animaciones visuales para representar de manera clara y efectiva la ejecución de los protocolos de caché. Esto facilita la comprensión de conceptos complejos.

- Interacción del Usuario: La inclusión de botones que permiten reiniciar la simulación y generar nuevas ejecuciones autogeneradas brinda flexibilidad al usuario para explorar diferentes escenarios y mejorar su comprensión.

- Comunicación Eficiente: La conexión mediante socket TCP/IP permite una comunicación eficiente con el backend, lo que garantiza una respuesta rápida.

- Reutilización de la Interfaz: La interfaz puede utilizarse para simular tanto el protocolo MESI como el MOESI, lo que amplía su utilidad y valor educativo.

- Enfoque Educativo: La interfaz sirve como una herramienta educativa efectiva para estudiantes y profesionales que desean comprender en profundidad los protocolos de caché.

Desventajas:

- Complejidad de Implementación: La implementación de una interfaz tan detallada con animaciones y comunicación por socket puede ser compleja y requerir recursos considerables.

- Requisitos de Recursos: Para un rendimiento óptimo, la simulación puede requerir hardware y recursos de cómputo relativamente potentes.

- Dependencia del Backend: La funcionalidad de la interfaz depende en gran medida del backend y de la calidad de los datos proporcionados en el JSON. Cualquier problema en el backend puede afectar la experiencia del usuario.

- Posible Sobrecarga Visual: La abundancia de detalles visuales y animaciones puede, en algunos casos, distraer o abrumar a los usuarios, dificultando la focalización en conceptos específicos.

Backend

El sistema MP estará compuesto por tres PEs (*Processing Unit*), cada uno de los cuales estará equipado con un conjunto de instrucciones y un registro. Además, cada PE estará asociado con una caché que constará de 4 entradas de datos de 32 bits. Cada línea de caché se compondrá de un bit de validez, un bit de suciedad, un tag, un estado y el dato almacenado. Todos los PEs se ejecutarán de manera concurrente utilizando threads y se comunicarán con el interconectado a través de una cola de mensajes proporcionada por el mismo. Por otra parte, se determinó que la caché tendrá una asociatividad directa y un política de reemplazo que determina la línea a reemplazar utilizando la siguiente operación

$$línea\ a\ reemplazar\ =\ Tag\ \ mod\ \ \#entries$$

Las instrucciones de los PEs se serializarán, lo que significa que cuando un PE desee ejecutar una transacción, enviará un mensaje a la cola de mensajes con la información necesaria y se procesarán conforme sea agregados a la cola. Una vez que el PE haya enviado el mensaje, quedará bloqueado hasta recibir una respuesta del bus de interconexión. La sincronización entre los threads de los PEs se llevará a cabo mediante el uso de semáforos.

El bus de interconexión, que se ejecutará en su propio thread, estará constantemente monitoreando la cola de mensajes. Cuando un mensaje llegue a la cola, el bus procesará la transacción asociada, realizando las modificaciones necesarias en las cachés locales, en otras cachés o en la memoria RAM. Una vez finalizada la transacción, el bus liberará al PE asociado para que pueda continuar con su procesamiento.

Tanto los protocolos MESI como MOESI se representarán mediante máquinas de estados que evaluarán las condiciones de la transacción para determinar las modificaciones y transiciones necesarias. Estas condiciones incluyen el tipo de transacción (lectura o escritura), la presencia del dato

en la caché local y la existencia del dato en otras cachés, permitiendo agrupar todas las transacciones en 8 casos y reutilizar dicha lógica para ambos protocolos.

Tipo de transacción	Caché local posee el dato	Cachés remotos poseen el dato	MESI	MOESI
Lectura	×	×	I-E	I-E
Lectura	×	✓	I-S	I-S
Lectura	✓	×	M-M E-E S-S	M-M E-E S-S
Lectura	✓	✓	S-S	S-S O-O
Escritura	×	×	I-M	I-M
Escritura	×	✓	I-M	I-M
Escritura	✓	×	E–M M-M S-M	E-M M-M S-M O-M
Escritura	✓	✓	S-M	S-M O-M

Tabla 1. Tabla con los casos evaluados para llevar a cabo las posibles transiciones.

Ventajas

- La serialización permite reducir la complejidad de la implementación del bus de interconexión, de este modo, se asegura que el bus ejecuta una transición a la vez.
- El uso de los diferentes casos para evaluar la transición y los cambios a realizar según corresponda ayuda a agrupar las diferentes transiciones haciéndolas más sencillas de identificar.

Desventajas

- La serilización reduce el rendimiento del sistema, pues obliga a que las instrucciones se ejecuten secuencialmente, provocando que se pueda dar el caso en que una instrucciones que se puede ejecutar perfectamente, tenga que esperar hasta que le toque su turno.

Especificación de componentes

Frontend

La interfaz de simulación de protocolos de caché MESI y MOESI se construye con un utilizando GameObjects como bloques de construcción fundamentales. Cada componente principal, como las CPU, las Cachés, el Bus, la RAM y las interconexiones, se implementa como GameObjects. Estos GameObjects poseen componentes de script que modelan su comportamiento y funcionalidad específica. Además, incluyen otros componentes, como imágenes y efectos de iluminación, para lograr una representación visual atractiva.

Los scripts de cada componente exponen funciones públicas que permiten, por ejemplo, modificar texto relacionado a líneas de almacenamiento de caché o mover componentes asociados para generar animaciones.

El control integral de la ejecución recae en un GameObject central denominado "GameMaster". Este componente es esencial para el funcionamiento de la simulación y realiza varias funciones clave. Primeramente establece la comunicación con el backend a través de la conexión TCP/IP, solicitando y recibiendo datos esenciales en formato JSON que se utilizarán en la simulación. Una vez que se recibe el JSON, el GameMaster realiza el análisis y procesamiento de los datos para extraer la información necesaria para la simulación. Estos datos incluyen el estado de las CPU, las Cachés, el Bus y la RAM para la ejecución de cada instrucción, así como otra información relevante para la animación. También expone funciones públicas que se asocian con los botones de la interfaz. Por ejemplo, la función de reinicio permite reiniciar la simulación.

En conjunto, esta estructura basada en GameObjects y el control central del GameMaster permiten una simulación detallada y precisa de los protocolos de caché MESI y MOESI. Cada componente interactúa según su comportamiento modelado, lo que resulta en una representación efectiva y visualmente rica de la ejecución de estos protocolos.

Backend

El bus de interconexión se ejecuta de manera continua, monitoreando constantemente la cola de mensajes. Esta cola actúa como un punto de reunión para todas las transacciones que los PEs desean ejecutar, y las serializa en orden. Una vez que se detecta un mensaje en la cola, se extrae de la misma y se procede a ejecutar el protocolo correspondiente utilizando la información proporcionada por la transacción.

Los *processing elements* o PEs, se han modelado a partir de una estructura que reúne las propiedades esenciales de este componente, junto con otras necesarias para el flujo del programa. En primer lugar, contamos con un identificador único para cada núcleo (core); a través de este identificador, podemos rastrear el origen de la instrucción en ejecución en el bus de interconexión. Cada PE se asocia con un conjunto de instrucciones generadas aleatoriamente, y se le asigna un límite máximo de hasta 10 instrucciones. Además, encontramos un flag, que permite identificar si el bus de interconexión se

encuentra ejecutando una instrucción de un PE específico, lo que, en caso afirmativo, bloquearía la ejecución de otras instrucciones para ese PE. También existe otro flag que señala el momento en que se completa el procesamiento de todo el conjunto de instrucciones asociado a un PE.

Las cachés, al igual que los PEs, se han modelado utilizando structs que incluyen varios componentes esenciales. Estos componentes consisten en el tag (etiqueta), que representa la dirección de memoria del dato en la caché, un bit de validez que indica si el dato en la caché es válido, un bit de suciedad (dirtiness) que refleja si el dato ha sido modificado desde la última vez que se leyó de la RAM, el estado de la línea de caché, según el protocolo utilizado, y, finalmente, el dato almacenado en la línea de caché.

La memoria RAM se representa mediante un arreglo de 16 números enteros, lo cual representa las 16 *entries* de acuerdo con la especificación brindada. Cada *entry* almacena datos de 32 bits. El índice dentro del arreglo corresponde directamente a la dirección de memoria.

Evaluación de desempeño: MESI vs MOESI

Para evaluar ambos protocolos, se recopilaron los resultados de diversas métricas a lo largo de un total de 10 ejecuciones con diferentes set de instrucciones, en cada una de las ejecuciones se utilizaron un set de 20 000 instrucciones para cada PEs. Posteriormente, se calculó el promedio de estas métricas con el objetivo de obtener una visión más equilibrada del rendimiento de ambos protocolos. A continuación se muestran los resultados obtenidos.

Métricas	**Protocolo**	
	MESI	**MOESI**
Increments	20043	20043
Read Requests	19987	19987
Write Requests	19970	19970
RAM writes	18226	17774
RAM reads	9115	9183
Invalidations	32039	32068
Shares	6892	6843

Tabla 2. Tabla comparativa de diferentes métricas para evaluar el desempeño de los protocolos MESI y MOESI.

Como se puede observar, la métrica en la que se destaca la mayor diferencia entre ambos protocolos es la cantidad de escrituras a la memoria RAM, donde MOESI presenta un menor número en comparación con MESI. La introducción del estado *Owned* permite compartir valores cuando los

datos compartidos están sucios, lo que reduce la necesidad de actualizar la memoria, lo cual es consistente con el resultado mencionado anteriormente.

En cuanto al resto de las métricas, se obtuvieron valores similares, y cualquier diferencia que exista no es muy notoria, lo cual podría atribuirse a la aleatoriedad de las instrucciones ejecutadas.

Métrica adicional para la evaluación del desempeño de protocolos

Además, se diseñó una métrica adicional para evaluar el rendimiento de ambos protocolos. Esta métrica consiste en medir el tiempo total que los protocolos dedicaron a acceder a la memoria RAM. Para calcular este tiempo, se sumaron tanto las operaciones de lectura como de escritura realizadas por ambos protocolos y se utilizó un valor de referencia para el tiempo de acceso a la memoria RAM.

Es importante destacar que el tiempo de acceso a la memoria RAM puede variar en función de diversos factores. Sin embargo, después de una investigación exhaustiva para determinar un valor que se aproxime a un estándar común, se ha establecido un tiempo de acceso a la RAM de 7.2 nanosegundos.

Tiempo de acceso a RAM: 7.2 ns	**Protocolo**	
	MESI	**MOESI**
Lecturas y escrituras a RAM	27341	26957
Tiempo total de acceso a RAM (ms)	0.196	0.194

Tabla 3. Tabla comparativa del tiempo en memoria RAM para los protocolos MESI y MOESI.

Como se puede observar en la tabla, la diferencia entre ambos protocolos es de $2\mu s$ favorable para el protocolo MOESI.

De lo anterior, podemos observar que la implementación del protocolo MOESI mejora el rendimiento del sistema, ya que reduce la cantidad de escrituras en la memoria RAM. Como resultado, se reduce el tiempo necesario para acceder a la memoria RAM.

Referencias

[1] Python.org. "Global Interpreter Lock." WikiPython. [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock. [Accessed: October 3, 2023]
[2] Sheu. “NON-VOLATILE MEMORY SOLUTIONS. [Online]. Available: https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/5746281 [Accessed: October 4, 2023]