Proyecto 1
MOESI Simulator: Modelo de protocolo para coherencia de caché en sistemas multiprocesador

Jose Alejandro Ibarra Campos
email: josealejandroibarra@hotmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—One of the main factors to consider in designing multi-processing computational systems is the coordination between different instances, especially when each core performs different tasks. Two pretty common operation in these types of workloads are the read and write operations, which involve the manipulation of physical memory. Most computers include at least a fast cache memory and a slower RAM (random access memory). In multi-core systems, there is a significant challenge in ensuring the coherence between the cache and the main memory, as many instances may be performing operations using the same cache data lines. This project presents a high-level UI simulator developed with Python and tkinter. The simulator serves as an easy way to see what's going on in a system composed of four cores, each with four lines of L1 cache, as well as eight blocks of global main memory of sixteen bits each. The MOESI cache coherence protocol will be implemented graphically as well, in order to verify that consistency is maintained between the cache lines. The app provides an interactive simulator that's simple to follow and understand for veterans in multi-processing and new-comers alike.**

***Palabras clave*—cache, coherence, moesi, multiprocessing**

I. Introducción

I-A. Contexto del problema

La coordinación entre los diferentes núcleos es uno de los retos más importantes en el desarrollo de sistemas con multi-procesos. Aquí, cada instancia de tipo procesador tiene la tarea de ejecutar sus instrucciones de forma correcta, y muchas veces requiere de leer o escribir datos en memoria para cumplirlas. Los conceptos generales de la arquitectura de computadores muestran la existencia del concepto de **pared de memoria**, que se refiere al fenómeno de que la velocidad de computación avanza más rápido que la velocidad de acceso a memoria. [1]

Las consecuencias de este concepto se han visto desde hace varias décadas, por lo que el diseño de estos sistemas ahora contempla una jerarquía de memoria, con diferentes tipos de esta según sea la aplicación. El truco está en balancear la rápida (cara) y la lenta (más barata) con base en los diferentes aspectos del diseño y la implementación.

Ahora, cuando se introducen múltiples instancias vivas de procesadores, cada una operando por aparte, se debe garantizar que la información sea coherente. Es decir, debe existir algún mecanismo para asegurar que los datos que estos utilizan en sus operaciones sean los adecuados, con el fin de evitar problemas más graves con su aspecto funcional. Para esto se utilizan diferentes protocolos que se encargan de realizar una transición de estados entre las diferentes líneas de caché, la memoria de más fácil acceso para cada procesador. Para este caso, se tomará como base el protocolo MOESI, uno de los más complejos pero utilizados a nivel industrial, que incluye cinco posibles estados de coherencia por línea. [2]

Se implementará un modelo de alto nivel llamado *MOESI Simulator*: un sistema multi-hilo en *Python* con interfaz en *tkinter*. Este contará con cuatro procesadores, cada uno con un controlador de caché y cuatro bloques de caché L1 con asociatividad *two-way*. Estos bloques están compuestos de cuatro líneas, en las que almacenan el estado del protocolo MOESI, dirección (binario de tres bits) y dato (hexadecimal de cuatro bits). El funcionamiento será de tipo simulador interactivo, con modos de ejecución aleatoria simultánea o una instrucción a la vez. Además, se podrán introducir instrucciones específicas para verificación.

I-B. Organización del documento

A continuación se presentará el orden de las siguientes secciones del documento. Primero se realizará una discusión acerca de los conceptos teóricos necesarios para una adecuada comprensión de la aplicación, en conjunto con aquellos relacionados con la arquitectura de computadores para sistemas multi-procesos.

Luego, un resumen del sistema desarrollado (aplicación gráfica), así como su funcionamiento general en los diferentes modos de operación. Finalmente, los apartados de resultados y conclusiones dejarán ver los principales hallazgos de este proyecto y su posterior análisis dentro del marco de diseño de sistemas computacionales desde el punto de vista ingenieril.

II. Marco teórico

MOESI es un protocolo de coherencia de caché (*cache coherence protocol*) para sistemas multi-procesos. Es similar al protocolo MESI o MSI, pero con un estado más que MESI y dos más que MSI. Cada letra hace referencia a uno de los cinco posibles estados para la línea de caché.

Importante recordar que para efectos prácticos, en este caso, una línea (sin estados) está compuesta por **dirección de memoria/*address*** y **dato/*data***. Además, una **copia** se refiere

a una línea de caché en un estado diferente a inválido (I) que se encuentre en otra instancia/procesador.

Abajo se describe rápidamente el significado de cada estado: [2]

- **M - *Modified*:** La versión más actualizada de la línea está en este cache. No existe ninguna copia en ningún otro caché. Este dato es diferente del de memoria.
- **O - *Owned*:** Está línea puede estar en más de un caché. Sin embargo, contiene la versión más actualizada del dato. Similar a M, solo la línea en un núcleo puede estar en este estado, los demás deberán hacerlo en el estado S.
- **E - *Exclusive*:** Esta línea de caché tiene el mismo dato que la memoria principal. No existe ninguna copia en ningún otro caché. Sucede cuando se lee directamente.
- **S - *Shared*:** La línea en este estado es coherente con la memoria principal. Pueden existir copias en otros cachés. El dato se comparte con otros procesadores.
- **I - *Invalid*:** La línea es inválida. Sigue varias reglas:
  - Si se hace un *write* y no está en M o E, todas las copias deben ser invalidadas, ya que estas no poseen el dato más actualizado.
  - El sistema puede descartar una línea inválida para hacer espacio que contenga otra dirección, dependiendo de la política de reemplazo para *cache lines*.
  - Si una línea se encuentra en estado M, las lecturas de otros cachés obtienen el dato actualizado y la que la compartió pasa a O (dueño).
  - Si una línea se encuentra en estado E, las lecturas de otros cachés pasan a S, y la que la compartió pasa a S (compartiendo).

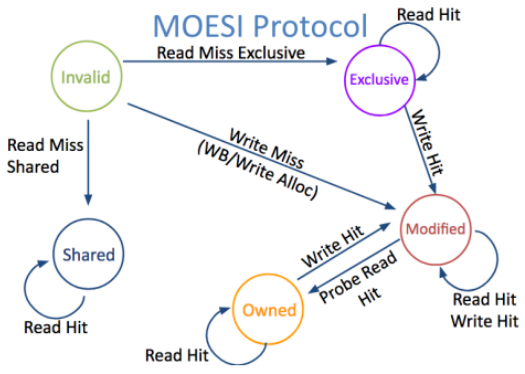

Figura 1: Máquina de estados de línea de caché para protocolo MOESI. [3]

En la figura 1 se aprecia de forma más gráfica las transiciones entre los estados del protocolo MOESI, junto con las condiciones para realizar cada cambio.

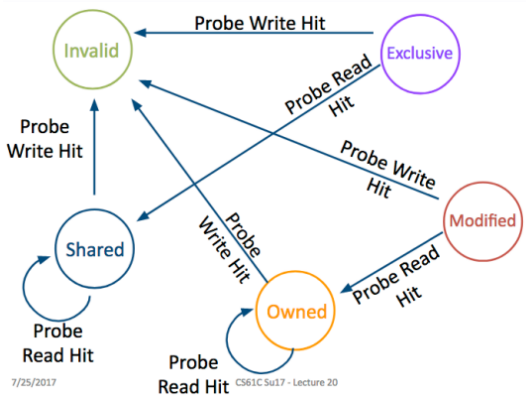

Figura 2: Máquina de estados de bus para protocolo MOESI. [3]

Además, para comunicar las instancias se debe considerar un componente bus, encargado de transmitir la información entre los procesadores y memoria. Este también tiene asociada su máquina de estados, visible en la figura 2. Como se observa, el bus permite comunicar los controladores de caché para realizar actualizaciones sobre las otras instancias, las cuales son necesarias dependiendo de la transición. Por ejemplo, si una línea pasa de S a M, debe notificar a las demás instancias que coloquen sus líneas en I.

III. Sistema desarrollado

Para modelar el sistema *MOESI Simulator* se utilizó el lenguaje de programación de alto nivel *Python*, en conjunto con la biblioteca gráfica incluida: *tkinter*. Internamente, se aprovechó la flexibilidad de los objetos y la sintaxis de estas herramientas para proveer una interfaz sencilla de utilizar y capaz de ser ejecutada de forma correcta en múltiples plataformas.

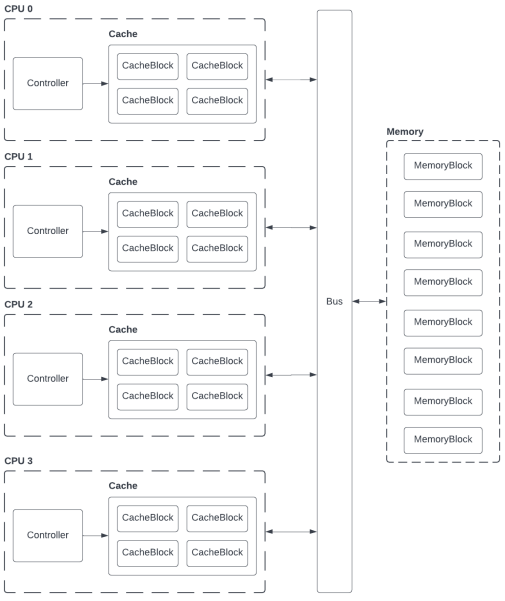

Figura 3: Diagrama de bloques general del sistema *MOESI Simulator*.

En la figura 3 es posible observar el diagrama de bloques para este sistema, consistente con el descrito en el primer apartado de este documento. Cada CPU tiene un controlador de caché, encargado de realizar las operaciones según la lógica de MOESI. También existe una conexión bidireccional entre los CPUs y el bus, y este de la misma manera con memoria.

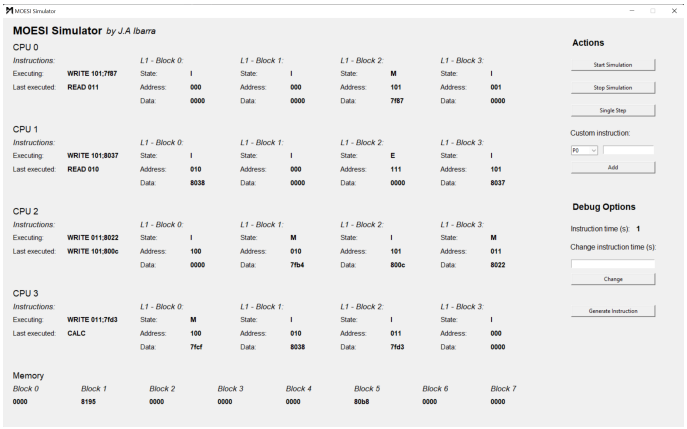

Figura 4: Interfaz de *MOESI Simulator* (en ejecución simultánea).

Luego, en la figura 4 se aprecia la interfaz gráfica del simulador. Aquí es posible interactuar con los diferentes modos de una forma sencilla. Esta muestra claramente un CPU por fila: la instrucción en ejecución, última instrucción ejecutada y cada bloque de caché con sus datos. En la parte de abajo se ve el contenido actual de la memoria.

A la derecha se obtiene un menú de acciones, con los modos de ejecución simultánea: cada procesador genera instrucciones aleatorias y las ejecuta, modo *single-step*: solo se ejecuta un ciclo de generación aleatoria, modo *custom instruction*: se puede introducir manualmente una instrucción para alguno o varios procesadores en el *entry*.

También es posible probar el generador de instrucciones aleatorias, así como modificar el tiempo base que dura en ejecutar una instrucción. Esto es importante porque el simulador incluye una simulación de la penalidad de memoria, para apreciar los efectos en el costo (tiempo) de ir a memoria a leer o escribir un dato.

IV. Resultados y análisis

A continuación se realizará una prueba rápida de *MOESI Simulator*, con ayuda del modo *single-step*, para observar la coherencia de las líneas de caché según el protocolo MOESI.

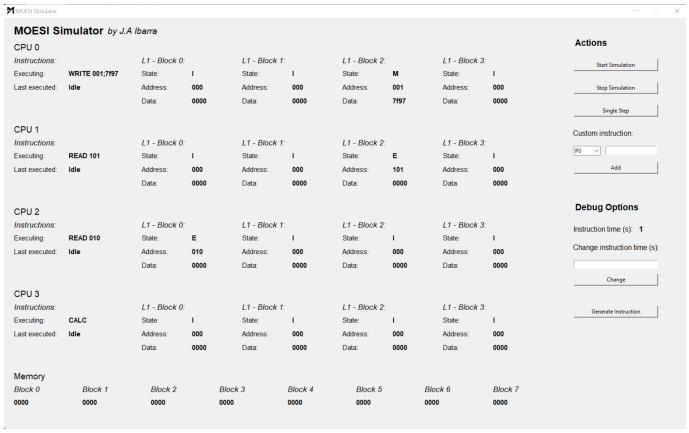

Figura 5: Interfaz de *MOESI Simulator* (*Single-step 1*).

En la figura 5 se observa la primera ejecución de las instrucciones aleatorias. Para el CPU0, se inicia con un *write* en la posición 001, que se ejecuta correctamente en el bloque 2 con estado M, según la asociatividad *two-way*. Luego, los procesadores CPU1 y CPU2 hacen *read* de 101 (impar) y 010 (par). Ningún otro procesador tiene el dato, por lo que debe ir a memoria a leer el 0 y poner el estado en E. Finalmente, el CPU3 hace CALC, una manera de simular alguna ejecución extra (no relacionada con memoria)

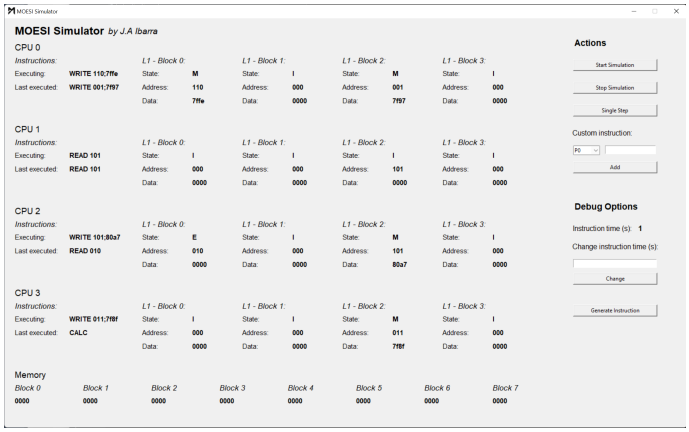

Figura 6: Interfaz de *MOESI Simulator* (*Single-step 2*).

En la segunda ejecución de la figura 6, el CPU0 escribe (estado en M), el CPU2 escribe en 101, por lo que el CPU1 termina con su línea 2 de 101 inválida. Luego, el CPU3 escribe en 011.

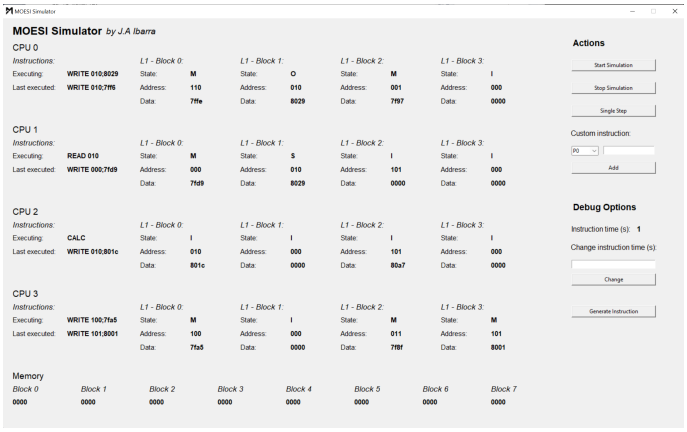

Figura 7: Interfaz de *MOESI Simulator* (*Single-step 4*).

Para la cuarta ejecución (figura 7), se aprecia que el CPU1 lee 010, por lo que pasa a estado S, ya que lo lee del CPU0 que lo acaba de modificar (este pasa a O). También se observa que el *write* del CPU0 sobre 010 invalida la línea 0 del CPU2.

De esta manera se observa el comportamiento correcto del simulador en los diferentes estados del protocolo MOESI. Una de las ventajas de dicha aplicación es que permite tener una mejor perspectiva de lo que está ocurriendo internamente dentro de un sistema de este tipo, ya que los hilos de *Python* permiten crear la ilusión de procesamiento paralelo para el usuario. Además, es posible interactuar de diferentes maneras con la interfaz y ver su respuesta ante diferentes casos.

V. Conclusiones

A través del desarrollo de este proyecto se notaron diferentes aspectos a tomar en cuenta cuando se trabaja con sistemas multi-proceso y modelado de los mismos.

Se debe prestar especial atención a las estructuras de las herramientas a utilizar y el objetivo deseado. En este caso, se utilizó un lenguaje de alto nivel, lo que brindó gran facilidad a la hora de codificar la solución. En términos prácticos, para el alcance que tiene este proyecto de comprobación académica, no es necesario complicarse con herramientas más avanzadas.

Otro detalle importante a manera de recomendación es la realización de esquemas o diagramas de manera previa que ayuden a comprender mejor el funcionamiento del sistema en general. Siempre es relevante mencionar este punto, ya que uno de los aspectos más importantes de la ingeniería consiste en un adecuado entendimiento del problema, así como el planteo de soluciones efectivas. Aquí, la visualización gráfica de los flujos y comunicación entre estructuras es de gran ayuda tanto para el desarrollador como para potenciales clientes o usuarios no versados en los conceptos complejos. Para *MOESI Simulator* se tomó como base un diagrama de bloques general (figura 3) y luego se expandió según las herramientas seleccionadas para la implementación.

Además, por la naturaleza del modelado, es vital mantener un contexto que permita alcanzar el resultado (observar la transición de estados en las líneas de caché) con el funcionamiento interno del sistema (interacciones entre objetos/memoria). Para alcanzar este punto, se requiere de una adecuada investigación y conocimiento de ambos marcos: hardware y software. Este puente es fundamental para la carrera de Ingeniería en Computadores y el desarrollo del ingeniero como tal, ya que no solo incluye una exhaustiva comprensión del problema, sus ejes, aristas, y situaciones, sino que inscribe en conjunto con este diversos detalles que permiten la fusión de conceptos que de otra manera no sería posible. En este caso, la aplicación *MOESI Simulator* es nada más una aproximación de cómo es en realidad un sistema multi-procesador con MOESI, porque internamente utiliza técnicas como hilos y manejo libre de memoria, que claramente son más avanzadas. Sin embargo, la ilusión es lo suficientemente convincente para entender el concepto.

Referencias

[1] Clements, S. (2014) What Is A Memory Wall? [En línea] Disponible en: https://www.offtek.co.uk/blog/what-is-a-memory-wall/ [Accedido: 29-Mar-2022].

[2] ARM Developer. (s.f) MESI and MOESI protocols [En línea] Disponible en: https://developer.arm.com/documentation/den0013/d/Multi-core-processors/Cache-coherency/MESI-and-MOESI-protocols [Accedido: 29-Mar-2022].

[3] Berkeley. (s.f) MOESI Cache Coherency [En línea] Disponible en: https://inst.eecs.berkeley.edu/~cs61c/su18/disc/11/Disc11Sol.pdf [Accedido: 29-Mar-2022].