Proyecto Individual 1: Protocolos de Coherencia en Sistemas Multiprocesador

Alex-Manuel Marín-Lopez
email: manuelmarinlopez07@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Abstract*—Modern processors take advantage of parallelism to increase performance. Instruction Level Parallelism (ILP) is one of the techniques available, however ILP presents difficulties that the systems has to solve, such as issue rates, cache misses and off-chip misses. Thread Level Parallelism (TLP) is another approach or technique available, but it comes with the downside that designing multi-threaded processors require additional considerations. The cache coherence problem is one of said considerations and is manageable by applying "snooping". Snooping can be implemented utilizing various protocols, such as MOESI. In this article we explore an implementation of a MOESI cache simulator in python.**

***Palabras clave*—Linux, MOESI, Cache, Parallel Programming, Threads, Multiprocessor.**

I. Introducción

Las arquitecturas multiprocesador poseen consideraciones inherentes a ellas, por ejemplo la coherencia de cache, el cual será el punto fuerte de este proyecto. Cada procesador, aun cuando estos cuentan con una memoria principal la cual es compartida, posee cache individual, de manera que sin las precauciones necesarias la información que se encuentra tanto en las caches como en memoria principal se puede encontrar lo que se denomina sucia, es decir valores distintos en cada sector posible. En el presente documento se explicará más a detalle el problema de coherencia y las posibles soluciones para el mismo.

En primer plano se analizan los problemas de coherencia de cache y consistencia de datos, para posteriormente introducir el protocolo de coherencia de cache MOESI. Seguidamente se justifican las distribuciones de probabilidad elegidas y finalmente se detalla la implementación de la solución definida acompañada de los resultados y análisis de estos.

II. Marco Teórico

A. Cache multihilo

La coherencia de caché define el comportamiento de las lecturas y escrituras en una misma ubicación en memoria, mientras que la consistencia define el comportamiento de las lecturas y escrituras con respecto a los accesos a otras ubicaciones en memoria[3].

Para garantizar la coherencia de caché, se debe mantener control del estado de cualquier bloque compartido. Existen dos formas de hacer esto[3]:

- Técnica basada en directorio: el estado de un bloque de la memoria física se mantiene en un unico lugar llamado **directorio**.
- Técnica de monitoreo(*snooping*): cada caché que posee una copia de la información de memoria física, también tiene una copia del estado del bloque.

B. Protocolo MOESI

El enfoque del presente documento se encuentra basado en el protocolo de coherencia MOESI. En este, se definen 5 posibles estados para un bloque de caché: *Invalid, Shared, Modified, Exclusive y Owned*.

El estado *Invalid* indica que el bloque de caché no cuenta con la información mas reciente para el espacio de memoria deseado.

Exclusive indica cuando un bloque de caché reside únicamente en una caché y se encuentra limpio. Si un bloque se encuentra en este estado, puede escribirse sin generar ninguna invalidación[3]. Cuando otro procesador incurre en un read miss, el bloque en este estado debe cambiar de estado a Shared para mantener coherencia[3].

El estado *Shared* permite determinar que el bloque en la caché se encuentra potencialmente compartido, por otra parte el estado *Modified* indica que el bloque ha sido actualizado en caché privada.

Por último, el estado *Owned*. Un bloque en este estado indica que el bloque es válido, de solo lectura, sucio y la caché es la dueña de este bloque. Esto último implica que la caché se encuentra a cargo de responder peticiones de coherencia del bloque.

El diagrama de estados de la figura 1 presenta la máquina de estados finitos de este protocolo.

C. Distribuciones de Probabilidad

El proyecto hace uso de dos distribuciones de probabilidad con el fin de evitar la función "random":

- Distribución Normal
- Distribución de Poisson

1) Distribución de Poisson: Dado un intervalo de numeros reales, suponga que ocurren conteos al azar a lo largo del intervalo en subintervalos con una longitud suficientemente pequeña tal que:

1) La probabilidad de más de un conteo en subintervalo es cero.

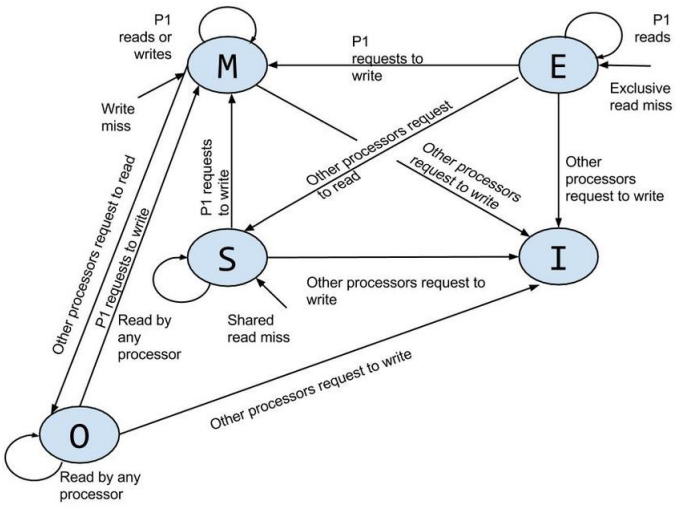

Fig. 1: Diagrama de estados finitos MOESI

2) La probabilidad de un conteo en un subintervalo es la misma para todos los intervalos proporcional a la longitud del intervalo.
3) El conteo para cada subintervalo es independiente de los demas subintervalos.

De cumplirse las condiciones, el experimento aleatorio se denomina proceso de Poisson [2]. Si el numero promedio de conteos en el intervalo se define como $l > 0$; la variable aleatoria X, que es igual al número de conteos en el intervalo, tiene una **distribucion de Poisson** con parametros l, y la distribución de masa de probabilidad de X es [2]:

$$f(x) = \frac{e^{-\lambda}\lambda^x}{x!}, x = 0, 1, 2, ...$$

Fig. 2: Función de masa de probabilidad de X

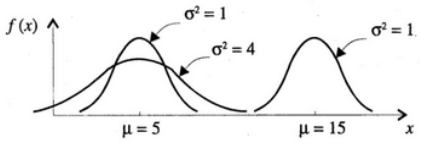

Fig. 3: Funciones de densidad de probabilidad normal para valores de los parametros

2) Distribucion Normal: Sea una variable aleatoria X con función de densidad de probabilidad:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Fig. 4: Función de densidad de probabilidad Normal

Dicha variable aleatoria, tiene una **distribución normal** [2].La figura 3 demuestra este concepto.

III. Sistema Desarrollado

En la etapa inicial de desarrollo se plantean dos posibles soluciones. Inicialmente se analiza la posibilidad de una solución web utilizando Angular o React como framework de trabajo. La idea consiste en aprovechar la estructura de componentes que proveen estos frameworks para simular concurrencia. Tras un estudio de esta solución se determina el camino a seguir: Python. La segunda opción corresponde a una aplicación de escritorio desarrollada en Python, con una interfaz basada en Tkinter. La base de la decisión corresponde a los potenciales problemas de sincronización que se pueden presentar al desarrollar la aplicación con componentes web, por lo que se determina que el uso de hilos es mejor (más natural). La figura 6 presenta el diagrama de clases de la solución. En este se pueden apreciar los componentes desarrollados para simular el protocolo MOESI:

- Processor: La clase procesor se encarga de representar un procesador en el sistema. De esta se realizan, en este caso, 4 instancias. Cada instancia de Processor cuenta con su propio ID, cache, Status, *Current Instruction*, entre otras características funcionales. Además el procesador cuenta con una referencia al bus para llevar a cabo el snooping.
- Bus: La clase Bus cuenta con una referencia a la memoria principal del sistema, un log de instrucciones y una referencia al array de procesadores creados. El bus es el encargado de portar la señal de cache miss o invalidación según corresponda.
- Memory: La clase memoria: Se utiliza para la memoria principal del sistema. La clase Memory cuenta con 8 bloques que representan las direcciones físicas de memoria del sistema y métodos para escritura y lectura sobre ella misma.

Así mismo se define el diagrama de bloques de la figura 5, que muestra las conexiones del sistema de acuerdo a lo definido en la especificación del proyecto.

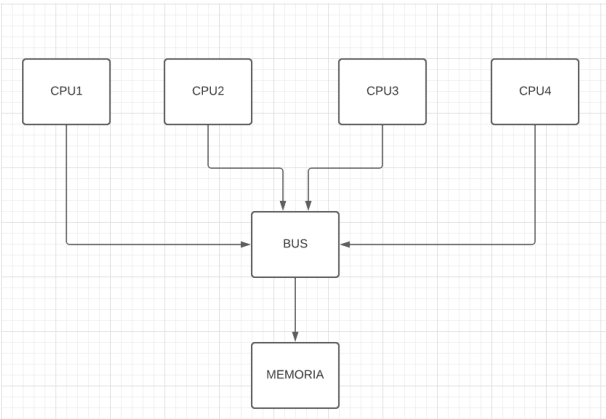

Fig. 5: Diagramas de Bloques del sistema

Como se menciono anteriormente, la interfaz se desarrolla en TKinter. Para esto se define un *layout* similar al diagrama de bloques, obviando el bus por temas de espacio. El *layout* muestra los 4 procesadores de manera simultánea, cada procesador a su vez muestra información como los bloques de cache, un mensaje de controlador,la última instrucción ejecutada y la

instrucción en ejecución. Adicionalmente se coloca un *"Log"* de instrucciones para facilitar el monitoreo de la coherencia en el sistema. Por su parte, la memoria muestra el valor de las direcciones físicas disponibles para los procesadores.

La figura 7 muestra el concepto de *layout*.

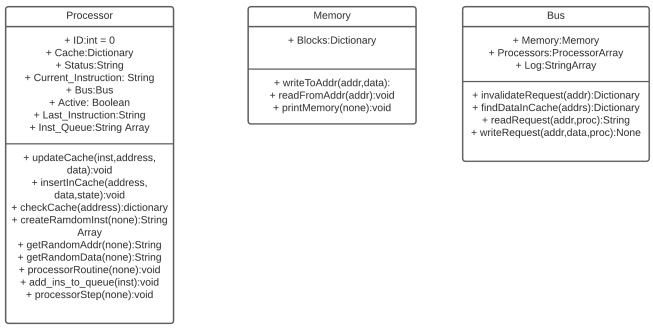

Fig. 6: Diagramas de Clases del sistema

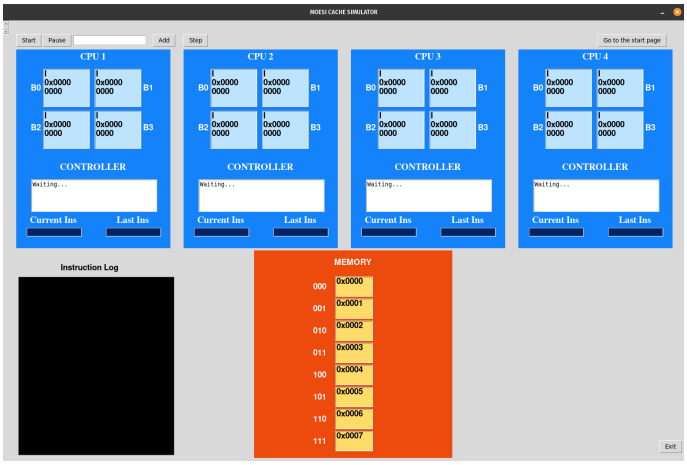

Fig. 7: Layout del programa

Cabe destacar que los procesadores cuentan con funcionalidades especiales como el método *step( )*, que se encarga de ejecutar la rutina del procesador 1 vez y el método *addInst( )* que permite añadir una instrucción al procesador en el momento deseado.

IV. Resultados y Analisis

Las distribuciones de probabilidad presentan un sin fin de posibilidades en la solución de problemas ingenieriles. Por ejemplo, utilizar la distribución normal para la generación de instrucciones permite definir el tipo de instrucción recurrente. Es decir, en caso de querer minimizar el uso de Bus o aumentar la frecuencia de alguna instrucción en particular, esto se puede realizar con facilidad a través de la implementación de la distribución normal. De manera similar, la distribución de Poisson presenta características que la vuelven un buen candidato para la generación de direcciones aleatorias.

Por su parte, el protocolo MOESI cumple con su objetivo de aumentar coherencia y consistencia en el sistema. La adición de los estados Exclusive y Owned al protocolo MSI reducen el acceso a memoria y el uso de bus de manera considerable. Sin embargo, aun cuando se está logrando reducir el costo por acceder a memoria, MOESI es incapaz de eliminar el problema de pared de memoria en los casos de acceso, esto se ve reflejado en la espera que se debe realizar por parte de cada procesador cuando estos se topan con estas circunstancias. Por supuesto, en un sistema multiprocesador real existiría compensación para este problema en forma de otros procesadores trabajando mientras el procesador N accesa memoria.

En cuanto a arquitectura de sistemas, el proyecto refuerza conceptos y prácticas de programación y desarrollo de software. El producto final es completamente funcional con una única limitación en la pausa de los hilos dado que se utilizó Tkinter y esta herramienta se caracteriza por ser single-threaded. No obstante, se considera que el resultado final cuenta con las características solicitadas durante la asignación.

Atributos destacables de la implementación realizada son la escalabilidad y adaptabilidad. El proyecto está diseñado de manera que escalar a cualquiera tamaño sea sencillo y no consuma mucho tiempo. Así mismo, el proyecto es capaz de adaptarse a cualquier ambiente en el que se pueda ejecutar el programa de python sin mayores limitaciones o consideraciones adicionales.

V. Conclusiones

Los protocolos de coherencia representan el fundamento de los sistemas multiprocesadores con TLP. Estos permiten mantener coherencia y consistencia manteniendo en mente que en el mundo real los recursos son finitos y deben ser utilizados de manera eficiente y eficaz en todo momento.

Aun cuando Python es la mejor opción para realizar programas en muchos ámbitos de las ciencias de computación, la herramienta Tkinter presenta fuertes limitaciones en temas de paralelismo y uso de hilos. La solución proporcionada podría ser mejorada con una interfaz desarrollada en un ambiente más amigable a *multithreading*.

References

[1] S. Dey and M. Nair, "Design and implementation of a simple cache simulator in java to investigate mesi and moesi coherency protocols," International Journal of Computer Applications, vol. 87, 01 2014.

[2] D. Montgomery and G. Runger, Probabilidad y estadistica aplicadas la ingenieria, ser. Matematicas, Probabilidades y Estadıstica. Limusa Wiley,2002.

[3] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, ser. ISSN. Elsevier Science, 2011.