

# SW \$ Coherence modelling and evaluation

Cristian Marín, Cristhofer Azofeifa, Emily Sancho, Karla Rivera

**Abstract**—The main purpose of this project is to model and evaluate cache coherence protocols in multiprocessor (MP) systems. To support this work, it is based on the theoretical fundamentals of cache coherence, which are essential for ensuring the consistency of shared memory in MP systems. A detailed investigation will be conducted on the MESI and MOESI protocols. The methodology adopted includes the creation of a multiprocessor system with three Processing Elements (PEs), each with its own private cache and sharing memory. For implementation, the C++ programming language will be used for the Backend, and Python with Tkinter for the User Interface. Additionally, random test code will be generated to perform comparative analysis between the protocols. The results obtained will encompass the simulation of the MESI and MOESI protocols in the multiprocessor system, where their performance will be thoroughly analyzed. These results will be visually presented in a graphical interface that will display cache states, along with additional metrics that will be used to evaluate the overall system performance.

**Palabras clave**—MESI, MOESI, memoria, coherencia

## I. INTRODUCCIÓN

En un mundo cada vez más orientado hacia la informática y la computación, es crítico que la optimización del rendimiento de los sistemas se lleve a cabo de una buena manera. Los procesadores modernos, o PEs (Processing Elements), desempeñan un papel fundamental en la ejecución eficiente de tareas computacionales, y una de las claves para mejorar su desempeño es la gestión eficiente de la memoria a través de caches.

En este contexto, el proyecto se enfoca en el estudio y análisis de la coherencia de cache (Cache Coherence Modelling) concepto que permite garantizar la consistencia de la memoria compartida en un sistema multiprocesador, que en el caso del proyecto sería el de la Figura 1. A medida que se escala el sistema añadiendo más elementos de procesamiento (PEs) la coherencia de caché se vuelve más compleja de mantener, ya que más unidades pueden acceder y modificar la misma área de memoria compartida. Es por esto que la implementación de los protocolos adecuados de coherencia de caché es sumamente importante en el desarrollo de un sistema eficiente, que evite inconsistencia de datos y garantice un comportamiento predecible del sistema.

Este proyecto tiene como objetivo principal modelar y evaluar el rendimiento de dos protocolos de coherencia, específicamente el protocolo MESI (Modified, Exclusive, Shared, Invalid) y el MOESI (Modified, Owned, Exclusive, Shared, Invalid) [1]. Para ello, se llevará a cabo la implementación de un sistema con tres PEs, cada uno de los cuales tendrá su propia caché privada y tendrán una memoria compartida, esto se puede observar en la Figura 1. Además, se implementará una interfaz gráfica que permita mostrar al usuario el proceso que realiza cada uno de los protocolos.

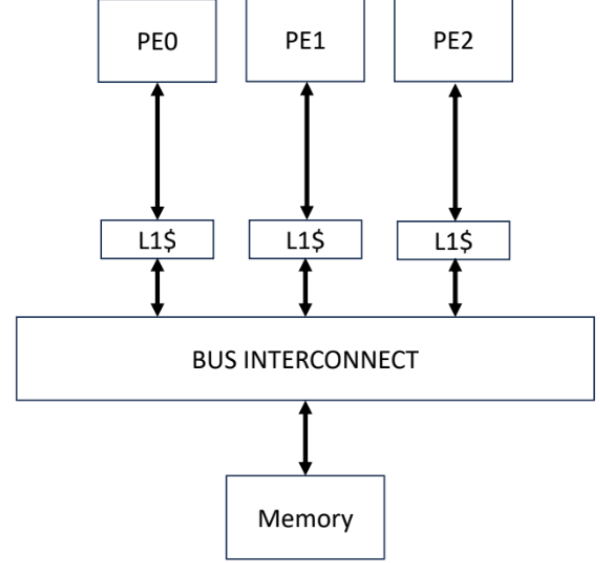


Fig. 1. Sistema multiprocesador

Por otra parte, también se incorporará un generador de código aleatorio para realizar pruebas, lo que añadirá dinamismo y permitirá realizar un mejor análisis comparativo de los dos protocolos.

## II. PROPUESTA DE DISEÑO

Para el diseño de nuestro sistema de modelado y evaluación de protocolos de coherencia de caché en sistemas multiprocesador (MP), se propone una arquitectura de dos capas compuesta por el Backend desarrollada en C++ y el Frontend que se desarrollará con Python y Tkinter.

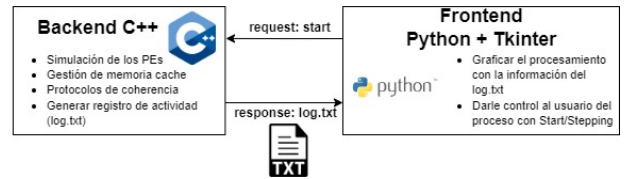


Fig. 2. Arquitectura propuesta.

Es importante mencionar que la comunicación entre ambas capas se hará mediante un archivo de texto o log, en el cual se le dará la información necesaria al Frontend para que se represente de manera correcta la ejecución realizada por el Backend. Por lo tanto, se puede decir que la comunicación es bidireccional, ya que el Backend le provee información al Frontend y este le envía una señal para iniciar la ejecución del programa.

### III. EVALUACIÓN DE LA PROPUESTA

A continuación se detallan las ventajas y desventajas tanto del Backend como del Frontend de nuestro proyecto. Esto permite evaluar de manera crítica los aspectos positivos y negativos de cada componente clave de nuestra aplicación, lo que a su vez nos ayudará en la toma de decisiones informadas para optimizar el desarrollo y el rendimiento del sistema en su conjunto.

En la siguiente sección se provee las ventajas y desventajas de las dos grandes capas por las que se compone el sistema.

#### A. Interfaz de Usuario (Python + Tkinter)

##### Ventajas:

- Tkinter ofrece una forma sencilla y rápida de crear interfaces gráficas completas.
- La combinación de Python y Tkinter facilita el desarrollo ágil de la interfaz de usuario.

##### Desventajas:

- Aunque Tkinter es amigable para los desarrolladores, puede tener restricciones en la personalización y diseño avanzado de la interfaz, lo que podría limitar la creatividad en la presentación visual.

#### B. Backend (C++)

##### Ventajas:

- La elección de C++ proporciona un rendimiento sólido, permitiendo una ejecución eficiente de la simulación.
- Brinda un control preciso sobre la gestión de la memoria y la optimización del rendimiento.
- La amplia disponibilidad de bibliotecas y recursos de desarrollo en C++ es una ventaja adicional.
- La capacidad de manejar eficazmente Threads es esencial para la gestión paralela de los PEs, lo que añade valor a la capa del Backend.

##### Desventajas:

- La implementación en C++ puede ser más compleja y detallada en comparación con algunos otros lenguajes de programación, lo que podría prolongar el desarrollo y requerir una curva de aprendizaje más empinada.
- La gestión manual de la memoria en C++ introduce el riesgo de errores de programación que deben ser cuidadosamente abordados.

Como se puede observar la Interfaz de Usuario (Python + Tkinter) se destaca por su capacidad para proporcionar una experiencia de usuario atractiva y accesible. La elección de Tkinter simplifica la creación de interfaces gráficas completas y la combinación con el lenguaje Python permite un desarrollo rápido y ágil.

Por otro lado, el Backend (C++) ofrece un rendimiento sólido y eficiente en la simulación y evaluación de protocolos de coherencia de caché. Además, proporciona un control preciso sobre la gestión de la memoria y la optimización del rendimiento, lo que es crucial en un proyecto de este tipo. Además, la amplia disponibilidad de bibliotecas y recursos de desarrollo en C++ brinda flexibilidad y apoyo. Por otro lado, la capacidad de manejar eficazmente Threads añade valor significativo a la capa del Backend.

### IV. ESPECIFICACIÓN DE COMPONENTES DEL SISTEMA

#### A. Diseño del Backend

Para la implementación de backend se dividió el problema en 8 grandes tareas de forma que al ejecutar cada una después de la anterior se pueda resolver el problema, como lo muestra el diagrama de la figura 4.

- Generación de código aleatorio: se genera tomando como semilla la hora exacta de iniciación del programa, también se corrobora que en todo código generado se tengan al menos una vez cada una de las posibles instrucciones esto para poder ejemplificar mejor el comportamiento del programa y ver todos los comportamientos en cada ejecución.
- Inicialización de las memorias, se define una memoria de instrucciones para cada uno de los PE donde se almacena el código a ejecutar, la memoria principal es un objeto singleton compartido por todo el sistema, esto para evitar copias que bajen la eficiencia del programa.
- Inicialización de los PE, una vez que se tiene el código a ejecutar en las correspondientes memorias, se inicializan los PE asociándolos a un manager singleton para mantener la identidad de cada PE y evitar que se generen más de una copia de cada uno, esto tanto para facilitar la utilización, como para evitar utilizar más memoria de la necesaria debido a pases de parámetro por copia.
- El bus es el objeto mediante el cual podemos enviar y recibir las solicitudes generadas por cada uno de los PE en ejecución, este bus es también el encargado de llevar cada uno de estas solicitudes, que pueden ser lectura, escritura o incremento, a cada uno de los protocolos para ser ejecutados.
- Una vez que los protocolos reciben cada una de las solicitudes, es aquí donde se ejecutan los cambios en memoria y se mandan las solicitudes de actualización en los estados de cada una de las entradas de memoria cache de los PE.
- Una vez finalizada la ejecución del programa se escribe un archivo log de cada una de las solicitudes, escrituras, lectura e incrementos que se ejecutaron, tanto a memoria principal como a memorias cache.

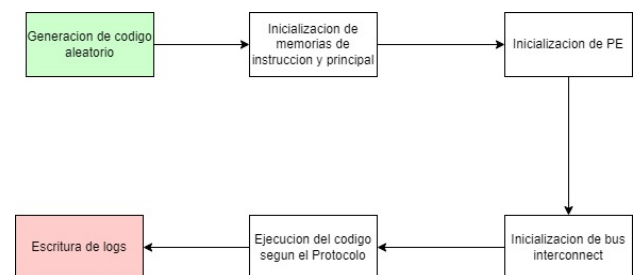


Fig. 3. Diseño del backend.

#### B. Diseño del Frontend

Se toma como referencia la recomendación dada por el profesor para facilitar la visualización y seguimiento de

la ejecución del programa de forma intuitiva, se hace en Python/Tkinter ya que la curva de aprendizaje del equipo es menor en este lenguaje y porque tiene funcionales que facilitan el manejo de objetos graficos. Se puede ver el diseño del frontend en la figura 3

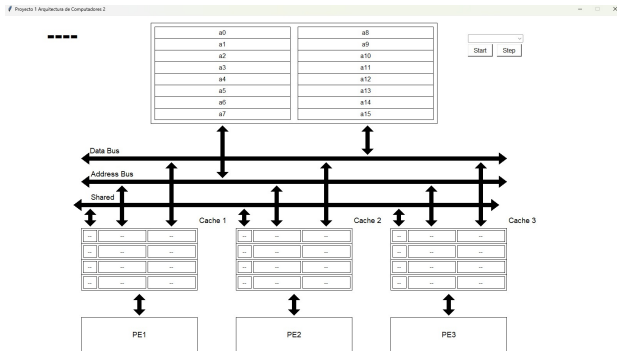


Fig. 4. Diseño del frontend.

## V. EXPLICACIÓN DEL PROCESO DE DISEÑO

- 1) **Investigación Inicial:** El proceso de diseño se inició con una fase de investigación detallada sobre cómo implementar y gestionar hilos en varios lenguajes de programación. Se investigaron lenguajes como C/C++, Cython, Java y Python para determinar cuál sería el más adecuado para la gestión paralela de los Procesadores Elementales (PEs) en el sistema multiprocesador.
- 2) **Demostración de Threads:** Como parte de la investigación inicial, se llevó a cabo la creación de un demo de manejo de hilos en cada uno de los lenguajes considerados. Esto permitió evaluar la eficiencia y la capacidad de gestión de hilos en cada lenguaje.
- 3) **Selección del Lenguaje:** Basándonos en los resultados de las demostraciones y las necesidades del proyecto, se tomó la decisión de utilizar el lenguaje de programación C++, el cual se utilizará para la implementación del backend del sistema. Esto debido a que el lenguaje proporciona un buen equilibrio entre rendimiento y capacidad de gestión de hilos, lo cual es fundamental para la implementación del sistema multiprocesador.
- 4) **Diseño de la Arquitectura General:** Una vez seleccionado el lenguaje principal, se procedió a diseñar la arquitectura general del sistema multiprocesador. Esto incluyó la definición de los componentes clave, como el Backend y la Interfaz de Usuario, y cómo interactuarían entre sí.
- 5) **Diseño Detallado de Componentes:** Se desarrollaron diseños detallados para cada componente del sistema. En el caso del Backend, se definió la manera en que se llevaría a cabo la simulación de los PEs y la gestión de la memoria caché. Para la Interfaz de Usuario, se diseñó la pantalla principal que mostrará el procesamiento dado por el backend.
- 6) **Implementación de Componentes:** Con los diseños detallados en su lugar, se procedió a la implementación

de los componentes del sistema. Se desarrolló el Backend para gestionar la simulación y los protocolos de coherencia de caché. La Interfaz de Usuario se creó para proporcionar a los usuarios la capacidad de interactuar con el sistema y observar los resultados del procesamiento.

## VI. RESULTADOS

En la Figura ?? se puede observar el archivo de actividad generado por el backend, el cual será utilizado por el Frontend para realizar la graficación.

### Log File output

```

r | 04 | 0      PE2 | 04 | E      PE3 | 04 | M
PE1 | 04 | M    w | 04 | 74      w | 04 | 78
r | 04 | 0      PE3 | 04 | I      PE3 | 04 | S
PE3 | 04 | M    PE2 | 04 | S      r | 04 | 78
PE3 | 04 | M    r | 04 | 74      PE2 | 04 | E
r | 04 | 0      PE3 | 04 | S      w | 04 | 95
PE3 | 04 | M    PE3 | 04 | E      PE3 | 04 | I
w | 04 | 70      w | 04 | 80      PE2 | 04 | S
PE3 | 04 | S    PE2 | 04 | I      r | 04 | 95
r | 04 | 70      r | 04 | 80      PE1 | 04 | E
PE2 | 04 | E    r | 04 | 80      w | 04 | 14
w | 04 | 74

```

Fig. 5. Archivo de log del backend

Por otro lado, en la Figura ?? y ?? se muestran los resultados obtenidos al correr los tests.

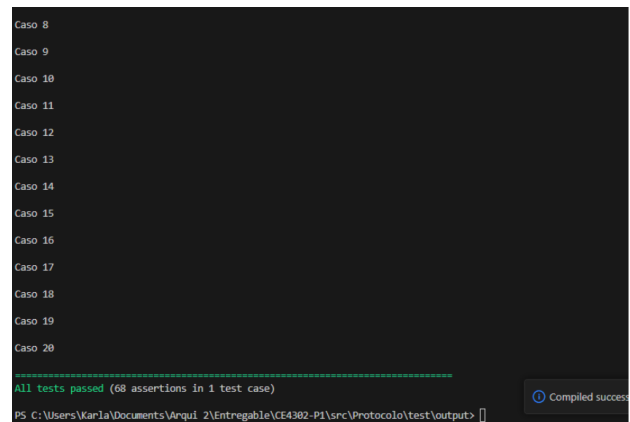


Fig. 6. Resultado de los tests de MESI

Se logró demostrar el funcionamiento de los protocolos de coherencia de caché, sin embargo, no logró realizar una instancia unificada donde se instancien ambos protocolos y estos reciban y ejecuten las instrucciones mediante threads.

## VII. CONCLUSIONES

Los protocolos de coherencia permiten mantener la validez de los datos en cachés privadas de los PE que tienen comunicación entre sí y la memoria principal.

Estos protocolos pueden ayudar a reducir la cantidad de conflictos para lectura y escritura de memoria y caché ya que ayudan a evitar estados de incoherencia de caché, esto mejora el rendimiento del sistema y mantiene los datos consistentes y actualizados.

```
Caso 12
Caso 13
Caso 14
Caso 15
Caso 16
Caso 17
Caso 18
Caso 19
Caso 20
Caso 21
Caso 22
Caso 23
Caso 24

=====
All tests passed (83 assertions in 1 test case)
PS C:\Users\Karla\Documents\Urqui 2\Entregable\CE4302-P1\src\Protocolo\test\output> []
```

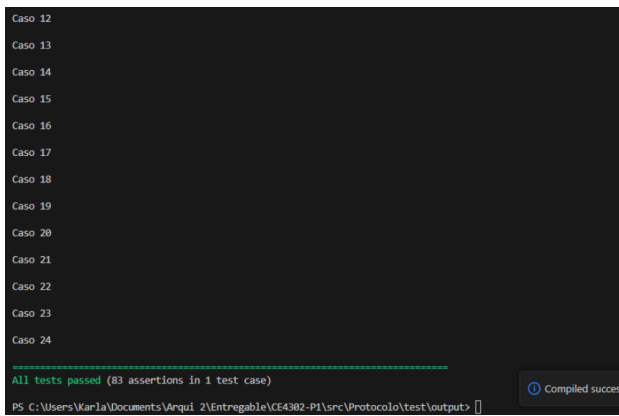


Fig. 7. Resultado de los tests de MOESI

Permite una evaluación constante de la caché, pero aumenta la complejidad de implementación del sistema.

#### REFERENCES

- [1] Onur Mutlu. Computer architecture: Cache coherence. <https://course.ece.cmu.edu/~ece740/f13/lib/exe/fetch.php?media=onur-740-fall13-module2.4-cache-coherence.pdf>. Carnegie Mellon University.