CE4202 - Proyecto 1

Modelo de protocolo para coherencia de caché en sistemas multiprocesador

Sebastián Calderón Yock

II Semestre 2022

Índice

Requerimientos del sistema

Especificación

El proyecto a desarrollar consiste en diseñar una aplicación de software con interfaz gráfica que modele un sistema multiprocesador con las siguientes características:

- ☐ Cuatro procesadores
- ☐ Cada procesador cuenta con una memoria caché local L1 con 4 bloques
- ☐ Los procesadores se encuentran conectados a memoria por medio de un bus
- ☐ La memoria caché L1 es mapeada de forma asociativa por set *one-way*
- ☐ Cada procesador deberá generar solicitudes de procesamiento o acceso a memoria (lectura o escritura) a diferentes regiones de memoria de forma aleatoria.

La aplicación deberá modelar un sistema similar al que se muestra en la Figura 1.

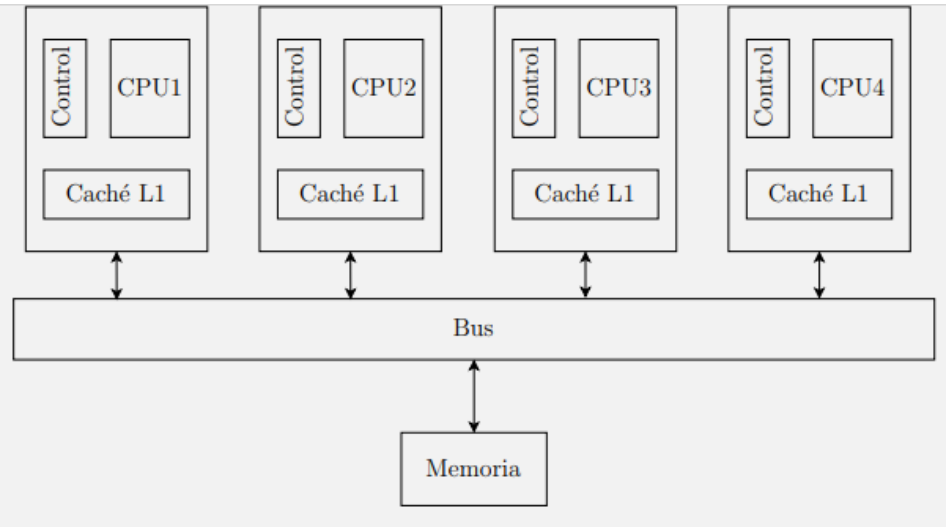


Figura 1: Arquitectura general del sistema

Sobre el modelo del sistema multiprocesador:

1. Deberá crearse una instancia del procesador de manera independiente para operar de forma paralela (mediante hilos).
2. Cada procesador deberá generar individualmente y de manera simultánea junto con los demás procesadores:

    a. Instrucciones de procesamiento (no requiere memoria). Estas instrucciones se llaman calc.
    b. Escritura o lectura a uno de los 8 bloques totales de la memoria compartida. Estas instrucciones se llaman read o write.
3. Cada una de las instrucciones debe generarse utilizando una distribución de probabilidad formal construida por el estudiante (binaria, de Poisson, hipergeométrica, etc). No se permite el uso de bibliotecas.
4. Una instrucción generada debe tener los siguientes componentes:
    a. Número de procesador
    b. Operación: read, write o calc
    c. En el caso de write se debe indicar la dirección de memoria (binario) y el dato a escribir (hexadecimal), respectivamente.
5. Una vez obtenida la instrucción, la dirección del bloque de memoria también será asignada de manera aleatoria con una distribución previamente documentada y justificada.
6. Una instrucción tiene el siguiente formato de ejemplo:

```
P0: READ 0100
P1: CALC
P3: WRITE 1010;4A3B
P4: CALC
```

El modelo del sistema deberá proveer una base de temporización para la generación de los eventos de procesamiento, lecturas y escrituras a definir por cada estudiante. Dicha base debe permitir la correcta visualización de todas las acciones generadas por el sistemas multiprocesador y sistema de coherencia de caché. En todo momento deberá mostrarse la acción de cada núcleo, así como el bloque de memoria que está utilizando. Cada procesador será identificado por su número único.

Sobre el modelo de memoria principal

La memoria principal será unificada y compartida, comunicándose con los procesadores mediante un único bus. Las funciones del modelo memoria principal serán:

- Actualizar el contenido compartido de los bloques en escrituras (según política de writeback).
- Permitir la visualización de los 8 bloques en todo momento.

El sistema deberá simular las condiciones propias de pared de memoria. Tendrá un retardo propio de lectura o escritura a memoria, cuando corresponda. El tiempo debe ser proporcional a la frecuencia elegida anteriormente, de manera que exista el problema de que el procesador es más rápido que la memoria.

Al iniciar la aplicación, el contenido de los bloques de memoria deben ser 0. Cada bloque es de 16 bits y debe ser representado en hexadecimal.

Sobre el modelo de memoria caché y sistema de coherencia

Cada procesador tiene su caché L1, se pueden almacenar 4 bloques en cada una y posee correspondencia asociativa por set one-way. Esta memoria servirá para almacenar cuatro bloques que requiera el procesador. La información que debe contener es:

- Número de bloque
- Estado de coherencia: M (modificado), S (shared), I (inválido), E (exclusive).
- Dirección de memoria
- Dato de 16 bits en hexadecimal

Se asumirá que si el dato es referenciado por primera vez, este no se encuentra en caché. La caché deberá generar alertas de misses tanto por escritura como por lectura. Además deberá permitir la visualización del contenido de los 4 bloques, por cada procesador, en todo momento.

Adicional al modelo de la caché, deberá diseñarse un modelo del sistema de coherencia, dentro del controlador, que deberá asegurar la coherencia entre todas las cachés. El modelo deberá incorporar una política de invalidación para las cachés de los demás procesadores. Ante una actualización en alguna caché, o un miss por invalidación, el sistema de coherencia deberá controlar la lectura del bloque correspondiente desde memoria principal, y la escritura hacia las demás cachés.

Para este diseño, se debe basar en el protocolo de monitoreo MESI. Debe quedar claro el tipo de protocolo utilizado y su descripción detallada. El controlador, además, deberá asegurar el correcto funcionamiento de la correspondencia de la caché.

Al iniciar la aplicación la caché estará fría, es decir, el contenido de los bloques de caché debe ser 0. Deberá realizar estos desaciertos obligatorios.

Sobre la interfaz gráfica

El sistema debe visualizarse por medio de una interfaz gráfica con el fin de observar el comportamiento de cada uno de los procesadores al generar las instrucciones. La temporización elegida debe permitir la correcta visualización de todas las acciones generadas por el sistema multiprocesador y sistema de coherencia de caché. Los elementos que deben visualizarse son:

- ☐ Identificador de cada procesador.
- ☐ Información de la caché L1 detallada en la sección anterior para cada procesador.
- ☐ Última instrucción generada, así como la instrucción generada para cada

procesador.

- [ ] Contenido de la memoria.
- [ ] Alertas de miss tanto de lectura como de escritura

El sistema a nivel temporal tendrá dos modos:

1. Paso a paso (solo ejecuta el siguiente ciclo).
2. Ejecución continua (no se detiene hasta que el usuario lo detenga con pausa).

Se usan pausas para poder distinguir de mejor manera la transición de los estados de los diferentes bloques de caché. Además, el usuario puede agregar una instrucción a ejecutarse en el siguiente ciclo. Para ello, cuando el sistema esté en pausa, se podrá elegir un procesador e indicar el tipo de instrucción, así como la lectura o escritura con sus respectivas direcciones y dato en caso de que aplique. Sólo se podrá agregar una instrucción en modo pausa y se verá reflejada luego de ejecutar la instrucción actual.

Requerimientos

De la especificación anterior es posible extraer una lista de requerimientos que se analizará por etapa.

Sistema multiprocesador

- [ ] Cuatro instancias de procesadores independientes que operen de forma paralela por medio de hilos.
- [ ] Algoritmo de generación de números aleatorios por medio de una distribución de probabilidad.
- [ ] Generación de instrucciones aleatorias (**calc**, **read** o **write**) por medio de una distribución de probabilidad. Las instrucciones deben contener: Número de procesador, tipo de operación y [dirección (aleatoria), dato] en el caso de write
- [ ] Generación de direcciones de memoria aleatorias por medio de distribución de probabilidad.
- [ ] Base de temporización que permita visualización.

Memoria principal

- [ ] Memoria unificada y compartida de 8 bloques con datos de 16 bits, accesible por medio de un bus.
- [ ] Lectura y escritura de cualquiera de los 8 bloques de memoria.
- [ ] Temporización que simule las condiciones de pared de memoria, proporcional al definido para el sistema en general.
- [ ] Inicialización de los bloques en 0

- [ ] Visualización de los datos de cada bloque en formato hexadecimal.

Memoria caché

- [ ] Caché L1 propia para cada procesador, con 4 bloques de memoria con correspondencia asociativa por set *one-way*.
- [ ] Bloques deben almacenar la siguiente información: Número de bloque, estado de coherencia (M, E, S, I), dirección de memoria y dato de 16 bits.
- [ ] Generación de alertas de misses para lecturas y escrituras

Controlador de caché

- [ ] Controlador de caché para cada procesador
- [ ] Modelo de coherencia de caché tipo MESI
- [ ] Política de invalidación para las cachés de los demás procesadores
- [ ] Protocolo de monitoreo MESI
- [ ] Asegurar la correspondencia de caché

Interfaz gráfica

- [ ] Interfaz que permita interactuar con la simulación y muestre los siguientes elementos:
  - [ ] Identificador de cada procesador
  - [ ] Información de la caché L1
  - [ ] Última instrucción generada y las instrucciones generadas para cada procesador
  - [ ] Contenido de la memoria
  - [ ] Alerta de miss para lectura y escritura

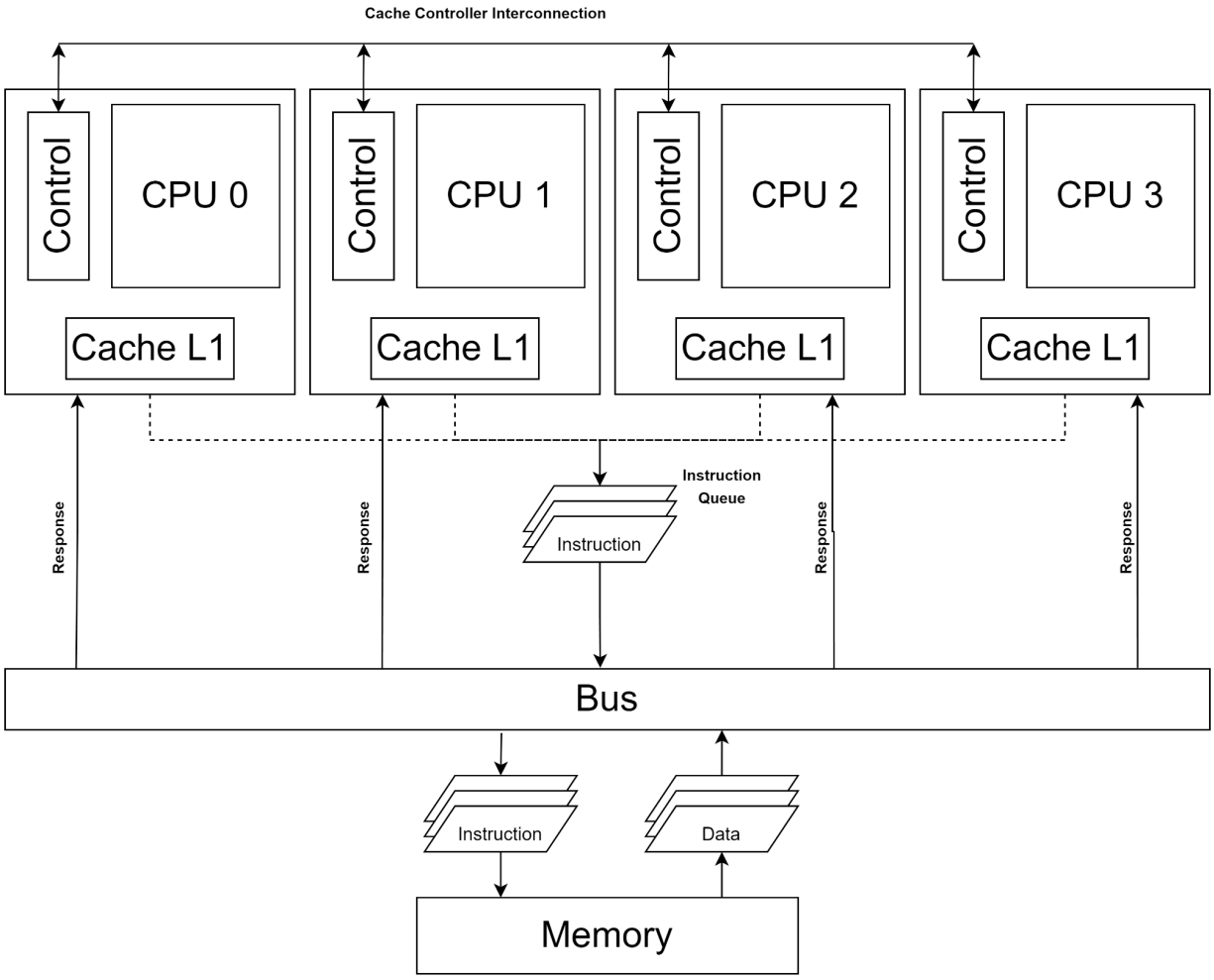
Opciones de solución

Diseños de Arquitectura del sistema

Se consideraron dos diseños para la implementación del sistema multiprocesador. En ambos diseños el procesador genera instrucciones aleatorias (calc, read o write). En caso de un read o un write, el controlador de caché evalúa si se cuenta con el dato, si debe realizar un broadcast a los demás controladores para obtener/invalidar el dato o si debe emitir una petición hacia memoria.

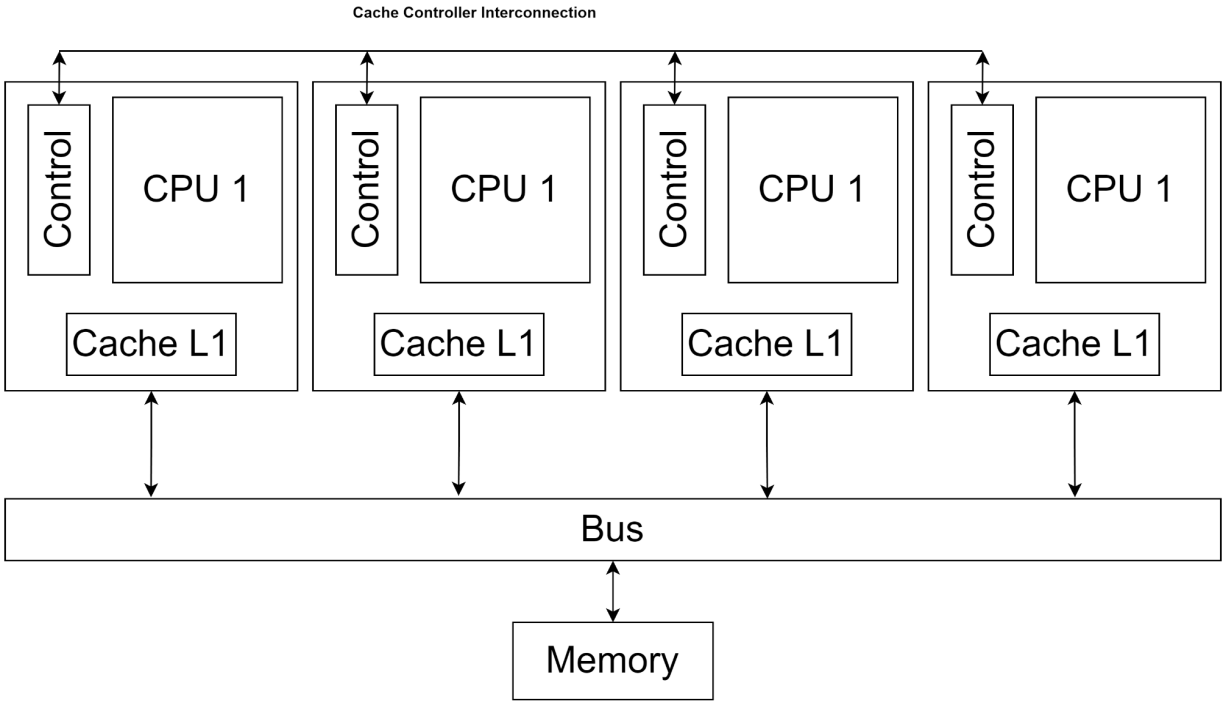
Uno de los diseños cuenta con colas de transmisión entre los procesadores y el bus y entre el bus y memoria. El otro diseño cuenta con un único canal de transmisión por medio del bus. Ambos diseños cuentan con una interconexión entre los controladores para agilizar la comunicación entre estos. Ambos diseños utilizan un mutex para restringir el acceso de los procesadores, ejecutándose en un hilo simultáneo, al bus (el cual es la región crítica en este sistema).

- Opción 1: modelo con colas de transmisión de datos e interconexión de controladores



En esta propuesta, existe una cola de instrucciones entre los procesadores y el bus, que permite a los procesadores emitir instrucciones sin bloquear a los demás procesadores de interactuar con el bus. Del mismo modo, existe una cola de instrucciones y una de respuestas entre el bus y la memoria. Esto permite a la memoria procesar múltiples peticiones y enviar los datos solicitados de forma paralela.

- Opción 2: modelo simplificado de comunicación secuencial con interconexión de controladores



En esta propuesta, existe un único canal de comunicación utilizable simultáneamente entre todos los dispositivos. Si se emite una petición a memoria por un procesador, hasta no haber obtenido la respuesta, los demás procesadores no pueden interactuar con el bus.

Lenguaje de programación/herramientas de desarrollo

Para este proyecto, dado que se busca simular hardware y su interacción, se consideraron las siguientes opciones de lenguajes/herramientas para la implementación:

- Opción 1: HDL

Una de las opciones consistía en HDL, pues cuenta con templates de dispositivos que permitirían simular más correctamente el comportamiento de hardware y su interacción. Del mismo modo, la lógica que debía implementarse tenía un enfoque más fidedigno al que habría con hardware real (máscaras de bits, emisión de descriptores con un formato estricto, etc)

- Opción 2: Simics

Otra de las opciones consideradas fue un simulador de sistema completo como Simics, el cual, similar a HDL, cuenta con templates de dispositivos, interfaces de conexión, definición de dispositivos predeterminados (como memoria), etc. El lenguaje de modelado DML es muy similar a C y soporta Python.

- Opción 3: C/C++

Se consideró C/C++ como una opción a un nivel más alto que las anteriores, porque permite el manejo de memoria. Esto permitía mantener cierta fidelidad del manejo de bits y transmisión de datos, respecto a un lenguaje de más alto nivel.

- Opción 4: lenguaje de alto nivel (Python)

Finalmente, se consideró un lenguaje de alto nivel, en específico Python, debido a su versatilidad, soporte y amplias bibliotecas.

Lenguaje de programación/herramientas de desarrollo para interfaz

- Opción 1: Qt

Se consideró Qt pues permite crear UI rápida y fácilmente, por medio de la construcción de Drag and Drop y asociar componentes con funciones. También por su fácil conexión con otros lenguajes (C, Python, etc)

- Opción 2: Pygame

Pygame fue considerado por ser propio de Python, lo cual podría reducir el ruido o posibles fuentes de error durante el manejo de hilos.

Comparación de opciones de solución

Diseños de Arquitectura del sistema

Sobre las opciones presentadas, la mayor diferencia entre una y otra es que el modelo simplificado no permite la interacción de otros procesadores con el bus de datos si se está realizando un fetch de memoria. Esto hace que el diseño con colas de transmisión sea más deseable por su mayor paralelización y porque tiene un mayor throughput hipotético. Es importante considerar que para este diseño se debe realizar una sincronización más compleja entre los dispositivos y podría introducir problemas imprevistos en el manejo de hilos.

Por el lado del diseño simplificado, aunque no es tan eficiente, es una abstracción pertinente que minimiza la cantidad de trabajo y cumple con todos los criterios a evaluar de la especificación, además de que esta diferencia de diseño no choca de ninguna forma con los conceptos que se buscan evaluar en este proyecto.

Lenguaje de programación/herramientas de desarrollo

Los lenguajes de descripción de hardware y el simulador de Simics son propuestas muy buenas, pues acelerarían todo el proceso de modelado de dispositivos como buses, memoria, caché, etc. Aunque igualmente se tendría que modelar un procesador que opere como la abstracción solicitada en la especificación del proyecto. Aunque son buenas propuestas, conllevan inconvenientes como la comunicación con la eventual interfaz, soporte de bibliotecas (random, testing, etc) y las complejidades propias del hardware que no deben contemplarse normalmente en software.
Por otro lado, C/C++ son convenientes por el manejo de memoria, pero tiene implicaciones en el proceso de desarrollo: una de ellas es que sería indiscutiblemente más lento que realizar el proyecto en un lenguaje de más alto nivel, así como el uso de debuggers puede dificultarse, también el log de errores, etc.
Respecto al lenguaje de alto nivel, el desarrollo sería más ágil en general y puede comunicarse sin problemas con cualquier implementación de UI sin necesidad de un traductor o una herramienta semejante. La dificultad que introduce es que todos los dispositivos y todo su comportamiento se debe modelar desde cero.

Lenguaje de programación/herramientas de desarrollo para interfaz

Los lenguajes/herramientas para la interfaz realmente no tienen mayor diferencia entre ellas para efectos de este proyecto. Ambas opciones proveen con facilidad los componentes necesarios para realizar una interfaz para el sistema. El criterio de selección se reduce a cuál de los dos permitiría implementar la interfaz más rápida y sencillamente.

Selección de la propuesta final

Diseños de Arquitectura del sistema

Debido a lo descrito en la sección de comparación entre los diseños de arquitectura, se selecciona la **segunda opción** ya que es una arquitectura mucho más simple, que reduciría eventuales problemas en manejo de hilos y sincronización y que cumple con los requisitos definidos para este proyecto.

Lenguaje de programación/herramientas de desarrollo

Se utilizará la opción de **alto nivel, Python,** debido a su facilidad de integración con la interfaz, la versatilidad en el desarrollo y el soporte de bibliotecas.

Lenguaje de programación/herramientas de desarrollo para interfaz

Se utilizará **Qt,** debido a su capacidad de implementar una interfaz con componentes simples rápidamente, sin necesidad de crear funciones para el control general de la aplicación.