Proyecto II: Documento de Diseño

Yordi Brenes Roda
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
ybrenesr@estudiantec.cr

Luis Andrey Zúñiga Hernández
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
lazh@estudiantec.cr

Fátima Alicia Leiva Chinchilla
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
2019153089@estudiantec.cr

Brian Wagemans Alvarado
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
briwag88@estudiantec.cr

I. Introducción

El presente informe corresponde al proyecto 1 del curso Arquitectura de computadores CE-4302 el cual consistió en la modelación y evaluación de dos protocolos de coherencia de caché. Los protocolos evaluados son el MESI y MOESI.

La microarquitectura modelada consiste en 3 Procesing Elements (PEs), cada uno con caché privada y una única memoria compartida (figura 1). Las caché tienen una política de write-back. Las dimensiones de las cachés y memoria compartida se muestran en la tabla I. Cada PE se ejecutará de forma paralela, esto será modelado utilizando threads.

TABLA I: Instrucciones de control de flujo de la arquitectura

Tipo de memoria	**Número de entries**	**tamaño de dato**
Memoria compartida	16	32 bits
caché privada	4	32 bits

El código que ejecutará cada procesador será generado aleatoriamente. Las instrucciones posibles se muestran en la tabla II.

TABLA II: Instrucciones de control de flujo de la arquitectura

Instrucción	**Sintaxis**	**Operación**
write	write addr reg	escribe el valor de reg en la dirección
read	read addr reg	carga el valor de la dirección en reg
incr	incr reg	incrementa el valor de reg en 1

El simulador permitirá avanzar paso a paso la ejecución del código y los cambios de estados en la chaché. Para esto, y otras funcionalidades, como ver los estados de las lineas de caché y cambiar entre protocolos, se usará una interfaz gráfica (UI). Una interfaz que se usará como guía se puede ver en la figura 2.Finalmente, la simulación presentará un resumen del total de transacciones y su tipo.

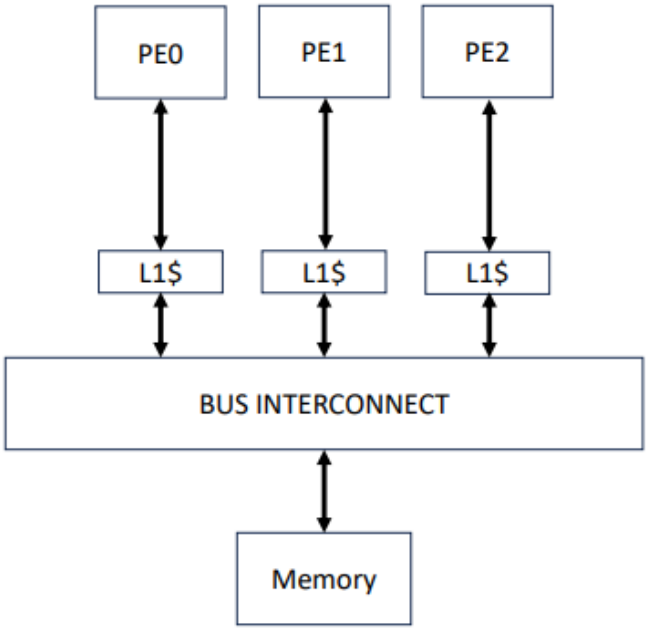

Fig. 1: Diagrama de la microarquitectura del sistema a modelar

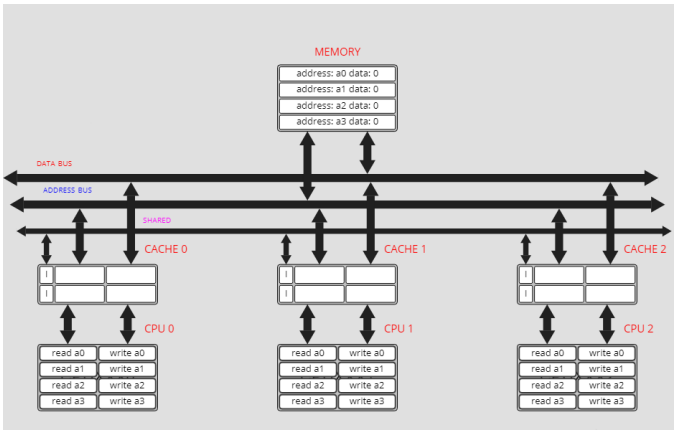

Fig. 2: Ejemplo de UI

II. Consideraciones

La proyecto debe ser basada en una arquitectura de software cliente-servidor.

El simulador debe permitir comparar el desempeño de un protocolo MESI y MOESI, por lo que el diseño y elección de los parámetros por medir, son esenciales. Además, la forma en que se genere el código va a impactar la frecuencia de transacciones de ambos protocolos.

El sistema no solo debe permitir la transferencia de datos de la memoria principal a caché, sino que tambíen de una caché a otra.

A continuación se determinará

1) El tipo de aplicación que se desarrollara
2) Herramientas de desarrollo
3) Protocolo de comunicación

III. Propuestas planteadas

A. Tipo de aplicación

Se busca una arquitectura de software cliente-servidor que modele y evalúe cada protocolo simulado. Se debe determinar si la aplicación desarrollada será de escritorio o web, para esto se evaluara los parámetros de rendimiento, versatilidad y escalabilidad

1) Rendimiento: Una aplicación de escritorio será más rápida y receptiva que una aplicación web, ya que la app de escritorio aprovecha mejor los recursos locales. Mientras que la comunicación en la aplicación web conlleva latencia.

2) Versatilidad: El principal atractivo de una aplicación de escritorio es que no se necesita una conexión a internet, cosa que no sucede en una aplicación web. En este ámbito, la aplicación de escritorio es mejor. Sin embargo, una aplicación de escritorio estará vinculada a una plataforma específica, mientras que una aplicación web será accesible desde cualquier tipo de navegador con acceso a internet. Otra gran desventaja de la aplicación de escritorio, es que, en caso de una actualización de software, esta debe hacerse manualmente, mientras que en web no.

3) Escalabilidad: En este ámbito, la aplicación web resulta ganadora. Esta permite llegar a un mayor número de dispositivos diferentes y hacer cambios en el programa de forma más simple.

B. Medio de desarrollo

Para el desarrollo web se pueden utilizar diferentes lenguajes y frameworks para implementar el backend y frontend. Se bajaraon los siguientes.

1) node.js y React
2) dotnet y Angular
3) Django y Python

C. Comunicación Backend-Frontend

Se plantearon dos protocolos de comunicación: UDP y HTTP. A continuación se evalúan los parámetros de fiabilidad y rendimiento. La seguridad en la comunicación no se considera un factor esencial en el simulador, por lo que no se tomará en cuenta.

1) Fiabilidad: UDP no garantiza la entrega de datos ni el orden correcto [1].. Aspecto crítico para el simulador. Por otro lado, HTTP garantiza la entrega de datos sin errores y en el orden correcto [2].

2) Rendimiento: El protocolo UDP está orientado a la velocidad, pero se da por sentado que datos que se pueden perder [1]. Por otra parte, el protocolo HTTP presenta latencias ya que debe establecer y finalizar conexiones [2].

IV. Selección de propuestas

A. Tipo de aplicación

En cuanto al rendimiento, la aplicación de escritorio resultó mejor, por otro lado, en el área de versatilidad y escalabilidad la aplicación web supera a la de escritorio.

Aunque en un principio no se tiene planeado realizar actualizaciones o escalar el programa, se decidió que tener la opción disponible no es despreciable. Como la aplicación web dominó en dos de los tres parámetros tomados en cuenta, y además, porque el rendimiento del programa no se considera un factor limitante, se desarrollará una aplicación web.

B. Comunicación Backend-Frontend

El hecho de que el protocolo UDP no garantiza la entrega de datos hace que la propuesta quede descalificada. Los datos son críticos para el funcionamiento correcto. Además, el rendimiento no es una preocupación por la poca carga de mensajes.

C. Herramientas de desarrollo

En este caso, por la familiaridad de desarrollo con las tecnologías, se eligió, unánimemente, que el entorno dotnet y Angular serán la mejor opción. La razón de mayor peso siendo la familiaridad con los entornos. Aspecto importante tomando en cuenta el nivel de carga académica y lo imprescindible que es la velocidad.

D. Conclusión

Al analizar cada propuesta se llegó a la conclusión de que se desarrollará una aplicación web, utilizando dotnet para el backend y angular para el frontend. Además, para la comunicación se usará el protocolo HTTP.

V. proceso de desarrollo

A. Estructura del código

Hay dos clases particularmente sobresalientes en la estructura del código. Estas son la clase de memoria y la clase de bus. El bus es el que verifica la existencia de datos en otras cachés. Ambas clases utilizan el patrón singleton para evitar duplicaciones en las instancias. Esta medida se tomó porque el desarrollo con hilos puede resultar engorroso.

Se eligió utilizar 8 registros por PE. Cada PE es representado por un hilo en ejecución. El acceso a memoria solo se puede dar a través del bus, por lo que el único recurso al que tienen acceso los PEs es el bus. Para evitar problemas en la consistencia de los datos, este recurso solo puede ser utilizado por un hilo (PE) a la vez. Se implementó un semáforo para evitar condiciones de carrera.

B. Semáforo

El semáforo controla quien accede al recurso bus, este posee una variable entera que representa el número posible de accesos futuros. El semáforo realizará dos operaciones.

1) down(): Si la disponibilidad del recurso es mayor a 0, disminuye el valor de accesos disponibles. Si la disponibilidad es 0, el hilo (PE) espera y no se realiza disminución.
2) up(): Incrementa el valor del semáforo. Si existe un hilo esperando, este toma el bus y continua ejecutándose.

A continuación se muestra el pseudocódigo de como se utiliza el semáforo (figura 3).

```
Function ExecA():
    down(resourse1)
    useResourse1();
    up(resourse1);
```

Fig. 3: Pseudocódigo uso de semáforo.

Al realizar una solicitud del bus, el semáforo debe verificar el valor de la variable de recursos disponibles y luego escribir sobre la variable. Como el semáforo debe controlar el uso de recursos para los tres hilos, teóricamente puede darse que, mientras que el semáforo esté actualizando el valor de la variable de conteo de recursos, otro hilo solicite el bus, generando que la verificación de los recursos sea incorrecta por que la escritura sobre la variable no sea ha realizado. En otras palabras, utilizando el semáforo, se trasladó el conflicto del bus al semáforo.

Aunque con tres PE el la posibilidad de problemas por concurrencia en el semáforo deberá ser prácticamente nula, se utilizaron locks en el proceso de verificación y escritura del semáforo para crear una región crítica de forma preventiva. En la figura 4 se muestra el pseudocódigo. Con la estructura de la figura 4, se garantiza la atomicidad de la secuencia de operaciones.

```
Function down(resourse1):
    if (lockObject):
        if(resourses > 0):
            asignResourse(resourse1);
            resourser--;
```

Fig. 4: Concurrencia en el semáforo

C. Flujo del código

Cuando un PE solicita un elemento, este puede o no puede estar en caché, esto corresponde a un hit o miss, respectivamente. El flujo de un hit o miss en caché para un hilo cualquiera se muestra en la figura 5.

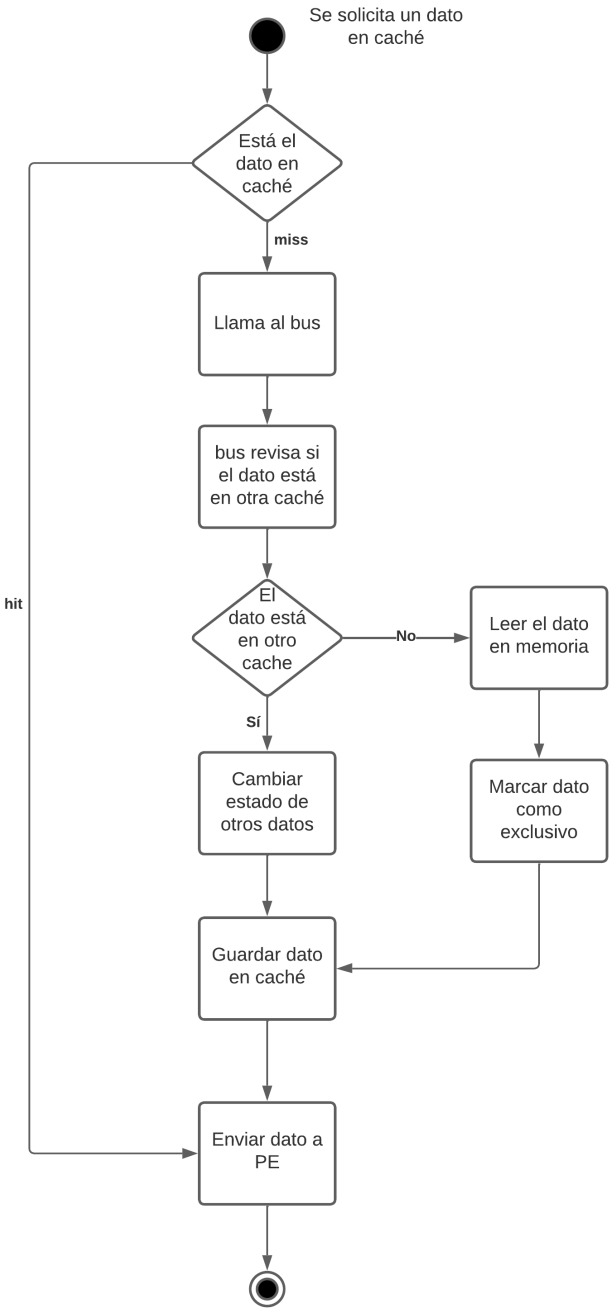

Fig. 5: flujo de acceso a caché.

VI. RECOMENDACIONES

La implementación del proyecto tiene las verificaciones de las transacciones de estados para MESI y MOESI muy poco cohesivos. Se recomienda, para aprovechar la naturaleza de máquina de estado de ambos protocolos, implementar el patrón State Pattern, el cual permite representar un objeto que puede tener varios estados y cambiar de un estado a otro si se cumplen cierta condición o eventos. Esto simplificaría la complejidad del código.

REFERENCIAS

[1] A. Zelinsky, "TCP VS UDP: The differences explained." [Online]. Available: https://medium.com/gitconnected/tcp-vs-udp-the-differences-explained-c2b9a88017d9

[2] "An overview of HTTP." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview