Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2343 – Arquitectura de Computadores

Ayudantía 7 – Solución propuesta

Profesor: Yadran Francisco Eterovic Solano

Ayudante: Germán Leandro Contreras Sagredo (glcontreras@uc.cl)

Nota al lector

El título dice "solución propuesta" por una razón bien sencilla: Estos ejercicios pueden tener más de un desarrollo correcto. Lo que se pretende hacer aquí es mostrar un camino a la solución, sin excluir la posibilidad de rutas alternativas igual de correctas.

Preguntas

1. a. Dentro del contexto de múltiples procesadores, existen dos formas de que estos tengan acceso a una misma fuente de memoria: `UMA` (*Uniform Memory Access*) y `NUMA` (`Non-Uniform Memory Access`). ¿En qué consiste cada una de estas? Mencione, además, una ventaja y desventaja para cada una.

   - **UMA:** Consiste en la arquitectura de memoria compartida en la que todos los procesadores acceden a la memoria de forma uniforme. Existen varios tipos, pero la estudiada en el curso es la `SMP` (*Symmetric Multiprocessor System*), donde todos los procesadores interactúan con la memoria a partir de un único bus de datos. Su ventaja es que el tiempo de acceso a una sección de memoria es independiente del procesador que trate de acceder a ella, haciéndolo útil en aplicaciones de propósito general y tiempo compartido para múltiples usuarios. Su desventaja, no obstante, es que es costoso de implementar (tamaño del bus de datos considerando la cantidad de procesadores), además de tener una escalabilidad restringida debido a la saturación del bus compartido en el caso de muchos procesadores.
   - **NUMA:** Consiste en la arquitectura de memoria compartida en la que esta se distribuye a partir de unidades de memoria locales a las que pueden acceder los distintos procesadores. Su ventaja radica en que es menos costoso de implementar que `UMA`, además de tener una escalabilidad mucho menos restringida al poder extenderse fácilmente (no dependen de un bus de acceso particular para todas las memorias locales disponibles). La desventaja es que los procesadores que busquen acceder a unidades de memoria local remotas tardarán más en poder concretar el acceso, debido a la red interconectada que se debe atravesar para la transferencia de datos.

b. ¿En qué consiste el mecanismo de consistencia de "*caché snooping*"?

Corresponde al mecanismo en el que cada *caché* posee un controlador que revisa las solicitudes de lectura y escritura realizadas en la memoria principal, revisando si la *caché* correspondiente contiene el dato accedido y modificado. Estas solicitudes se envían mediante *broadcast* a lo largo de todo el bus para que los controladores puedan ver las solicitudes y ejecutar acciones dependiendo de estas. La acción a realizar dependerá del protocolo implementado (por ejemplo, se pueden invalidar las líneas de la caché que hayan sido modificadas, o bien se pueden sobreescribir).

c. Existe otro mecanismo de consistencia de *caché* conocido como coherencia basada en directorio. En este, existen uno o varios directorios en la memoria que mantienen registro de los elementos compartidos por las *cachés* y sus estados. Tiene la ventaja de ser escalable, pero la desventaja de que puede producir *overhead* por el contenido que no se busca acceder dentro de este. En base a esto, ¿qué ventaja y desventaja posee el mecanismo de *snooping* si se compara con el basado en directorio?

- **Ventaja:** Es fácil de implementar en un sistema basado en buses de datos, debido a la implementación directa del *broadcast* de modificaciones realizadas sobre un dato en particular.
- **Desventaja:** Es limitado por el bus utilizado. Si existen muchos procesadores (y por ende, controladores de *caché*), entonces la gran cantidad de tráfico saturará el bus.

d. Explique el protocolo *write through* de consistencia de *cachés*. ¿Qué otras variaciones existen?

En este protocolo, siempre se actualiza la memoria, no solo las *cachés*. Supongamos que tenemos los procesadores `A` y `B`. A partir de estos tenemos los siguientes casos:

- ***Read Miss*:** El procesador `A` quiere leer un dato, pero no se encuentra en su *caché*. Accede a la memoria principal para leerlo y el controlador de su *caché* se encarga de escribirlo en una de sus líneas. Como se manda una solicitud mediante *broadcast*, el controlador de la *caché* del procesador `B` se entera, pero no hace nada dado que no se generan cambios.
- ***Read Hit*:** El procesador `A` quiere leer un dato y se encuentra en su *caché*. Accede a esta para leerlo. Como no se manda ninguna solicitud por la interacción directa con la *caché* local, el controlador de la *caché* del procesador `B` no se entera de lo sucedido.
- ***Write Miss*:** El procesador `A` quiere escribir sobre un dato, pero no se encuentra en su *caché*. Accede a la memoria principal para escribirlo, pero su controlador de *caché* no escribe la línea recientemente modificada. Como se envía mediante *broadcast*, el controlador de la *caché* del procesador `B` se enterará y verá si la palabra modificada está o no contenida. Si no la tiene, no realiza acción alguna. En otro caso, la línea será marcada como inválida para su posterior sobreescritura.
- ***Write Hit*:** El procesador `A` quiere escribir sobre un dato y se encuentra en su *caché*. Como además será escrita en memoria, se enviará la solicitud de escritura mediante *broadcast* y el controlador de la *caché* realizará el mismo procedimiento señalado en el punto anterior.

El protocolo antes mencionado se puede catalogar como "*write invalidate*" o con estrategia de invalidación (se invalidan las entradas de la caché que son obsoletas). Una variante es denominada "*write update*" o con estrategia de actualización, en la que en vez de marcar

la línea como inválida, se actualiza su contenido de forma inmediata con el nuevo valor de la memoria principal (lo que es equivalente a realizar un *read* de dicha palabra posterior a la invalidación de la entrada en el protocolo anterior).

e. Mencione y explique cada estado del protocolo *write-back* `MESI`.

El protocolo *write-back* `MESI` busca mejorar el desempeño de la consistencia de *caché* evitando actualizar en cada solicitud de escritura la memoria principal. Para ello, maneja un total de cuatro estados para cada línea de la caché:

- ***M**odified*: Indica que la línea de la *caché* corresponde a un bloque de la memoria principal que ha sido modificado (siendo el procesador el que lo modificó desde su *caché*).
- ***E**xclusive*: Indica que dicha línea de la *caché* corresponde a un bloque de la memoria principal que solo ha sido accedido por el procesador correspondiente, haciéndola **exclusiva**.
- ***S**hared*: Indica que dicha línea de la *caché* corresponde a un bloque de la memoria principal que ha sido accedido por más de un procesador, haciéndola **compartida**.
- ***I**nvalid*: Indica que dicha línea de la *caché* es **inválida** (es decir, está disponible para su uso pues lo contenido en ella no tiene validez alguna, ya sea por ser recientemente inicializada o por no estar actualizada).

Puntos importantes a considerar a partir de los estados anteriores:

- La primera vez que se accede a un bloque de la memoria principal, se lleva a la *caché* del procesador que solicita el acceso y se marca con estado `E`.
- Si el mismo procesador hace lecturas del mismo bloque, lo hace desde la línea de la *caché* y no actualiza su estado, no se hace uso del bus.
- Si otro procesador lleva dicho bloque a su *caché*, se da aviso de la existencia de una copia y ambos marcan la línea correspondiente con estado `S`.
- Si alguno de estos procesadores hace una lectura del mismo bloque, nuevamente se hace desde la línea de la *caché* y el estado no se modifica, no se hace uso del bus.
- Si un procesador quiere modificar un bloque de memoria que está contenido en una línea de su *caché* y tiene estado `E`, lo hace de forma directa en esta y pasa su estado a `M`, no hace uso del bus.
- Si un procesador quiere modificar un bloque de memoria que está contenido en una línea de su *caché* y tiene estado `S`, lo hace de forma directa en esta, pasa su estado a `M`, y hace uso del bus para enviar una señal de invalidación al resto de los procesadores. Los que contengan dicha línea, la pasarán a estado `I`.
- Si ahora otro procesador hace una lectura del bloque de memoria que fue actualizado y que se encuentra en una línea en estado `M` en el procesador que lo modificó, entonces este último hará una solicitud de espera, en la que el procesador que desea leer lo hace una vez que el bloque de memoria principal ha sido actualizado.

- Si este mismo procesador hubiera hecho una escritura sobre dicho bloque, entonces el procesador que posee la línea en estado `M` hará una solicitud de espera para modificar el bloque en la memoria principal y, posteriormente, marca su línea en estado `I` ya que ya no será consistente. El procesador que hace la escritura, por otra parte, no necesariamente guardará la línea en su *caché*. Al igual que en el protocolo *write through*, dependerá de la variante. En la versión estudiada en el curso no se hace, pero sí en caso de emplear una polícita *write-allocate*.

2. a. **(IIC2523 - I1 - II/2015)** Una arquitectura `UMA`, aún cuando es más sencilla, es siempre más eficiente que una arquitectura `NUMA`. ¿Es esta sentencia verdadera o falsa? Justifique.

   La sentencia es **falsa**. Es más eficiente solo en un contexto acotado de procesadores. Cuando se tiene una cantidad de procesadores mayor, el bus compartido de datos comienza a saturarse por todas las solicitudes de lectura y escritura mediante *broadcast*, lo que hace más eficiente a la arquitectura `NUMA` ya que muchas de estas se hacen directamente a las unidades locales de memoria, disminuyendo el uso de los buses compartidos.

   b. **(Examen - II/2018)** En un multiprocesador (varias CPUs que comparten una memoria común a través de un bus), cada CPU puede tener su propia memoria *caché* para así evitar tener que recurrir frecuentemente a la memoria principal a través del bus. Sin embargo, esto normalmente da origen al problema de coherencia de *cachés*. Para solucionar estos problemas, los controladores de *cachés* son diseñados de manera que "observen" (*snoop*) las solicitudes que pasan por el bus (y que fueron hechas por alguna otra *caché*) y que hagan algo en ciertos casos. El conjunto de reglas que define qué hacer y cuándo se llama protocolo de consistencia de *cachés*.

      I. Considera el protocolo *write through* estudiado en clase, cuya esencia es que todas las operaciones de escritura resultan en que la palabra que está siendo escrita en la *caché* también es escrita en la memoria para mantener la memoria actualizada todo el tiempo. Si el controlador de caché solo pudiera observar las líneas de dirección del bus, y no las de datos, ¿se vería el protocolo *write through* afectado por esta situación? Explica.

      En el protocolo básico **no** es un problema. El controlador de la *caché* ve la dirección que fue modificada y simplemente ve si esta se encuentra en la suya. Si no está, no hace nada. En otro caso, marca la entrada como inválida y finaliza el funcionamiento. El problema está en la **variante** (con estrategia de actualización), debido a que si da cuenta de que la línea quedará inválida, no podrá copiar el valor directamente en la *caché* dado que no lo puede ver en el bus. Sería necesario modificar el protocolo para que se haga de manera consecuente un *read* desde los procesadores con inconsistencia.

      II. En los protocolos de tipo *write-back* no todas las escrituras van directamente a la memoria: cuando una línea de la *caché* es modificada, se pone un bit de la caché en 1 indicando que la línea de la *caché* está correcta pero la memoria no; finalmente, la línea es escrita en la memoria, pero posiblemente después de sufrir varias escrituras. El protocolo MESI estudiado en clase define cuatro estados para cada línea de la *caché*: *modified*, *exclusive*, *shared* (compartido), *invalid*. Si solo pudiéramos tener tres estados, ¿cuáles estados podrían ser eliminados (solo uno a la vez) y cuáles serían las consecuencias en cada caso?

         - `M:` Ya no se podría marcar una entrada como modificada. Esto implicaría que las *cachés* ya no podrían mantener el valor modificado para escribir sobre su *cachés* directamente, sino que se tendría que escribir directamente en la memoria principal para mantener el valor actualizado. Esto sería básicamente tener un protocolo *write through*.
         - `E:` Ya no se podría marcar una entrada como exclusiva. Esto no supone un gran problema, ya que se puede dejar como estándar el estado `S` y, al hacer *broadcast* de las modificaciones, si ninguna otra *caché* posee la palabra modificada en una de sus líneas, simplemente no ocurre nada. El funcionamiento sigue siendo el mismo.

- `S:` Ya no se podría marcar una entrada como compartida. No obstante, se podría interpretar el estado `E` como el compartido y se tiene la misma solución del punto anterior.
- `I:` Ya no se podrían marcar entradas como inválidas. Esto, en primer lugar, supone un problema para el estado inicial de las *cachés* donde ninguna entrada es inicialmente válida. Habría que utilizar otro de los estados por *default* para no afectar el funcionamiento y definir alguna revisión de parte del controlador para determinar que la *caché* está vacía (afectando considerablemente el desempeño). Por otra parte, al hacer escrituras sería imperante actualizar directamente el valor de todas las *caché* que comparten la palabra modificada, dado que no pueden mantener un estado de invalidez. Esta es la señal **menos conveniente** para modificar.

Importante a notar: Las modificaciones realizadas al eliminar los estados `E` y `S` son, de hecho, un protocolo anterior al `MESI`: el protocolo `MSI`. El protocolo `MESI` no es más que una variante actualizada de él. Existen otros protocolos variantes que son interesantes: `MOSI`, `MOESI` y `MESIF` (enlaces de referencia en cada nombre). Estos tienen la gracia de que aprovechan las transferencia entre *cachés*, haciendo los protocolos más eficientes (recordar que las *caché* son unidades de memoria rápidas).