Ayuda del Simulador SMPCache

1. Repaso de los sistemas de memoria jerárquicos.

Mencionamos a continuación los conceptos más básicos de las memorias caché, que han servido como base para la construcción de este simulador.

La caché es un dispositivo que se encuentra entre la Unidad Central de Proceso (CPU) y la Memoria Principal (MP). Casi todos los computadores actuales poseen caché, algunos más de una vez (jerarquía multinivel). Es una memoria pequeña y rápida que contiene una copia de aquellas posiciones de MP que están siendo utilizadas por la CPU. Se fundamenta en el **principio de localidad**: Cuando la CPU pide referenciar un elemento, éste tenderá a ser referenciado pronto (*localidad temporal*, lo que nos induce a tener la caché como una memoria pequeña y rápida), y tiene cerca unos elementos con alta probabilidad de ser accedidos (*localidad espacial*, que nos lleva a considerar la ubicación de conjuntos de palabras en la caché, o bloques).

En la **Tabla 1** pueden verse algunas de las variables con las que podemos estudiar el rendimiento de la caché.

Símbolo	Descripción
A = Acierto	Un acierto es un acceso con éxito a la caché.
F = Fallo	Un fallo se produce cuando la información buscada no se encuentra en caché.
FA = Frecuencia de aciertos	Es el % de aciertos en accesos a caché.
FF = Frecuencia de fallos	FF=1-FA; Tiene un significado análogo.
Para un programa: **NA** = Número de aciertos. **NF** = Número de fallos. **NC** = Nº total de accesos o referencias a memoria.	Se cumple: FA=NA/NC FF=NF/NC NC=NA+NF

Tabla 1: Algunas de las variables con las que podemos estudiar el rendimiento de la caché.

Consideramos en nuestro estudio una organización de la caché y la memoria principal (MP) basada en una estructura de bloques, según la **Figura 1**.

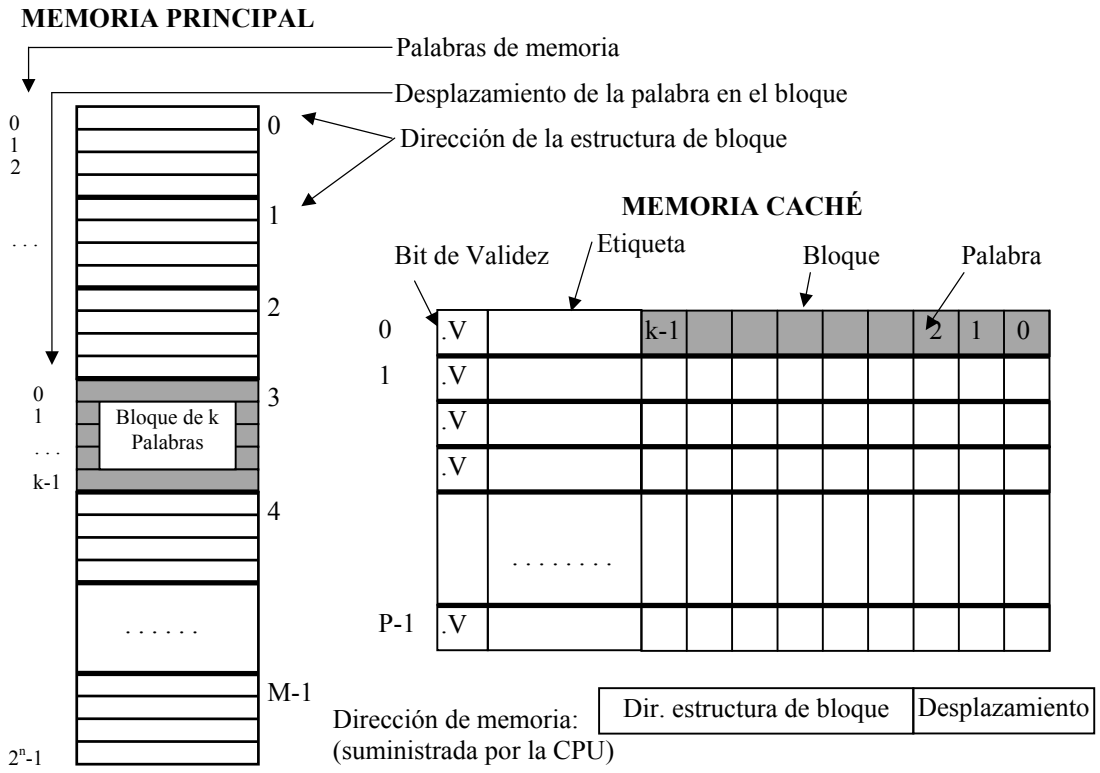


Figura 1: Organización de la caché y la memoria principal basada en una estructura de bloques.

Un bloque es la unidad de información referenciable en la jerarquía de memoria, compuesta de palabras de memoria (leídas o escritas a petición de la CPU, pero como parte de un bloque). Se denomina como **línea o partición de caché** al **bloque de caché** (de k palabras), además de la etiqueta (que identifica la dirección de la estructura de bloque de MP

correspondiente) y el bit de validez (bit que indica si el bloque contiene datos válidos). Llamamos **conjunto** al grupo de bloques de caché cuyas etiquetas son examinadas en paralelo.

La dirección de memoria especifica por completo la localización de una palabra dentro de la memoria principal. Consta de la dirección de la estructura de bloque (que identifica un bloque dentro de la memoria principal) y la dirección del desplazamiento de bloque (que posiciona la palabra dentro del bloque). Esta es la dirección de memoria o dirección suministrada por la CPU. Para "convertirla" a dirección de caché, se interpretará de acuerdo con la función de correspondencia considerada.

Como ya hemos mencionado, el motivo por el que se agrupan las palabras en bloques es para aprovechar el principio de localidad espacial. Los bloques son la unidad de información que van a intercambiarse la memoria principal y la memoria caché (o las cachés entre sí), mientras la CPU solicita lecturas o escrituras sobre una palabra concreta.

La **función de correspondencia** es un algoritmo o procedimiento que asigna a los bloques de la MP posiciones definidas en la caché. Esto es necesario porque existen menos particiones de caché que bloques en MP. La correspondencia se apoya en una lógica de asignación de bloques. Son tres las funciones de correspondencia: directa, totalmente asociativa, y asociativa por conjuntos de n vías. La **correspondencia es directa** cuando a cada bloque de MP solamente le corresponde una partición de caché. Es **totalmente asociativa** cuando un bloque se puede ubicar en cualquier posición de la caché. Finalmente, es **asociativa por conjuntos de N vías** cuando la caché se organiza en un número de conjuntos de N bloques; entonces un bloque de MP se corresponde con un conjunto, pudiéndose ubicar en cualquiera de los bloques que lo componen.

Cuando hay que ubicar un bloque de MP en la caché (reemplazando otro), la correspondencia nos informa de los posibles conjuntos o bloques donde sustituir. Unos **algoritmos de reemplazo**, implementados por hardware, deciden dónde hacer la sustitución (excepto en el caso de correspondencia directa). Las estrategias comunes son: **Aleatoria, LRU** (menos recientemente usado), **FIFO** (primero en entrar, primero en salir) y **LFU** (menos frecuentemente usado).

La operación común de una caché comienza con su **inicialización** (cuando todos los bits de validez se ponen a 0). Tras ello, la caché está preparada para que se realicen en ella operaciones de lectura y escritura. Básicamente, la **operación de lectura** comienza cuando la CPU solicita una lectura de memoria. Se lee primero de caché, y si allí no se encuentra la palabra requerida (*fallo de caché*), se acude a MP, de donde se lee la palabra, y se emplaza el bloque que contiene esa palabra en la caché. En una escritura, la CPU pide reemplazar una porción de bloque (una palabra). Esto implica leer el bloque original, modificar parte de él, y escribir de nuevo el valor del bloque. Hay dos **políticas básicas para escrituras**: **Escritura directa** (si hay acierto, la información se escribe en el bloque de la caché y en el bloque de la MP en paralelo), y **Post-escritura** (si hay acierto, la información se escribe sólo en el bloque de la caché, el bloque modificado se escribe en MP sólo cuando vaya a ser reemplazado).

Los **fallos de caché** se pueden clasificar en **forzosos** (cuando el primer acceso a un bloque no está en la caché, y debe llevarse a la misma), **de capacidad** (cuando la caché no tiene capacidad suficiente para contener todos los bloques necesarios de un programa, produciéndose fallos debido a los bloques que continuamente se reemplazan) y **de conflicto** (cuando a un bloque de caché se le asocian demasiados bloques de MP, generándose fallos de colisión).

2. Función de correspondencia.

La función de correspondencia es un algoritmo que establece una relación entre los bloques de memoria principal y las posiciones que van a ocupar en las cachés.

Básicamente, las funciones de correspondencia se van a englobar en uno de los siguientes tipos, aunque existen otras formas de realizarla. Estos son:

- Correspondencia directa.
- Correspondencia totalmente asociativa.
- Correspondencia asociativa por conjuntos.

Estas son, por otro lado, las funciones de correspondencia que el usuario puede seleccionar dentro de este simulador.

La memoria caché dispone para cada bloque de una serie de bits adicionales que nos dan información acerca de su estado, etc. Como un bloque de memoria caché puede contener diferentes bloques de memoria principal, necesitaremos saber cuál es el que se encuentra almacenado en cada momento. Para ello, se hace uso de una **etiqueta** que estará constituida por una serie de bits que nos permitan identificarlo de forma única.

Para comprobar si un bloque se encuentra en la caché, habrá que comparar las diferentes etiquetas con una obtenida a partir de la dirección deseada, produciéndose un acierto en el caso de que alguna coincida. Estas comparaciones son normalmente realizadas en paralelo, lo que supone una mayor complejidad hardware.

No existe una forma determinada para establecer la etiqueta y como veremos a continuación existen diferentes estrategias.

2.1. Función de correspondencia directa.

Según este tipo de correspondencia, un bloque de memoria principal únicamente podrá ser ubicado en una posición determinada de la memoria caché.

Para realizarla se puede utilizar cualquier función que establezca una relación única entre un bloque de memoria principal y una posición en la caché. Normalmente se utiliza la función módulo (resto de la división entera) que podemos expresar de la siguiente manera:

POS = D módulo N

Donde:

- POS es la posición que ocupará el bloque dentro de la memoria caché.
- D es la dirección del bloque de memoria principal.
- N es el número de bloques que tiene la caché.

De acuerdo a la ecuación anterior, el formato de la dirección de caché será algo parecido al siguiente:

Etiqueta	Indice	Desplazamiento

El **desplazamiento** será el mismo que el que tenga la dirección del bloque de memoria principal, ya que el tamaño de los bloques es igual, tanto en memoria principal como en caché.

El **índice** es el conjunto de bits de la dirección de bloque de memoria principal, que se utilizan para obtener la posición que ocupará el bloque dentro de la caché. No es necesario que ésta se obtenga a partir de todos los bits de la dirección de bloque. Bastará con que el conjunto de bits seleccionados sea igual que el número de bits necesarios para direccionar todos los bloques de caché.

La **etiqueta** será el resto de bits de la dirección de bloque de memoria principal, que no se han utilizado como índice. Son los bits que se comparan a la hora de determinar si un bloque se encuentra en caché o no.

La **Figura 2** muestra un esquema de correspondencia directa.

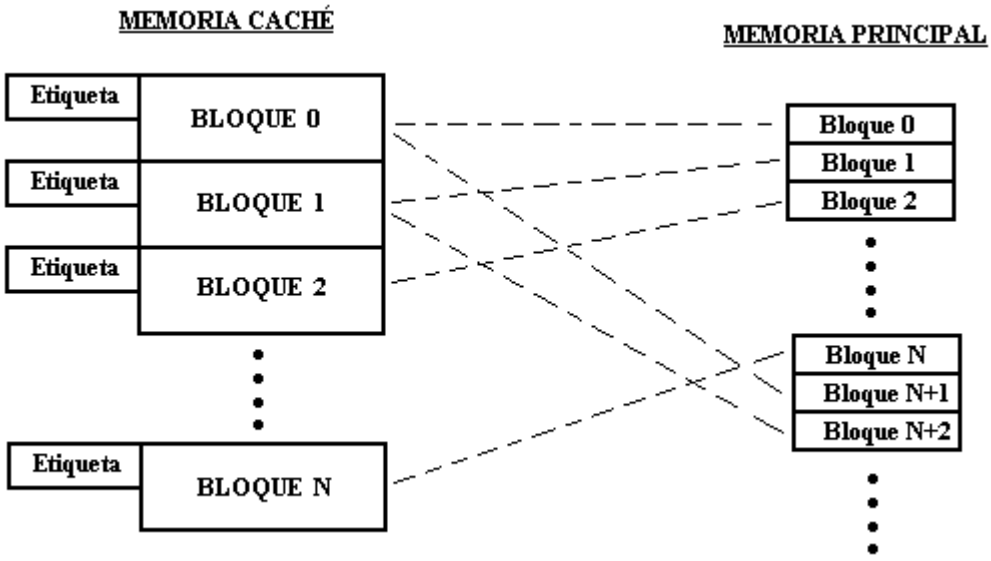


Figura 2: Esquema de correspondencia directa.

Las principales ventajas de esta técnica son su simplicidad y bajo coste. Su principal desventaja se produce en el caso de que se haga uso continuo de dos bloques de memoria que ocupen la misma posición en la caché, se producirán fallos de caché (fallos de conflicto) cada vez que se haga uso de ellos.

Para este tipo de correspondencia, no es necesaria la implantación de una función de reemplazamiento en el sistema, puesto que ya sabemos de antemano cuál va a ser el bloque a sustituir.

2.2. Función de correspondencia totalmente asociativa.

Permite almacenar un bloque de memoria principal en cualquier posición en caché. En este caso la dirección de caché es la misma que la dirección de memoria y la etiqueta contiene la dirección del bloque completa. Por tanto, el formato de la dirección de caché será algo parecido al siguiente:

Etiqueta	Desplazamiento

Como puede estudiarse en la ayuda dedicada a las funciones de reemplazamiento, existen diferentes estrategias a la hora de seleccionar la posición a ocupar.

Mostramos en la **Figura 3** el esquema de este tipo de correspondencia.

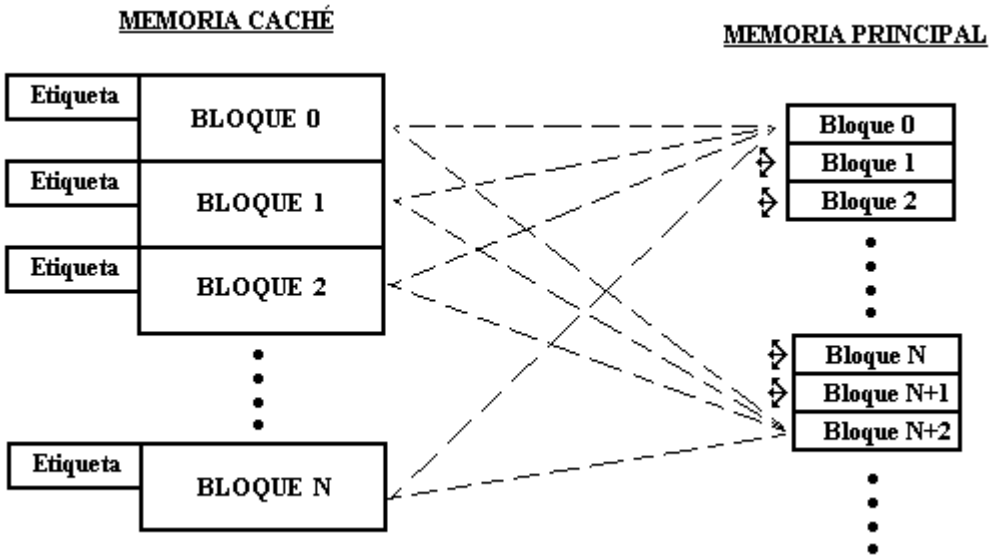


Figura 3: Esquema de correspondencia totalmente asociativa.

La ventaja de este sistema es que un bloque puede ser asignado en cualquier posición. Entre sus desventajas cabe destacar que las búsquedas son más complejas y requieren de una circuitería adicional, ya que se deben realizar las comparaciones de las etiquetas en paralelo, en todos los bloques de caché.

2.3. Función de correspondencia asociativa por conjuntos.

Es una combinación de la función de correspondencia directa y la totalmente asociativa. En ella se divide la caché en diferentes **conjuntos**, que constan cada uno de un número determinado de bloques de caché (un bloque de caché = una **vía**). Se seleccionará uno de los conjuntos a través de una función de correspondencia directa y una vez que se haya hecho, se elegirá uno de los bloques incluidos en él a través de una función de correspondencia totalmente asociativa.

En este caso, la dirección de caché sería igual que en la correspondencia directa, con la diferencia de que el índice se refiere a un conjunto en lugar de a un bloque:

Etiqueta	Indice	Desplazamiento

A medida, que se vaya incrementando el número de vías (bloques) por conjunto, disminuirá el número de conjuntos, por lo que también lo hará el tamaño del índice, y aumentará el tamaño de la etiqueta.

Mostramos el esquema de este sistema en la **Figura 4**.

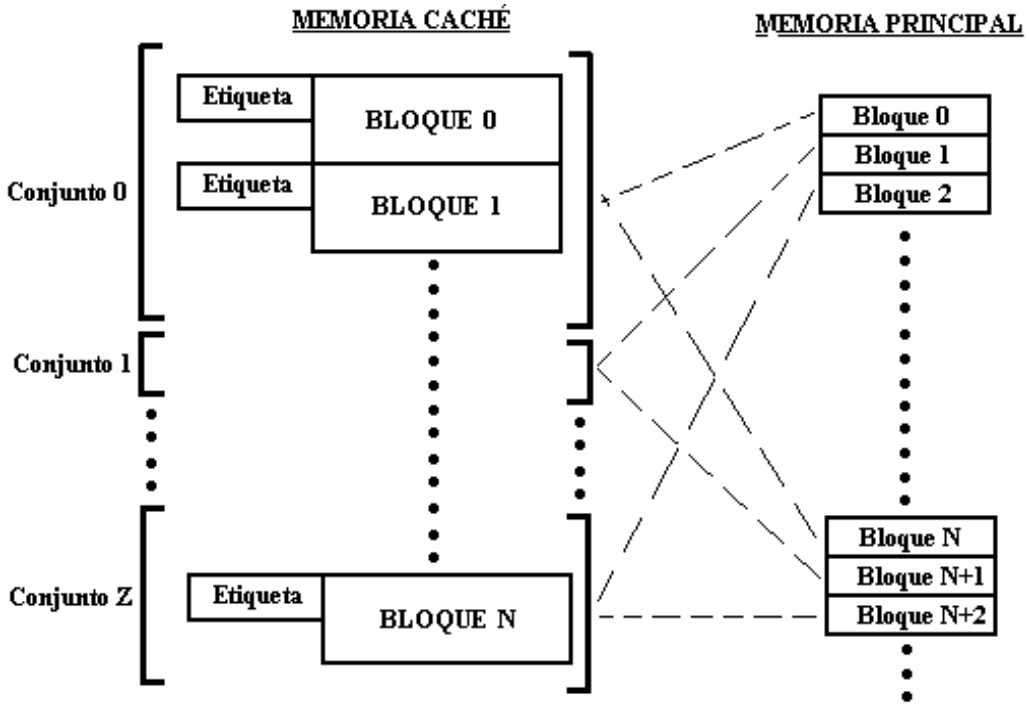


Figura 4: Esquema de correspondencia asociativa por conjuntos.

A la hora de realizar las búsquedas, se localizará a través del índice el conjunto de memoria donde podría encontrarse el bloque y se comprobarán en paralelo las etiquetas de los bloques almacenados en ese conjunto con la del bloque a buscar.

Existe una regla empírica, según la cual, una caché con función de correspondencia directa de tamaño T, tiene aproximadamente la misma FF (frecuencia de fallos) que una caché con función de correspondencia asociativa por conjuntos de 2 vías de tamaño T/2.

Normalmente se suele utilizar esta función de correspondencia con conjuntos de 2 vías, con lo que se mejora la tasa de acierto respecto a la directa. También se utilizan conjuntos de 4 vías, con lo que obtenemos una mejora con un coste razonable. A partir de 4 vías, el beneficio que produce no resulta rentable, ya que se incrementa notablemente el coste.

La forma en que se selecciona el bloque de entre todos los que incluye el conjunto, dependerá de la función de reemplazamiento de la que disponga el sistema.

3. Algoritmo de reemplazamiento.

Su función es la de seleccionar un bloque de caché de entre un grupo de ellos, de acuerdo a un determinado criterio que va a depender del método de reemplazamiento utilizado. Este algoritmo es muy importante y habrá que usarlo cada vez que haya que sustituir un bloque de caché por otro.

La función de reemplazamiento sólo tiene sentido cuando se utilizan funciones de correspondencia asociativa por conjuntos o totalmente asociativa, ya que si estamos utilizando correspondencia directa, solamente existiría un único bloque candidato para ser sustituido.

A medida que va aumentando el tamaño de la caché, la función de reemplamiento tiene una menor importancia, porque la tasa de aciertos va a ser muy alta.

Existen una gran diversidad de funciones de reemplazamiento. Las más utilizadas son:

- Aleatoria.
- LRU.
- FIFO.
- LFU.

De entre todas ellas, las más utilizadas son la aleatoria y la LRU, siendo ésta última bastante mejor que la aleatoria, cuando los tamaños de caché son pequeños.

3.1. Algoritmo de reemplazamiento aleatorio.

Con esta estrategia, elegiremos uno de los posibles bloques de forma aleatoria. Así conseguiremos que todos los bloques de caché sean sustituidos uniformemente.

Este método resulta fácil de implementar, pero tiene el inconveniente de que bloques que han sido utilizados recientemente, o que son utilizados con mucha frecuencia, pueden ser sustituidos.

3.2. Algoritmo de reemplazamiento LRU (menos recientemente usado).

Este método explota el principio de localidad temporal, que dice que bloques que han sido accedidos hace poco es probable que vuelvan a ser utilizados pronto. De entre los posibles bloques a sustituir se elegirá aquel que lleve más tiempo sin ser utilizado. Esta estrategia requiere de un hardware más complejo, ya que ha de registrarse la última vez que fue accedido un bloque.

Cuando el número de bloques de caché es muy grande, esta estrategia se encarece bastante, ya que hay que hacer un gran número de comparaciones al hacer la selección.

3.3. Algoritmo de reemplazamiento FIFO (1º en entrar 1º en salir).

En este método el bloque sustituido será aquel que llegó antes a la caché.

3.4. Algoritmo de reemplazamiento LFU (menos frecuentemente usado).

En este método el bloque a sustituir será aquel que se acceda con menos frecuencia. Habrá que registrar por lo tanto, la frecuencia de uso de los diferentes bloques de caché. Esta

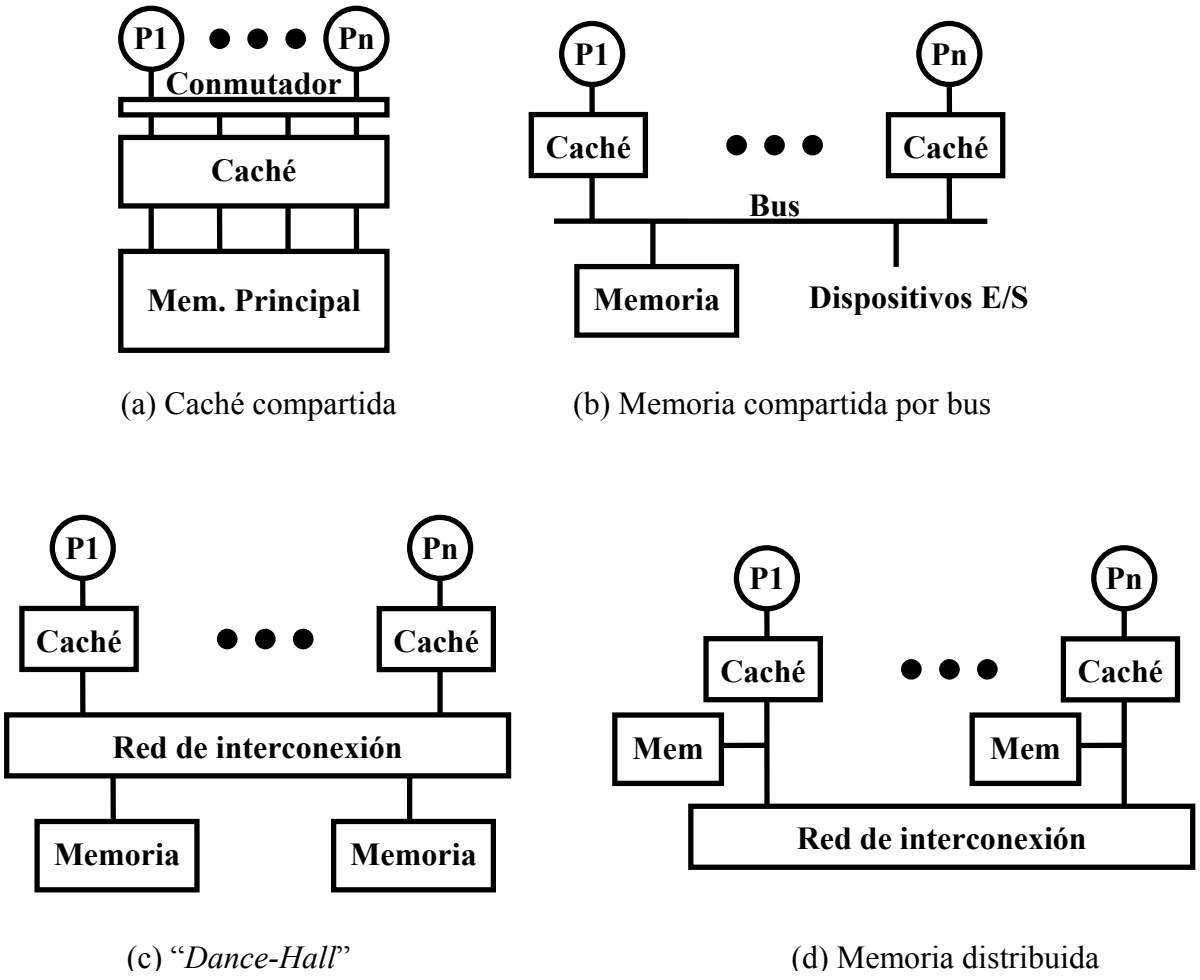
frecuencia se debe ir recalculando cada vez que se realice una operación en caché, lo que hace que este método tenga un coste adicional elevado.

La frecuencia de uso se suele calcular dividiendo el número de veces que se ha usado un bloque por el tiempo que hace que entró en caché.

4. Las posibles jerarquías de memoria en multiprocesadores.

La forma más frecuente de arquitectura paralela son los multiprocesadores de pequeña escala que proporcionan un espacio de direcciones físicas global y acceso simétrico a toda la memoria principal desde cualquier procesador. Estos a menudo se denominan **Multiprocesadores Simétricos** o SMP (del inglés ***S**ymmetric **M**ulti**P**rocessor*). Los SMPs dominan el mercado de los servidores y cada vez son más comunes en el de ordenadores de tipo "desktop". También son importantes en la construcción de bloques básicos para sistemas de mayor escala.

En general, las jerarquías de memoria en multiprocesadores se encuentran principalmente en una de las cuatro categorías mostradas en la **Figura 5**, y que corresponden aproximadamente con la escala de multiprocesadores que estamos considerando. Las tres primeras categorías están relacionadas con multiprocesadores simétricos (toda la memoria principal está igual de lejos para cada procesador), mientras la cuarta no.



(a) Caché compartida

(b) Memoria compartida por bus

(c) "*Dance-Hall*"

(d) Memoria distribuida

Figura 5. Jerarquías de memoria comúnmente encontradas en multiprocesadores.

En la primera categoría, la aproximación de "caché compartida", la interconexión se localiza entre los procesadores y una caché compartida de primer nivel, la cual continúa las conexiones hacia un subsistema de memoria principal compartido. Esta aproximación se ha utilizado para conectar un número muy escaso de procesadores (de 2 a 8). Esta técnica se aplica sólo a muy baja escala porque la interconexión entre los procesadores y la caché compartida de primer nivel se convierte probablemente en un camino crítico que determina la duración del ciclo de reloj de una máquina. Además, la caché compartida debe repartir un ancho de banda "inmenso" entre los procesadores conectados a ella.

En la segunda categoría, la aproximación de "memoria compartida por bus", la interconexión se establece a través de un bus compartido localizado entre las cachés privadas de cada procesador y el subsistema de memoria principal compartida. Esta aproximación se ha utilizado ampliamente para multiprocesadores de escala baja a media, constituidos incluso por unas pocas decenas de procesadores. Es el modelo dominante de máquina paralela que se vende en la actualidad, y esencialmente todos los microprocesadores modernos han realizado considerables esfuerzos de diseño para poder soportar configuraciones de memoria compartida con "caché coherente". El límite de escala viene principalmente establecido por las limitaciones del ancho de banda del bus y el sistema de memoria compartida.

La tercera categoría también sitúa la interconexión entre las cachés y la memoria principal, pero la interconexión ahora es una red punto-a-punto escalable y la memoria se divide en muchos módulos lógicos. Esta aproximación es escalable y simétrica, toda la memoria principal está igual de lejos (remota) de cada procesador; pero su limitación se debe a que toda la memoria está, de hecho, muy lejos de cada procesador. Por este motivo las arquitecturas de este tipo han caído en desuso.

La cuarta categoría, la aproximación de "memoria distribuida", no es simétrica. Entre los nodos de procesamiento existe una interconexión escalable, pero cada nodo tiene su porción local de la memoria principal global. Porción que es accedida más rápidamente. Explotando la localidad en la distribución de los datos, se espera que la mayoría de los accesos se satisfagan en la memoria local y no haya que recorrer la red.

En todos estos casos, las cachés juegan un papel esencial en la reducción del tiempo de acceso medio a memoria desde el punto de vista del procesador, y en la reducción de los requerimientos de ancho de banda que cada procesador necesita para la interconexión y el sistema de memoria compartidos. Sin embargo, en todas, excepto en la aproximación de caché compartida, el uso de cachés privadas por cada procesador da lugar a un importante reto, denominado como **coherencia de caché**. El problema ocurre cuando un bloque de memoria está presente en las cachés de uno o más procesadores, y otro procesador modifica ese bloque de memoria. A menos que se tome algún tipo de acción especial, esos otros procesadores seguirán accediendo a una copia obsoleta del bloque, que no es otra que la que se encuentra en sus cachés.

En este simulador nos hemos centrado en las máquinas de escala pequeña a moderada que se organizan como en la Figura 5(b). Estas máquinas explotan las propiedades fundamentales de un bus para solventar el problema de coherencia de caché. No hemos tenido en cuenta las categorías (a) y (c), pues como hemos dicho su uso es mínimo. Además, la categoría (a) carece del problema de coherencia de caché. También se descartó la categoría (d) por no ser propia de multiprocesadores simétricos.

5. El problema de la coherencia de caché.

Piense por un momento en su modelo intuitivo sobre qué debería hacer una memoria.

Debería proporcionar un conjunto de localizaciones que mantuvieran valores y cuando una localización se leyera debería retornar el último valor escrito en esa localización. Esta es la propiedad fundamental de la memoria con la que contamos en los programas secuenciales cuando usamos la memoria para comunicar un valor desde un punto (en un programa) donde se computa a otros donde se utiliza. Contamos con esa misma propiedad en un sistema de memoria cuando utilizamos un espacio de direcciones compartido para comunicar datos entre hilos (*threads*) o procesos en un procesador. Una lectura retorna el último valor escrito en esa localización, sin tener en cuenta qué proceso lo escribió. Confiamos que se cumpla esta misma propiedad cuando los dos procesos se ejecutan en procesadores distintos. La existencia de caché no interfiere en la ejecución de múltiples procesos en un solo procesador, ya que todos ven la memoria a través de la misma jerarquía de caché. El problema aparece tan pronto como ejecutamos los procesos en dos procesadores distintos, que ven la memoria a través de diferentes cachés. Uno podría ver el valor nuevo en su caché, mientras el otro todavía ve el valor antiguo.

Realmente, algunos problemas aparecen incluso en uniprocesadores cuando tratamos las E/S. Por ejemplo, un dispositivo DMA que gestione el bus de memoria principal, podría leer un valor obsoleto de una posición en memoria principal, porque el último valor para esa posición se encuentre en la caché post-escritura de un procesador. Similarmente, cuando el dispositivo DMA escribe en una posición de memoria principal, a menos que se tome alguna acción especial, el procesador podría continuar viendo el valor antiguo si esa posición fue previamente cargada en su caché. Como las operaciones de E/S son mucho menos frecuentes que las operaciones de memoria, se han adoptado varias soluciones simples en uniprocesadores. Por ejemplo, los segmentos del espacio de memoria utilizados para E/S se podrían marcar como imposibles de gestionar por caché, o el procesador podría usar siempre cargas/almacenamientos no gestionados por caché para posiciones utilizadas en la comunicación con dispositivos de E/S. Para dispositivos de E/S que transfieren grandes bloques de datos, tales como los discos, se utiliza a menudo el apoyo del propio sistema operativo para asegurar la coherencia. En muchos sistemas la página de memoria desde/hacia la que se transferirán los datos se reemplaza de la caché del procesador por parte del sistema operativo antes de que se permita la operación de E/S. En otros sistemas, se hace que todo el tráfico de E/S fluya a través de la jerarquía de caché del procesador, así se mantiene la coherencia. Esto, lógicamente, tiene el problema de que contamina la jerarquía de caché con datos que podrían no ser de interés inmediato para el procesador. En la actualidad, básicamente todos los microprocesadores proporcionan soporte para la coherencia de caché; además de crear un chip “preparado para multiprocesadores”, esto también resuelve el problema de la coherencia en E/S.

El problema de la coherencia de caché en multiprocesadores, en el cual se centra este simulador, es un problema tan extendido como crítico. El problema podemos ilustrarlo mediante el ejemplo de la **Figura 6**.

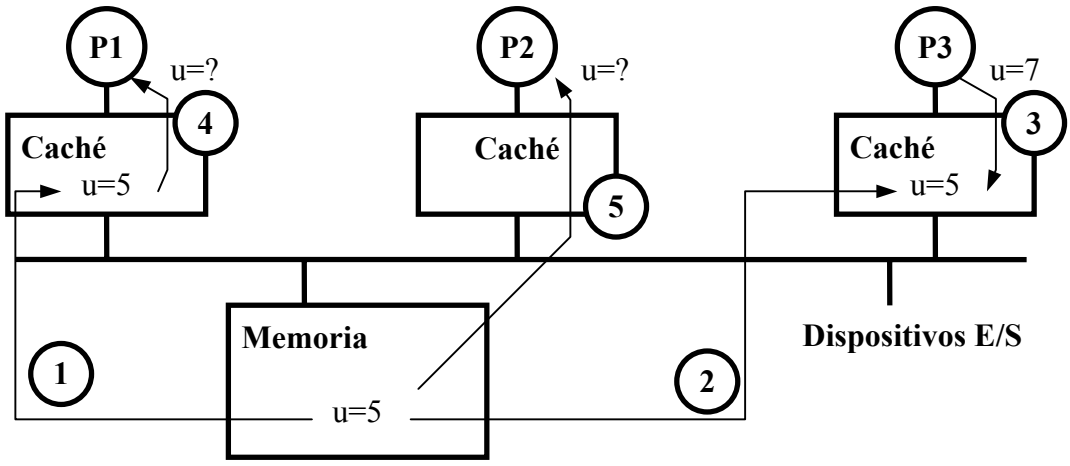


Figura 6. Ejemplo del problema de coherencia de caché.

La figura muestra tres procesadores con cachés conectadas por un bus a una memoria principal compartida. Se realiza por parte de los procesadores una serie de accesos a la posición **u**. Primero, el procesador **P1** lee **u**, trayendo una copia a su caché. Después el procesador **P3** lee **u**, con lo que también pone una copia en su caché. Tras esto, el procesador **P3** escribe en la posición **u** cambiando su valor de 5 a 7. Con una caché de escritura directa, esto causaría que la posición en memoria principal se actualizara; sin embargo, cuando el procesador **P1** vuelve a leer la posición **u** (acción 4), desafortunadamente leerá el valor obsoleto (5) de su propia caché en lugar del valor correcto (7) de la memoria principal. La situación es incluso peor si las cachés son de post-escritura, ya que la escritura de **P3** podría simplemente establecer el bit de modificado asociado con el bloque de caché que mantiene la posición **u**, y no actualizaría la memoria principal directamente. Sólo cuando este bloque de caché sea posteriormente reemplazado de la caché de **P3**, su contenido sería escrito en la memoria principal. No sólo **P1** leerá el valor obsoleto, sino que cuando el procesador **P2** lea la posición **u** (acción 5), no la encontrará en su caché y leerá el valor obsoleto de 5 de la memoria principal, en lugar del 7. Finalmente, si múltiples procesadores escriben distintos valores para la posición **u** en sus cachés de post-escritura, el valor final que quedará en la memoria principal se determinará por el orden en el que los bloques de caché que contienen la posición **u** se reemplazan, y no tendrá nada que ver con el orden en el que las escrituras de la posición **u** ocurrieron.

Claramente, el comportamiento que acabamos de describir viola nuestra noción intuitiva de lo que la memoria debería hacer. A diferencia de las E/S, las lecturas y escrituras de variables compartidas se espera que sean operaciones frecuentes en multiprocesadores; es la forma en la que múltiples procesos pertenecientes a una aplicación paralela se comunican entre sí, por lo que no nos convienen las soluciones planteadas para el caso de la coherencia en E/S (desactivación de la caché, etc.). Dicho de otro modo, no nos convienen los **esquemas de coherencia caché basados en software**. En su lugar, nos hemos centrado en los esquemas de coherencia caché basados en hardware, es decir, la coherencia de caché necesita ser tratada como un asunto básico en el diseño del hardware. Por ejemplo, las copias de caché obsoletas de una localización compartida deben eliminarse cuando esa localización se modifica, ya sea invalidándolas o actualizándolas con el nuevo valor. Dentro de los esquemas de coherencia caché basados en hardware podemos diferenciar entre **arquitecturas de red con caché coherente** y **protocolos de coherencia caché**. Las arquitecturas de red proponen

multiprocesadores con una jerarquía de buses, en los que el tráfico de la red se reduce mediante protocolos de coherencia caché jerárquicos. Estas arquitecturas quedan fuera de nuestros objetivos, pues se alejan de lo que hemos definido como SMP con memoria compartida por bus, más bien son una combinación de varios de estos SMPs. Sin embargo, si hemos estudiado detenidamente el concepto de protocolo de coherencia caché. Hay dos clases de protocolos para mantener la coherencia caché de múltiples procesadores:

- **Basados en directorio**: La información sobre un bloque de memoria física se mantiene en una única posición.
- **Espionaje (snooping)**: Cada caché que tiene una copia de datos de un bloque de memoria física también tiene una copia de la información sobre él. Estas cachés se utilizan habitualmente en sistemas de memoria compartida con un bus común, es decir, en los que se centra este simulador.

En los protocolos basados en directorio hay lógicamente un único directorio que contiene el estado de cada bloque en memoria principal. La información del directorio puede incluir las cachés que tengan copias del bloque, si está modificado (*dirty*), etc. Las entradas al directorio pueden estar distribuidas para que diferentes peticiones puedan ir a diferentes memorias, reduciendo así la contención. Sin embargo, conservan la característica de que el estado compartido de un bloque está siempre en una única posición conocida.

Los protocolos de espionaje se hicieron populares con los multiprocesadores que utilizaban microprocesadores y cachés, en sistemas de memoria compartida, porque pueden utilizar una conexión física preexistente: el bus a memoria. Es decir, parecen más adecuados para los sistemas en los que se centra este simulador, por ello, nos hemos concentrado en protocolos de espionaje (de todas formas, las mismas ideas se aplican a las cachés basadas en directorio, excepto que la información sobre el estado de las cachés se gestiona de forma diferente, y se involucran sólo si el directorio indica que tienen una copia de un bloque cuyo estado debe cambiar). Además, el espionaje tiene una ventaja sobre los protocolos de directorio, y ésta es que la información de coherencia es proporcional al número de bloques de una caché en lugar de al número de bloques de memoria principal.

6. Coherencia de caché mediante espionaje del bus (*bus snooping*).

Una solución simple y elegante al problema de la coherencia de caché surge de la propia naturaleza de un bus. El bus es un conjunto simple de cables que conectan varios dispositivos, cada uno de los cuales puede observar cada transacción en el bus, por ejemplo cada lectura o escritura en el bus compartido. Cuando un procesador genera una petición a su caché, el controlador examina el estado de la caché y realiza la acción adecuada, que puede incluir generar transacciones de bus para acceder a memoria. La coherencia se mantiene teniendo a cada controlador de caché “espiando” (vigilando) en el bus y monitorizando las transacciones, como se ilustra en la **Figura 7**.

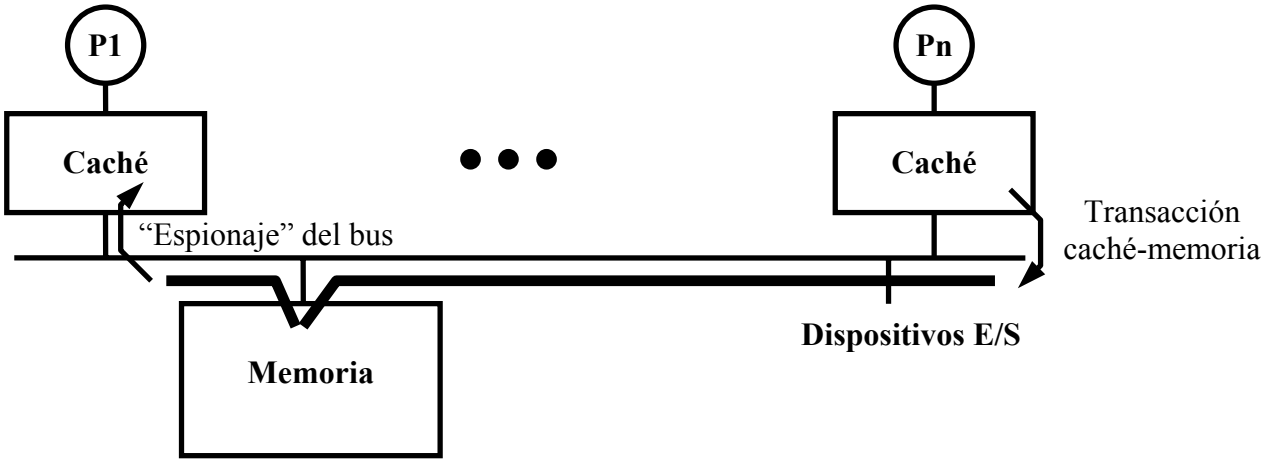


Figura 7. Multiprocesador con coherencia de caché mediante espionaje del bus.

El controlador de caché que "espía" también realizará cierta acción si una transacción del bus le es relevante, es decir, implica un bloque de memoria del cual tiene una copia en su caché. De hecho, como la asignación y reemplazamiento de datos en cachés se gestiona con granularidad de un bloque de caché (generalmente con longitud de varias palabras) y las búsquedas fallidas en caché son de bloques de datos, la mayoría de las veces la coherencia también se mantiene con la granularidad de un bloque de caché. Es decir, o bien un bloque entero de caché está en un estado válido o no lo está todo él. Por tanto, el bloque de caché es la granularidad (unidad) de asignación de caché, de transferencia de datos entre cachés y de coherencia.

El ejemplo más simple de mantenimiento de la coherencia es un sistema que posee cachés de un solo nivel y escritura directa. Es básicamente la aproximación seguida por los primeros SMPs basados en buses comerciales a mediados de los 80. En este caso, cada instrucción de almacenamiento causa que aparezca una transacción de escritura en el bus, por lo que cada caché observa todo almacenamiento. Si una caché que está espiando tiene una copia del bloque, invalida (o actualiza) su copia. La siguiente vez que acceda al bloque, verá el valor más reciente escrito sobre el bus. La memoria siempre contiene datos válidos, por lo que la caché no necesita realizar ningún tipo de acción cuando observa una lectura en el bus.

En general, un esquema de coherencia de caché basado en la técnica de espionaje del bus asegura que:

- Todas las transacciones "necesarias" aparecen en el bus.
- Cada caché monitoriza el bus para en las transacciones que le sean relevantes tomar las acciones oportunas.

El chequeo para determinar si una transacción del bus es relevante para una caché es esencialmente la misma comparación de etiquetas (de bloques de caché) que se realiza para una petición (de datos) por parte del procesador. La acción adecuada podría implicar invalidar o actualizar el contenido o estado de ese bloque de memoria, y/o suministrar el último valor para ese bloque de memoria desde la caché al bus.

Un *protocolo de espionaje de coherencia de caché* une dos facetas básicas de la arquitectura del computador: las transacciones en el bus y el diagrama de transición de

estados asociado con un bloque de caché. Recuerde que una transacción en el bus consiste en tres fases: arbitración, comando/dirección y datos. En la fase de arbitración, los dispositivos que desean realizar (o mandar) una transacción indican su petición del bus y el árbitro del bus selecciona una de éstas y responde mediante su señal de asignación del bus. Tras la asignación, el dispositivo coloca el comando, por ejemplo leer o escribir, y la dirección asociada en las líneas de control y dirección del bus. Todos los dispositivos observan la dirección y uno de ellos reconece que es el responsable para esa dirección particular. Para una transacción de lectura, la fase de dirección viene seguida con la transferencia de datos. Las transacciones de escritura varían de un bus a otro en que los datos se transfieran durante o después de la fase de dirección. Para muchos buses, el dispositivo esclavo (servidor) puede introducir una señal de espera para retener la transferencia de datos hasta que esté preparado. Esta señal de listo es diferente de cualquier otra señal del bus, porque es un OR-cableado a través de todos los procesadores, es decir, es un 1 lógico si cualquier dispositivo la emite. El dispositivo maestro (cliente) no necesita saber qué dispositivo esclavo está participando en la transferencia, sólo si hay alguno y está listo.

La segunda noción básica es que cada bloque en una caché de uniprocesador tiene un estado asociado, junto con la etiqueta y los datos, que indica la disposición del bloque, por ejemplo, no válido, válido, modificado. La política de caché se define por el *diagrama de transición de estados de un bloque de caché*. Las transiciones de un bloque suceden cuando se accede a una dirección dentro del bloque. Para una caché de escritura directa y sin asignación-en-escritura (*no-write-allocate*), sólo se necesitan dos estados: válido y no válido. Inicialmente todos los bloques son no válidos (lógicamente, todos los bloques de memoria que no están residentes en caché pueden verse como que se encuentran en el estado “no válido”). Cuando un procesador produzca un fallo de caché en una operación de lectura, se generará una transacción en el bus para cargar el bloque desde la memoria y el bloque se marcará como válido. Las escrituras generan una transacción en el bus para actualizar la memoria y el bloque de caché si está presente en el estado válido. Las escrituras nunca cambian el estado del bloque (si un bloque se reemplaza, podría marcarse como no válido hasta que la memoria proporcione el nuevo bloque, con lo cual vuelve a ser válido). Una caché de post-escritura necesita un estado adicional, indicando si un bloque está modificado o “sucio”.

En un esquema de espionaje para coherencia de caché, cada controlador de caché recibe dos conjuntos de entradas: las peticiones de memoria invocadas por el procesador, y las informaciones espiadas en el bus sobre transacciones de otras cachés. En respuesta a estas entradas, el controlador actualiza el estado del bloque apropiado en la caché; Responde al procesador con los datos solicitados, posiblemente generando nuevas transacciones en el bus para obtener los datos; Responde a las transacciones en el bus generadas por otros actualizando su estado y, a veces, interviene completando la transacción en curso. El protocolo de espionaje es, por tanto, un algoritmo distribuido representado por una colección de esas *máquinas de estado finitas* que cooperan. Este protocolo se especifica mediante los siguientes componentes:

1. El conjunto de estados asociados con los bloques de memoria en las cachés locales.
2. El diagrama de transición de estados que toma como entradas el estado actual y la petición del procesador o la transacción en el bus observada, y produce como salida el siguiente estado para el bloque de caché.
3. Las acciones reales asociadas con cada transición de estado, que se determinan en parte por el conjunto de acciones factibles definidas según el diseño del bus,

la caché y el procesador.

El diagrama de estados de la **Figura 8** describe un protocolo simple basado en invalidación para una caché coherente de escritura directa y sin asignación-en-escritura.

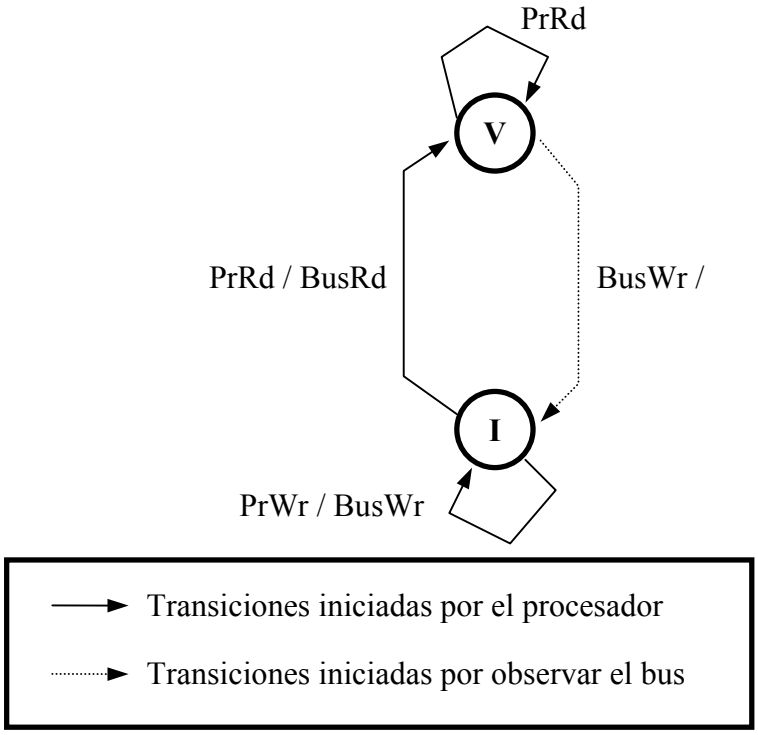


Figura 8. Coherencia mediante espionaje para un multiprocesador con cachés de escritura directa y sin asignación-en-escritura (*no-write-allocate*).

Como en el caso uniprocesador, cada bloque de caché tiene sólo dos estados: no válido (I, de inválido) y válido (V). Las transiciones están etiquetadas con la entrada causante de la transición y la salida generada con la transición (la notación A/B significa que si observas A entonces generas la transacción B). Desde el punto de vista del procesador, las peticiones pueden ser de lectura (PrRd) o escritura (PrWr). Desde el punto de vista del bus, el controlador podría observar/generar transacciones en el bus de lectura (BusRd) o escritura (BusWr). Por ejemplo, cuando en una petición de lectura del procesador (PrRd) se produce un fallo en caché (bloque no válido), se genera una transacción BusRd, y tras completar esta transacción el bloque pasa al estado "válido". Siempre que el procesador solicite una escritura (PrWr) en una posición, se genera una transacción en el bus (BusWr) que actualiza esa posición en la memoria principal sin cambiar el bloque de estado. La mejora clave respecto al diagrama de estados de un uniprocesador es que cuando el "espía" (vigilante) del bus observa una transacción de escritura para un bloque de memoria que se encuentra en la caché local, establece el estado de caché para ese bloque a no válido descartando así su copia. La figura muestra estas transiciones inducidas por el bus con línea punteada. Por extensión, si cualquier procesador genera una escritura para un bloque que está en la caché de cualquier otro, todos los otros invalidarán sus copias del bloque. La colección de cachés implementa de esta forma una disciplina de un único escritor, múltiples lectores.

El problema de la aproximación de escritura directa es que cada instrucción de almacenamiento va a la memoria, motivo por el que **los microprocesadores más modernos**

usan cachés con post-escritura. Este problema se agrava en el entorno multiprocesador, ya que cada almacenamiento desde cada procesador consume un ancho de banda precioso del bus compartido, dando lugar a una baja escalabilidad de tales multiprocesadores. Veámoslo más claramente con un ejemplo. Consideremos un procesador RISC superescalar que proporciona dos instrucciones por ciclo y ejecutándose a 200 Mhz. Supongamos que el CPI (ciclos de reloj por instrucción) medio para este procesador es 1, el 15% de todas las instrucciones son almacenamientos, y cada almacenamiento escribe 8 bytes de datos. ¿Cuántos procesadores sería capaz de soportar un bus de 1 Gb/s sin saturarse? Veamos la solución. Un único procesador generará 30 millones de almacenamientos por segundo, por lo que el ancho de banda total usado para escrituras es de 240 Mbytes de datos por segundo por procesador (ignorando las direcciones y otras informaciones, e ignorando las lecturas que fallen en caché). Un bus de un Gb/s soportará sólo alrededor de cuatro procesadores.

Para la mayoría de las aplicaciones, una caché de post-escritura podría absorver la inmesa mayoría de las escrituras. Sin embargo, si las escrituras no van a memoria no generan transacciones en el bus, y no queda claro cómo las otras cachés observarán estas modificaciones y mantendrán la coherencia de caché. Un problema algo más delicado es que si se permite que las escrituras ocurran en diferentes cachés concurrentemente, no existe un orden obvio de la secuencia de escrituras. Necesitaremos protocolos de coherencia de caché más sofisticados que hagan visibles los eventos "críticos" en las otras cachés y aseguren la *serialización de escritura*.

Este simulador se ha centrado en el diseño de varios protocolos de espionaje que hacen un uso eficiente del limitado ancho de banda del bus compartido. Todos los protocolos estudiados utilizan cachés de post-escritura, permitiendo que varios procesadores escriban diferentes bloques en sus cachés locales de manera concurrente y sin transacciones en el bus.

Recordemos que con una caché post-escritura (o escritura con retardo) en un uniprocesador, un fallo de escritura en caché por parte del procesador causa que la caché lea el bloque entero de la memoria, actualice una palabra, y retenga el bloque como "modificado" (o sucio) por lo que sería escrito con retardo en un reemplazamiento posterior. En un multiprocesador, el estado modificado también se utiliza por los protocolos para indicar la posesión del bloque. En un fallo de escritura en caché se utiliza una forma especial de transacción de lectura, una lectura exclusiva, para indicar a las otras cachés el impedimento de escritura y adquirir la posesión del bloque. Esto pondría el bloque en la caché en estado modificado, donde podría ahora ser escrito. Múltiples procesadores no pueden escribir el mismo bloque concurrentemente, ya que podría conducir a valores inconsistentes: Las transacciones en el bus de lectura exclusiva generadas por sus escrituras serán serializadas por el bus, por lo que sólo uno de ellos puede tener la posesión del bloque en cada instante. Las acciones de coherencia de caché se dirigen por estos dos tipos de transacciones: lectura y lectura exclusiva. Finalmente, cuando el bloque se descarte de la caché, los datos se escribirán en memoria, pero este proceso no es causado por una operación de memoria sobre ese bloque y es casi inherente al protocolo. Se han creado muchos protocolos para cachés con post-escritura, nosotros sólo hemos estudiado las alternativas básicas.

También consideramos en este simulador los protocolos basados en actualización (y no invalidación). En los protocolos basados en actualización, cuando un procesador escribe en una localización compartida, su valor es actualizado en las cachés de todos los otros procesadores que mantienen ese bloque de memoria (se trata de un entorno con difusión de escritura; Los diseños con difusión de lectura también se han investigado, en ellos la caché que contiene la copia modificada la pone en el bus en una lectura, en este punto todas las otras copias también se actualizan). Así, cuando estos procesadores acceden posteriormente a ese bloque, pueden hacer referencia a los nuevos datos con una baja latencia. Las cachés de todos

los otros procesadores se actualizan con una única transacción en el bus, por tanto, conservando el ancho de banda cuando hay varios procesadores que comparten el bloque. Por contra, con los protocolos basados en invalidación, en una operación de escritura el estado de caché de ese bloque de memoria se establece a no válido en todas las otras cachés. Las escrituras subsiguientes a ese bloque por el mismo procesador no crean tráfico adicional en el bus, hasta que el bloque es leído por otro procesador. Esto es interesante cuando se espera que un único procesador realice múltiples escrituras sobre el mismo bloque de memoria antes de que otros procesadores lean los contenidos de ese bloque de memoria. En general, las estrategias basadas en invalidación se han mostrado más robustas y son por tanto proporcionadas como el protocolo por defecto por la mayoría de vendedores. Algunos vendedores proporcionan un protocolo de actualización como una opción a ser usada selectivamente para bloques que correspondan a páginas o estructuras de datos concretas.

Las elecciones realizadas respecto al protocolo y estrategias de caché con actualización frente a invalidación afectan directamente a la selección de estados, el diagrama de transición de estados, y las acciones asociadas. Se dispone, por tanto, de una flexibilidad sustancial para la arquitectura del computador en las tareas de diseño a este nivel. En lugar de estudiar todas las posibles elecciones, hemos considerado los tres protocolos de coherencia habituales, que mostrarán las opciones de diseño:

- El protocolo de 3 estados MSI.
- El protocolo de 4 estados MESI.
- El protocolo de 4 estados Dragon.

6.1. El protocolo de 3 estados MSI.

Es un protocolo básico de invalidación para cachés con post-escritura. Es muy similar al usado en la serie de máquinas multiprocesador Silicon Graphics 4D. El protocolo utiliza los tres estados requeridos por cualquier caché con post-escritura para distinguir los bloques válidos que no se han modificado (limpios) de aquellos que se han modificado (sucios). Concretamente, los estados son: no válido (**I**, de inválido), compartido (**S**, del inglés "***S****hared*"), y modificado (**M**). De aquí el propio nombre del protocolo. No válido tiene el significado obvio. Compartido significa que el bloque se encuentra sin ser modificado en una o más cachés de procesador; las copias en memoria principal y en estas cachés se encuentran todas actualizadas. Modificado, también denominado *sucio*, significa que sólo un procesador tienen una copia válida del bloque en su caché; la copia en memoria principal está obsoleta y ninguna otra caché puede tener el bloque con el estado compartido o modificado. El truco en este tipo de protocolos está en que antes de que un bloque pueda ser actualizado y colocado en el estado modificado, todas las otras copias potenciales deben ser invalidadas. Esto implica una transacción en el bus explícita, que sirva para ordenar la actualización a la vez que cause las invalidaciones y asegure que la escritura es visible para los demás.

Se asume que el procesador emite dos tipos de peticiones, lecturas (PrRd) y escrituras (PrWr). La lectura o escritura podría ser a un bloque de memoria que existe en la caché o a uno que no está en ella. En este último caso, la copia del bloque de memoria en la caché tendrá que ser reemplazada por el bloque nuevo solicitado, y si el bloque existente en esa posición fue modificado tendrá que ser post-escrito en memoria principal.

Además se asume que el bus permite las siguientes transacciones:

- Lectura (BusRd): El controlador de caché pone la dirección en el bus y pide una copia del bloque sin intención de modificarlo. El sistema de memoria

(posiblemente otra caché) suministra los datos. Esta transacción se genera por una PrRd que falla en la caché, y el procesador espera una respuesta de datos como resultado de la misma.

- Lectura exclusiva (BusRdX): El controlador de caché pone la dirección en el bus y solicita una copia *exclusiva* del bloque con la intención de modificarlo. El sistema de memoria (posiblemente otra caché) suministra los datos. Todas las otras cachés necesitan invalidar su posible copia. Esta transacción se genera por un PrWr a un bloque que no está en su caché o está en la caché pero no en el estado modificado. Una vez que la caché obtiene la copia exclusiva, la escritura (PrWr) puede realizarse en la caché. El procesador podría requerir un asentimiento (reconocimiento) como resultado de esta transacción.
- Post-escritura (BusWB): El controlador de caché pone la dirección y el contenido para un bloque de memoria en el bus. La memoria principal se actualiza con el último contenido. Esta transacción se genera por el controlador de caché en una post-escritura; el procesador no tiene conomiento de la misma, y no espera una respuesta.

La lectura exclusiva del bus (BusRdX, a veces denominada *lectura-para-posesión*) es una nueva transacción que podría no existir excepto para asegurar la coherencia de caché. Generada como un resultado de una escritura de un procesador, señala a las cachés que tomen una acción diferente a la de una lectura convencional. Además de cambiar el estado de los bloques en sus cachés, el controlador de caché puede intervenir en la transacción del bus y "vaciar" (poner) el contenido del bloque referenciado en el bus, en lugar de permitir a la memoria suministrar los datos. Este es el segundo concepto nuevo que se requiere para soportar los protocolos de post-escritura. El controlador de caché también puede iniciar nuevas transacciones en el bus, y suministrar datos para post-escrituras, o recoger los datos suministrados por el sistema de memoria.

El diagrama de estados para este protocolo de espionaje es el mostrado en la **Figura 9**.

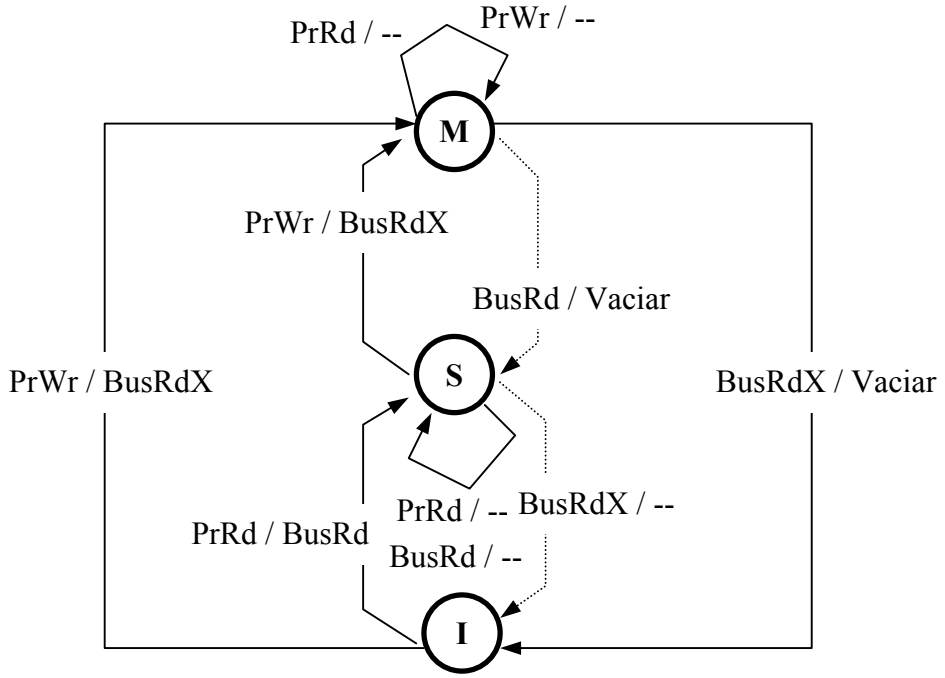


Figura 9. Protocolo de invalidación básico de 3 estados (MSI).

Los estados se han puesto de manera que cuanto más cercanos estén a la parte superior indican que el bloque está asociado más estrechamente con el procesador en cuestión. La notación A/B significa que si observamos proveniente del procesador o el bus un evento A, entonces además del cambio de estado, generaremos la transacción en el bus o acción B. "--" indica la acción nula. Si un arco tiene asociado múltiples pares A/B, simplemente indica que múltiples entradas pueden causar la misma transición de estados. Una lectura del procesador (PrRd) a un bloque no residente en caché ("no válido") causa una transacción BusRd que sirve ese bloque. El bloque que se acaba de ubicar en caché pasa del estado no válido (I) al estado compartido (S), exista o no cualquier otra caché que mantenga una copia del mismo bloque. Cualquier otra caché con el bloque en el estado compartido (S) observa el BusRd, pero no realiza ninguna acción especial, permitiendo que la memoria responda con los datos. Sin embargo, una caché con el bloque en el estado modificado (sólo puede haber una) debe involucrarse en la transacción, ya que la copia de memoria está obsoleta. Esa caché "vaciará" (pondrá) los datos del bloque en el bus, en lugar de hacerlo la memoria, y pasará su bloque al estado compartido (S). Tanto la memoria como la caché solicitante necesitan recoger el bloque. Esto puede realizarse bien con una transferencia caché-a-caché, o señalando un error en la transacción de lectura en el bus y generando una transacción de escritura para actualizar la memoria. En el último caso, la caché original repetirá finalmente su petición y obtendrá el bloque desde la memoria.

La escritura (PrWr) en un bloque no válido (I) es un fallo de escritura, que se sirve primero cargando el bloque entero en caché y luego modificando los bytes deseados dentro del mismo. Los fallos de escritura generan una transacción de lectura exclusiva en el bus (BusRdX), que causa que todas las otras copias del bloque en caché sean invalidadas, por tanto otorgando a la caché solicitante la propiedad exclusiva del bloque. El bloque pasará al estado de modificado (M) y luego será escrito. Si cualquier otra caché solicita posteriormente un acceso exclusivo, el bloque pasará al estado no válido (I), después de vaciar (poner) la copia exclusiva en el bus.

La transacción más interesante tiene lugar cuando se escribe (PrWr) sobre un bloque compartido (S). Se trata de una escritura con éxito (en caché) y tradicionalmente podría servirse completamente por la caché en un uniprocesador. Sin embargo, para mantener la coherencia se debe generar una transacción en el bus, que haga la escritura visible a los otros procesadores. Por eso se trata esencialmente como un fallo de escritura, usando una transacción de lectura exclusiva (BusRdX) para adquirir la posesión del bloque. En este caso, los datos que se obtienen en la lectura exclusiva generalmente pueden ignorarse, ya que todavía están en caché. Las escrituras (PrWr) posteriores al bloque mientras esté en el estado modificado (M) no generarán transacciones en el bus.

Obsérvese que para especificar completamente el protocolo, cada estado en el diagrama debe tener arcos de salida con las etiquetas correspondientes a todos los eventos observables (las entradas desde la caché y el bus), y también las acciones asociadas a éstos. Algunas veces las acciones pueden ser nulas, y en ese caso, podríamos especificar explícitamente las acciones nulas (véase los estados S y M en la figura), o podríamos simplemente omitir esos arcos en el diagrama (véase el estado I en la figura). Además, en un fallo de caché, cuando se trae el bloque de memoria nuevo, las transiciones de estado se hacen como si el estado previo del bloque fuera el estado no válido (I). Un reemplazamiento, lógicamente, pasa un bloque de caché al estado no válido al quitarlo de la caché. Los reemplazamientos no aparecen en la figura, porque se deben a fallos de caché sobre un bloque diferente que cae dentro del mismo conjunto de caché. La transición de reemplazamiento desde M a I genera una transacción de post-escritura. En esta transacción las otras cachés no tienen que realizar ningún tipo de

acción especial.

Para ilustrar algunas de las elecciones implícitas de diseño que se han realizado en el protocolo, vamos a examinar más a fondo la transición desde el estado M cuando se observa un BusRd para ese bloque. En la Figura 9 la transición nos lleva al estado S y el contenido del bloque de memoria se coloca en el bus. Mientras que es imprescindible que el contenido se coloque en el bus, podríamos en lugar de ir al estado S ir al I. La elección de ir al estado S frente al I refleja la aseveración del diseñador de que es más probable que el procesador original continúe leyendo ese bloque a que el nuevo procesador escriba dentro de poco en ese bloque de memoria. Intuitivamente, esta aseveración "apuesta" por los datos mayoritariamente leídos, que es lo común en muchos programas. Sin embargo, un caso común donde esto no se cumple es con una bandera o bufer que se utilice para transferir información de acá para allá entre dos procesos; un procesador escribe en él, el otro lo lee y modifica, entonces el primero lo lee y modifica, y así sucesivamente. El problema de apostar por las lecturas compartidas, en este caso, es que cada escritura se precede por una invalidación, por lo que se incrementa la latencia de la operación de tipo ping-pong. De hecho, el protocolo de coherencia usado en el primer multiprocesador **Synapse** hace la elección alternativa de ir directamente del estado M al I en un BusRd. Algunas máquinas (la Symmetry de Sequent (modelo B) y la Alewife del MIT) intentan adaptar el protocolo cuando el patrón de accesos indica datos migratorios de esta forma.

6.2. El protocolo de 4 estados MESI.

El protocolo MESI es un protocolo de invalidación con post-escritura. Con el protocolo MSI surge una seria preocupación si consideramos una aplicación secuencial ejecutándose en un multiprocesador; en realidad, tal uso multiprogramado constituye la carga de trabajo más común en multiprocesadores de baja escala. Cuando se lee y modifica un dato, en el protocolo MSI se generan dos transacciones en el bus incluso aunque no haya nunca ninguna otra caché que comparta este dato. La primera es un BusRd que pone el bloque de memoria en el estado S, y la segunda es un BusRdX que pasa el bloque del estado S al M. Añadiendo un estado que indique que el bloque está "limpio (no modificado) y sin compartir" y pasándolo a este estado, podemos ahorrarnos la última transacción ya que el estado indica que ningún otro procesador lo tiene en caché. Este nuevo estado establece un nivel intermedio de enlace, entre el estado compartido (S) y el modificado (M). En muchos microprocesadores modernos se utilizan distintas variantes de este protocolo MESI, incluyendo el Pentium de Intel, el PowerPC 601, y el MIPS R4400 usado en los multiprocesadores Silicon Graphics Challenge. El protocolo fue publicado por primera vez por investigadores de la Universidad de Illinois en Urbana-Champaign, por eso a menudo se le denomina **protocolo de Illinois**. Este protocolo puede considerarse una mejora del **protocolo Write Once**, al igual que el **protocolo de Berkeley**. El protocolo de Illinois mejora la gestión de los datos privados (en una sola caché) realizada por el protocolo Write Once gracias a su estado E. Por su parte, el protocolo de Berkeley fue diseñado específicamente para la estación de trabajo multiprocesador SPUR ("Symbolic Processing Using RISC" -procesamiento simbólico usando RISC) de la Universidad de California en Berkeley, y tiene en cuenta la discrepancia entre los tiempos de acceso de la memoria y las cachés para optimizar las transferencias caché-a-caché.

El protocolo MESI consiste en cuatro estados: modificado (M) o sucio, exclusivo (E), compartido (S, del inglés ***S**hared*), y no válido (I, de inválido). Los estados I y M tienen el mismo significado que en el protocolo MSI. El estado E, el estado "exclusivo" o "limpio y sin compartir", significa que sólo una caché (esta caché) tiene una copia del bloque, y éste no ha sido modificado (es decir, la memoria principal está actualizada). El estado S significa que

potencialmente dos o más procesadores tienen este bloque en su caché en un estado no modificado.

Cuando el bloque se lee por primera vez por un procesador, si existe una copia válida en otra caché entonces entra en la caché en el estado S como de costumbre. Sin embargo, si ninguna otra caché tiene una copia a la vez (por ejemplo, en una aplicación secuencial), entra en la caché en el estado E. Cuando ese bloque es escrito por el mismo procesador, podemos pasar directamente al estado M sin generar otra transacción en el bus, porque sabemos que ninguna otra caché tiene una copia. Si otra caché ha obtenido una copia mientras tanto, el estado del bloque debería haberse degradado del estado E al S por el protocolo de espionaje.

Este protocolo coloca un nuevo requisito en la interconexión física del bus. Debe existir una señal adicional, la señal *compartido* (S), disponible para los controladores de caché para poder determinar en un fallo de lectura (en caché) si cualquier otra caché mantiene los datos. Durante la fase de dirección de la transacción en el bus, todas las cachés determinan si contienen el bloque solicitado y, si es así, emiten la señal compartido. Esta es una línea de cableado-OR, por lo que el controlador que hace la petición puede observar si existe cualquier otro procesador que mantenga en caché el bloque de memoria referenciado, y por tanto decidir si cargar el bloque solicitado en el estado E o en el estado S.

La **Figura 10** muestra el diagrama de transición de estados para el protocolo MESI.

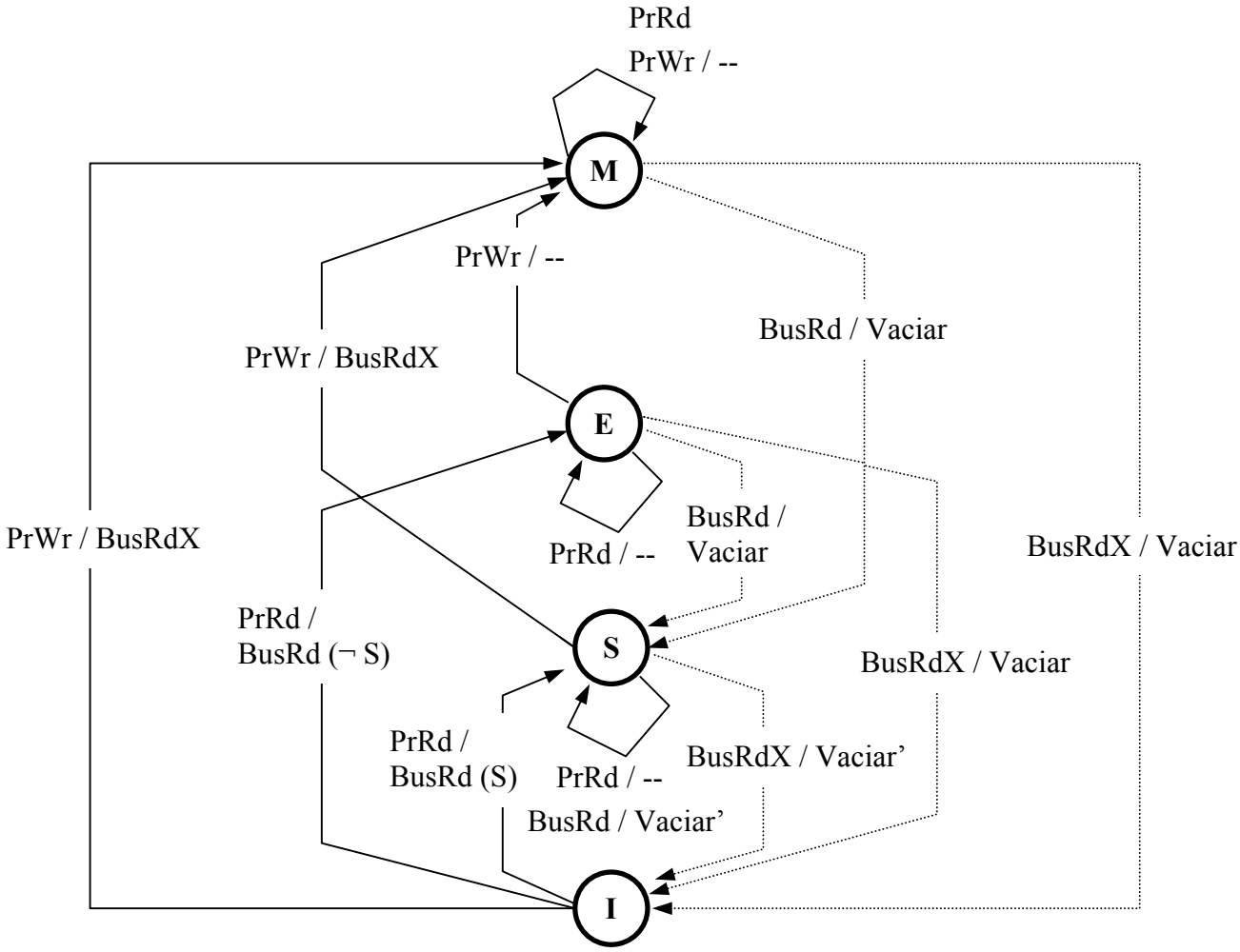


Figura 10. Diagrama de transición de estados para el protocolo MESI de Illinois.

La notación es la misma que la seguida para el protocolo MSI. BusRd(S) significa que cuando ocurrió la transacción de lectura se confirmó la señal "S", y BusRd(¬ S) significa que la señal "S" no se confirmó. Un BusRd simple significa que nos da igual el valor de la señal S para esa transición. Una escritura sobre un bloque en cualquier estado elevará el bloque al estado M, pero si estaba en el estado E no se requerirá ninguna transacción en el bus. Observando un BusRd el bloque pasará del estado E al S, ya que existe otra copia en alguna caché. Nótese que es posible para un bloque estar en el estado S incluso si no existe ninguna otra copia, pues las copias podrían ser descartadas (S → I) sin notificarlo a otras cachés.

Una pregunta interesante en este protocolo es quién debería suministrar el bloque para una transacción BusRd cuando tanto la memoria como otra caché tienen una copia del mismo. En la versión original de Illinois del protocolo MESI la caché, y no la memoria principal, suministraba los datos. El motivo para esta aproximación era que las cachés se construían con SRAM, y no DRAM, con lo que podían suministrar los datos más rápidamente. Sin embargo, esta ventaja era en gran parte mitigada con la complejidad añadida al protocolo - la memoria debe esperar hasta estar segura de que ninguna caché suministrará los datos antes de tomar el bus. Si los datos residen en varias cachés, se necesita que exista un algoritmo de selección que determine cuál proporcionará los datos. Este es el motivo de la notación Vaciar' en el diagrama, que indica que el vaciar (poner) el contenido del bloque en el bus sólo se cumple para ese procesador, los procesadores restantes no realizarán nada. Por otro lado, esta aproximación es útil en multiprocesadores con memoria distribuida físicamente, porque la latencia para obtener los datos desde una caché cercana podría ser mucho más pequeña que la de una unidad de memoria lejana. Este efecto puede ser especialmente importante para máquinas construidas como una red de nodos SMP, porque las cachés dentro del nodo SMP podrían suministrar los datos. El multiprocesador DASH de Stanford utilizaba estas transferencias caché-a-caché por este motivo.

6.3. El protocolo de 4 estados Dragon.

Se trata de un protocolo básico basado en actualización (no invalidación) para cachés con post-escritura, una versión mejorada del que se utiliza en los multiprocesadores SparcServer de SUN. Este protocolo fue propuesto inicialmente por los investigadores de Xerox PARC para su sistema multiprocesador Dragon.

El protocolo Dragon consiste en cuatros estados: *exclusivo* (E), *compartido-no-modificado* (SC, del inglés ***S**hared-**C**lean*), *compartido-modificado* (SM, del inglés ***S**hared-**M**odified*), y *modificado* (M). El estado exclusivo significa que sólo una caché (esta caché) tiene una copia del bloque, y éste no ha sido modificado (es decir, la memoria principal está actualizada). El motivo para añadir el estado E en el protocolo Dragon es el mismo que para el protocolo de Illinois o MESI. El estado SC significa que potencialmente dos o más procesadores (incluyendo esta caché) tienen este bloque en su caché; la memoria principal podría o no estar actualizada. El estado SM significa que potencialmente dos o más procesadores tienen este bloque en su caché, la memoria principal no está actualizada, y es responsabilidad de este procesador el actualizar la memoria principal a la vez que este bloque se reemplace en la caché. El estado M significa, como en los otros protocolos, que el bloque se modifica en esta caché sólo, la memoria principal está obsoleta, y es responsabilidad de este procesador actualizar la memoria principal durante el reemplazamiento del bloque. Obsérvese que no existe un estado explícito no válido (I) como en los protocolos MSI y MESI. Esto es porque Dragon es un protocolo basado en actualización (no invalidación); el protocolo siempre mantiene los bloques en caché actualizados, por lo que siempre es correcto utilizar los datos presentes en caché (lógicamente existe un "quinto estado", pero es bastante

rudimentario; Se proporciona un bit de modo para forzar los fallos (de caché), y el software de inicialización lee los datos para cada bloque de la caché en el modo de **fallo** para conseguir llenarlos).

Las peticiones del procesador, las transacciones en el bus, y las acciones para el protocolo Dragon son similares a las del protocolo MESI de Illinois. Todavía se asume que el procesador sólo emite las peticiones de lectura (PrRd) y escritura (PrWr). Sin embargo, dado que no tenemos un estado no válido en el protocolo, para especificar las acciones cuando se solicita por primera vez un bloque nuevo de memoria por el procesador, añadimos dos tipos más de petición: fallo de lectura (PrRdMiss) y fallo de escritura (PrWrMiss) del procesador. En cuanto a las transacciones en el bus, tenemos la lectura del bus (BusRd), la actualización del bus (BusUpd), y la post-escritura del bus (BusWB). Las transacciones BusRd y BusWB tienen el significado normal que se definió para los protocolos MSI y MESI. BusUpd es una transacción nueva que toma la palabra específica escrita por el procesador y la difunde en el bus para que todas las otras cachés puedan actualizarse. Difundiendo sólo el contenido de la palabra modificada en lugar del bloque de caché entero, se espera que el ancho de banda del bus se utilice más eficientemente. Como en el protocolo MESI, para soportar los estados E y M, tenemos disponible la señal compartido (S) para el controlador de caché. Esta señal se emite si existe cualquier procesador, a parte del solicitante, que mantiene en caché el bloque de memoria referenciado. Finalmente, en cuanto a las acciones, la única capacidad nueva necesitada es para que el controlador de caché actualice un bloque de memoria en la caché local (etiquetado como "Actualizar") con el contenido que se está difundiendo en el bus por una transacción BusUpd relevante.

La **Figura 11** muestra el diagrama de transición de estados para el protocolo de actualización Dragon.

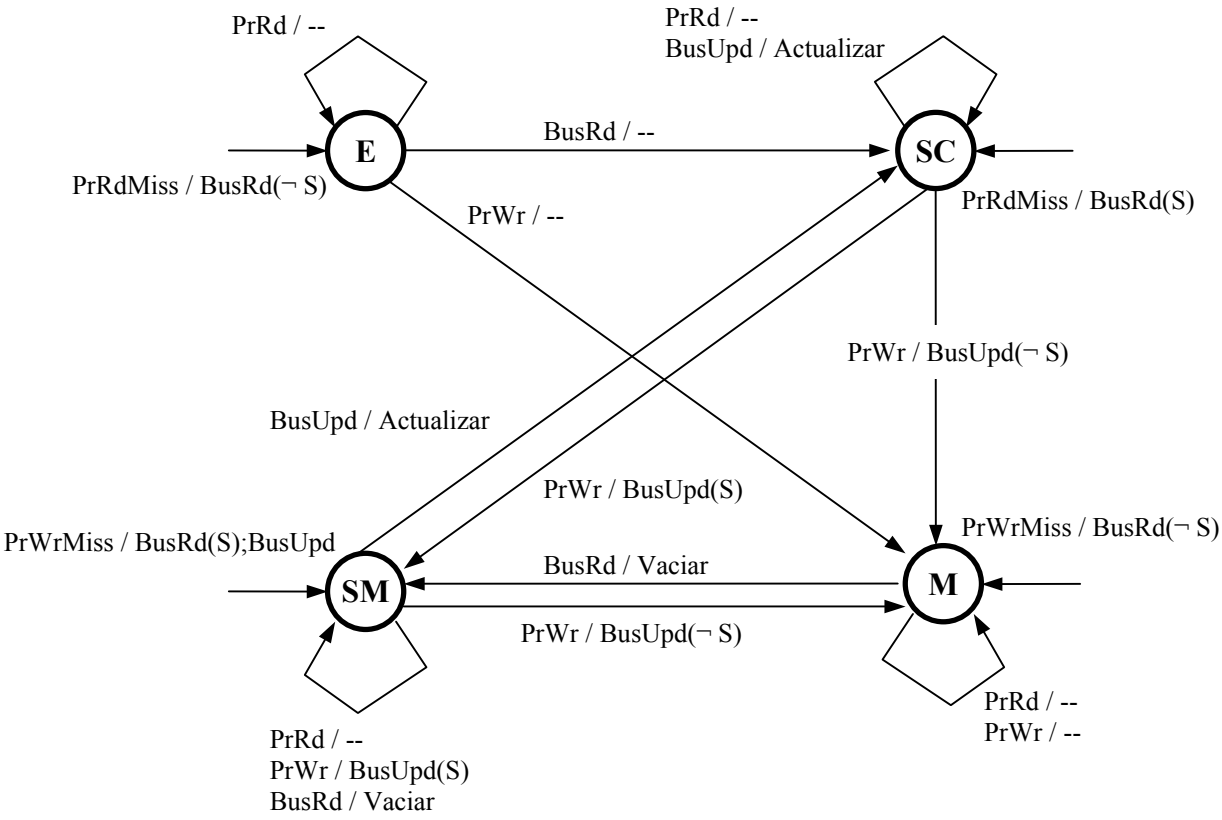


Figura 11. Diagrama de transición de estados para el protocolo de actualización Dragon.

Para tomar un punto de vista centrado en el procesador, también se puede explicar el diagrama en términos de las acciones que se realizan cuando un procesador provoca un fallo de lectura, un acierto de escritura, o un fallo de escritura (en un acierto de lectura no se realiza ninguna acción).

- **Fallo de lectura**: Se genera una transacción BusRd. Dependiendo del estado de la señal compartido (S), el bloque pasa al estado E o SC en la caché local. Más concretamente, si el bloque de memoria está en los estados M o SM en una de las otras cachés, esa caché afirma la señal compartido y suministra los datos más recientes de ese bloque en el bus, y el bloque se carga en la caché local en el estado SC. Si la otra caché tiene el bloque en el estado M, cambia su estado a SM. Si ninguna otra caché tiene una copia del bloque, entonces la línea de la señal compartido permanece sin afirmar (sin ponerse a 1), los datos se suministran por la memoria principal y el bloque se carga en la caché local en el estado E.
- **Acierto de escritura**: Si el bloque está en el estado M en la caché local, entonces no se necesita realizar ninguna acción. Si el bloque está en el estado E en la caché local, entonces internamente cambia al estado M y de nuevo no se necesita realizar ninguna acción adicional. Sin embargo, si el bloque está en los estados SC y SM, se genera una transacción BusUpd. Si cualquiera de las otras cachés tiene una copia de los datos, actualizarán sus copias en caché, y cambiarán su estado a SC. La caché local también actualiza su copia del bloque y cambia su estado a SM. Si ninguna otra caché tiene una copia de los datos, la señal compartido permanece sin afirmar, la copia local se actualiza y el estado se cambia a M.
- **Fallo de escritura**: Un fallo de escritura se trata simplemente como una transacción de fallo de lectura seguida por una transacción de acierto de escritura. Por tanto, primero se genera un BusRd, y luego si el bloque también se encontraba en otras cachés (es decir, el bloque se habría cargado localmente en el estado SC), se genera un BusUpd. Si no se encontraba en ninguna otra caché el bloque pasará al estado M y sino al SM.

De nuevo, se han realizado muchas elecciones de diseño implícitas dentro de este protocolo. Por ejemplo, es factible eliminar el estado compartido-modificado (SM). De hecho, el protocolo de actualización utilizado en el multiprocesador **Firefly** de DEC hizo precisamente eso. El razonamiento era que cada vez que ocurre la transacción BusUpd, la memoria principal también puede actualizar su contenido junto con las otras cachés que mantienen ese bloque, y por tanto, no se necesita el estado compartido-modificado. El protocolo Dragon en su lugar se basó en la suposición de que como las cachés utilizan SRAM y la memoria principal usa DRAM, es mucho más rápido actualizar las cachés que la memoria principal, y por tanto no es apropiado esperar hasta que la memoria principal se actualice en cada transacción BusUpd. Otra elección delicada, por ejemplo, es la relacionada con la acción a realizar en los reemplazamientos de caché. Cuando se reemplaza un bloque en el estado SC, se debería informar a las otras cachés de ese reemplazamiento, para que si sólo existe una caché con una copia del bloque de memoria, pueda cambiar su estado al de exclusivo (E) o modificado (M).

7. Arbitración del bus.

Al trabajar con sistemas multiprocesador con bus compartido es posible que varios de los controladores de caché quieran acceder a la vez al bus. Para establecer cuál de ellos lo tomará es necesario utilizar un algoritmo de arbitración del bus.

Dentro de este simulador la arbitración del bus se podrá realizar mediante los algoritmos de arbitración: aleatorio, LRU y LFU. Los conceptos teóricos de estos algoritmos son similares a los que se explican para los algoritmos de reemplazamiento.

8. Ficheros de configuración.

Para almacenar la configuración del simulador SMPCache se utilizan ficheros con extensión **.CFG**. Estos ficheros serán ficheros de texto con el siguiente formato:

```
Nº procesadores:
3
Protocolo coherencia caché:
2
Arbitración bus:
1
Ancho palabra (bits):
64
Palabras en un bloque:
128
Bloques en memoria:
1024
Bloques en caché:
64
Función correspondencia:
3
Nº conjuntos:
0
Algoritmo reemplazamiento:
1
Niveles de caché:
1
Política de escritura:
2
```

Como vemos, en primer lugar se escribe un literal indicando lo que se está configurando (nº procesadores, protocolo de coherencia, etc.) y en la siguiente línea se establece la configuración. Todos los ficheros .CFG tendrán el mismo formato, de manera que los "comentarios" aparecen en las líneas pares, y las configuraciones en sí en las impares.

En el caso de configuraciones numéricas como el nº de procesadores, el ancho de la palabra, las palabras en un bloque, etc.; Lo que se escribe en el fichero es la propia configuración numérica. Para el resto de configuraciones se establece una codificación numérica para cada una de las posibles opciones. La

Tabla 2 muestra las opciones de configuración y su codificación numérica asociada.

	Valores posibles	Codificación
Protocolo de coherencia caché	MSI MESI DRAGON	1 2 3
Arbitración del bus	Aleatoria LRU LFU	1 2 3
Función de correspondencia	Directa Asociativa por conjuntos Totalmente asociativa	1 2 3
Número de conjuntos	NO 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 ó 2048	0 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 ó 2048
Algoritmo de reemplazamiento	NO Aleatorio LRU FIFO LFU	0 1 2 3 4
Política de escritura	Escritura directa Post-escritura	1 2

Tabla 2: Opciones de configuración no numéricas y su codificación numérica asociada.

Aunque el número de niveles de caché y la política de escritura no son configurables, también aparecerán en los ficheros .CFG. Esto facilitaría posibles ampliaciones futuras.

9. Ficheros con trazas de memoria.

Para usar el simulador, es necesario contar con unos ficheros de datos que contengan el tipo de acceso y dirección de memoria demandada por la CPU durante la ejecución de un programa (traza de memoria). Estos ficheros de datos van a tener un determinado formato para que el simulador pueda tratarlos, y van a permitir emular los programas a procesar por los distintos procesadores que pueden existir en nuestro SMP.

Los ficheros con trazas de memoria tendrán extensión .PRG, y van a ser ficheros de texto. Constarán de un conjunto de líneas, cada una de las cuales contiene dos números, separados por un único espacio en blanco:

etiqueta valor

Donde:

- **Etiqueta** es un número decimal que identifica el tipo de operación de acceso a memoria demandada por la CPU en un momento dado: capturar una instrucción (0), leer un dato de memoria (2) o escribir un dato en memoria (3).

- **Valor** es un número hexadecimal, que indica la dirección efectiva de la palabra de memoria que se va a acceder. Esta dirección será traducida por el simulador para localizar la palabra dentro de la estructura de bloques del sistema de memoria.

En la **Figura 12** observamos un ejemplo de lo que representa una traza de memoria tal como la utilizará el simulador:

Por ejemplo, esta porción de una traza de memoria muestra 6 capturas de instrucciones de un determinado programa. Tres instrucciones implican lectura, y una requiere escribir en memoria. En total, esas 6 instrucciones implican 10 accesos a memoria.

```
0 00001c07
0 00001da4
2 00007a50
0 00001e03
0 00001fb7
2 00007a51
0 0000201b
2 00007d70
0 0000211e
3 00007c50
```

Figura 12: Ejemplo de traza de memoria.

Como sabemos, la memoria principal almacena conjuntos de palabras dentro de un mismo bloque. La CPU invoca una operación de lectura/escritura sobre una determinada palabra, identificando el bloque en el que se encuentra y el desplazamiento que hay que realizar para obtenerla dentro de ese bloque. Por tanto, las direcciones de memoria anteriores van a tener el siguiente formato interno:

Dirección de bloque	**Desplazamiento palabra dentro bloque**

El motivo por el que se agrupan las palabras en bloques es para aprovechar el principio de localidad espacial. Los bloques son la unidad de información que van a intercambiarse la memoria principal y la memoria caché (o las cachés entre sí).