## ALGORITMOS DE SUSTITUCIÓN

Una vez que se ha llenado la caché, para introducir un nuevo bloque debe sustituirse uno de los bloques existentes. Para el caso de correspondencia directa, solo hay una posible línea para cada bloque particular y no hay elección posible. Para las técnicas asociativas se requieren algoritmos de sustitución. Para conseguir alta velocidad, tales algoritmos deben implementarse en hardware. Se han probado diversos algoritmos; mencionaremos cuatro de los más comunes. El más efectivo es probablemente el denominado "utilizado menos recientemente" (LRU, *least-recently used*): se sustituye el bloque que se ha mantenido en la caché por más tiempo sin haber sido referenciado. Esto es fácil de implementar para la asociativa por conjuntos de dos vías. Cada línea incluye un bit USO. Cuando una línea es referenciada se pone a 1 su bit USO y a 0 el de la otra línea del mismo conjunto. Cuando va a transferirse un bloque al conjunto, se utiliza la línea cuyo bit USO es 0. Ya que estamos suponiendo que son más probables de referenciar las posiciones de memoria utilizadas más recientemente, el LRU debiera dar la mejor tasa de aciertos. Otra posibilidad es el *primero en entrar-primero en salir* (FIFO, *First-In-First-Out*): se sustituye aquel bloque del conjunto que ha estado más tiempo en la caché. El algoritmo FIFO puede implementarse fácilmente mediante una técnica cíclica (*round-robin*) o buffer circular. Otra posibilidad más es la del *utilizado menos frecuentemente* (LFU, *Least-Frequently Used*): se sustituye aquel bloque del conjunto que ha experimentado menos referencias. LFU podría implementarse asociando un contador a cada línea. Una técnica no basada en el grado de utilización consiste simplemente en coger una línea al azar (aleatoria) entre las posibles candidatas. Estudios realizados mediante simulación han mostrado que la sustitución aleatoria proporciona unas prestaciones solo ligeramente inferiores a un algoritmo basado en la utilización [SMIT82].

## POLÍTICA DE ESCRITURA

Hay dos casos a considerar cuando se ha de reemplazar un bloque de la caché. Si el bloque antiguo de la caché no debe ser modificado, puede sobrescribirse con el nuevo bloque sin necesidad de actualizar el antiguo. Si se ha realizado al menos una operación de escritura sobre una palabra de la línea correspondiente de la caché, entonces la memoria principal debe actualizarse, rescribiendo la línea de caché en el bloque de memoria antes de transferir el nuevo bloque. Son posibles varias políticas de escritura con distintos compromisos entre prestaciones y coste económico. Hay dos problemas contra los que luchar. En primer lugar, más de un dispositivo puede tener acceso a la memoria principal. Por ejemplo, un módulo de E/S puede escribir/leer directamente en/de memoria. Si una palabra ha sido modificada solo en la caché, la correspondiente palabra de memoria no es válida. Además, si el dispositivo de E/S ha alterado la memoria principal, entonces la palabra de caché no es válida. Un problema más complejo ocurre cuando varios procesadores se conectan al mismo bus y cada uno de ellos tiene su propia caché local. En tal caso, si se modifica una palabra en una de las cachés, podría presumiblemente invalidar una palabra de otras cachés.

La técnica más sencilla se denomina **escritura inmediata**. Utilizando esta técnica, todas las operaciones de escritura se hacen tanto en caché como en memoria principal, asegurando que el contenido de la memoria principal siempre es válido. Cualquier otro módulo procesador-caché puede monitorizar el tráfico a memoria principal para mantener la coherencia en su propia caché. La principal desventaja de esta técnica es que genera un tráfico sustancial con la memoria que puede originar un cuello de botella. Una técnica alternativa, conocida como **postescritura**, minimiza las escrituras

en memoria. Con la postescritura, las actualizaciones se hacen solo en la caché. Cuando tiene lugar una actualización, se activa un bit ACTUALIZAR asociado a la línea. Después, cuando el bloque es sustituido, es (post) escrito en memoria principal si y solo si el bit ACTUALIZAR está activo. El problema de este esquema es que se tienen porciones de memoria principal que no son válidas, y los accesos por parte de los módulos de E/S tendrán que hacerse solo a través de la caché. Esto complica la circuitería y genera un cuello de botella potencial. La experiencia ha demostrado que el porcentaje de referencias a memoria para escritura es del orden del 15 por ciento [SMIT82]. Sin embargo, en aplicaciones de HPC, este porcentaje puede aproximarse al 33 por ciento (multiplicación de vectores) e incluso superar el 50 por ciento (en transposición de matrices).

**Ejemplo 4.3.** Considere una caché con un tamaño de línea de 32 bytes y una memoria principal que requiere 30 ns para transferir una palabra de 4 bytes. Para cualquier línea de caché que sea escrita al menos una vez antes de ser intercambiada, ¿qué número medio de veces debe haber sido escrita la línea antes del intercambio para que la postescritura resulte más eficiente que la escritura inmediata?

Para el caso de postescritura, cada línea modificada debe escribirse en memoria sólo una vez, al intercambiarse, invirtiendo $8 \times 30 = 240$ ns. En el caso de escritura inmediata, cada actualización de la línea exige escribir una palabra en memoria principal, tardando 30 ns. Por tanto, si en promedio, en las líneas en las que se escribe se realizan más de ocho escrituras antes de ser intercambiadas, la postescritura resulta más eficiente.

En una estructura de bus en la que más de un dispositivo (normalmente un procesador) tiene una caché y la memoria principal es compartida, se tropieza con un nuevo problema. Si se modifican los datos de una caché, se invalida no solamente la palabra correspondiente de memoria principal, sino también la misma palabra en otras cachés (si coincide que otras cachés tengan la misma palabra). Incluso si se utiliza una política de escritura inmediata, las otras cachés pueden contener datos no válidos. Un sistema que evite este problema se dice que mantiene la coherencia de caché. Entre las posibles aproximaciones a la coherencia de caché se incluyen:

- **Vigilancia del bus con escritura inmediata:** cada controlador de caché monitoriza las líneas de direcciones para detectar operaciones de escritura en memoria por parte de otros maestros del bus. Si otro maestro escribe en una posición de memoria compartida que también reside en la memoria caché, el controlador de caché invalida el elemento de la caché. Esta estrategia depende del uso de una política de escritura inmediata por parte de todos los controladores de caché.
- **Transparencia hardware:** se utiliza hardware adicional para asegurar que todas las actualizaciones de memoria principal, vía caché, quedan reflejadas en todas las cachés. Así, si un procesador modifica una palabra de su caché, esta actualización se escribe en memoria principal. Además, de manera similar se actualizan todas las palabras coincidentes de otras cachés.
- **Memoria excluida de caché:** solo una porción de memoria principal se comparte por más de un procesador, y esta se diseña como no transferible a caché. En un sistema de este tipo, todos los accesos a la memoria compartida son fallos de caché, porque la memoria compartida nunca se copia en la caché. La memoria excluida de caché puede ser identificada utilizando lógica de selección de chip o los bits más significativos de la dirección.

La coherencia de caché es un campo activo de investigación y será tratado con más detalle en el Capítulo 18.

## TAMAÑO DE LÍNEA

Otro elemento de diseño es el tamaño de línea. Cuando se recupera y ubica en caché un bloque de datos, se recuperan no sólo la palabra deseada sino además algunas palabras adyacentes. A medida que aumenta el tamaño de bloque, la tasa de aciertos primero aumenta debido al principio de localidad, el cual establece que es probable que los datos en la vecindad de una palabra referenciada sean referenciados en un futuro próximo. Al aumentar el tamaño de bloque, más datos útiles son llevados a la caché. Sin embargo, la tasa de aciertos comenzará a decrecer cuando el tamaño de bloque se haga aún mayor y la probabilidad de utilizar la nueva información captada se haga menor que la de reutilizar la información que tiene que reemplazarse. Dos efectos concretos entran en juego:

- Bloques más grandes reducen el número de bloques que caben en la caché. Dado que cada bloque captado se escribe sobre contenidos anteriores de la caché, un número reducido de bloques da lugar a que se sobrescriban datos poco después de haber sido captados.
- A medida que un bloque se hace más grande, cada palabra adicional está más lejos de la requerida y por tanto es más improbable que sea necesaria a corto plazo.

La relación entre tamaño de bloque y tasa de aciertos es compleja, dependiendo de las características de localidad de cada programa particular, no habiéndose encontrado un valor óptimo definitivo. Un tamaño entre 8 y 64 bytes parece estar razonablemente próximo al óptimo [SMIT87, PRZY88, PRZY90, HAND98]. Para sistemas de HPC es más frecuente usar tamaños de línea de caché de 64 y 128 bytes.

## NÚMERO DE CACHÉS

Cuando se introdujeron originalmente las cachés, un sistema tenía normalmente solo una caché. Más recientemente, se ha convertido en una norma el uso de múltiples cachés. Hay dos aspectos de diseño relacionados con este tema que son el número de niveles de caché, y el uso de caché unificada frente al de cachés separadas.

**Cachés multinivel.** Con el aumento de densidad de integración, ha sido posible tener una caché en el mismo chip del procesador: caché *on-chip*. Comparada con la accesible a través de un bus externo, la caché *on-chip* reduce la actividad del bus externo del procesador y por tanto reduce los tiempos de ejecución e incrementa las prestaciones globales del sistema. Cuando la instrucción o el dato requeridos se encuentran en la caché *on-chip,* se elimina el acceso al bus. Debido a que los caminos de datos internos al procesador son muy cortos en comparación con la longitud de los buses, los accesos a la caché *on-chip* se efectúan apreciablemente más rápidos que los ciclos de bus, incluso en ausencia de estados de espera. Además, durante este periodo el bus está libre para realizar otras transferencias.

La inclusión de una caché *on-chip* deja abierta la cuestión de si es además deseable una caché externa u *off-chip.* Normalmente la respuesta es afirmativa, y los diseños más actuales incluyen tanto caché *on-chip* como externa. La estructura más sencilla de este tipo se denomina caché de dos niveles, siendo la caché interna el nivel 1 (L1), y la externa el nivel 2 (L2). La razón por la que se inclu-

ye una caché L2 es la siguiente. Si no hay caché L2 y el procesador hace una petición de acceso a una posición de memoria que no está en la caché L1, entonces el procesador debe acceder a la DRAM o la ROM a través del bus. Debido a la lentitud usual del bus y a los tiempos de acceso de las memorias, se obtienen bajas prestaciones. Por otra parte, si se utiliza una caché L2 SRAM (RAM estática), entonces con frecuencia la información que falta puede recuperarse fácilmente. Si la SRAM es suficientemente rápida para adecuarse a la velocidad del bus, los datos pueden accederse con cero estados de espera, el tipo más rápido de transferencia de bus.

En la actualidad son destacables dos características de diseño de las cachés multinivel. En primer lugar, para el caso de una caché L2 externa, muchos diseños no usan el bus del sistema como camino para las transferencias entre la caché L2 y el procesador, sino que se emplea un camino de datos aparte para reducir el tráfico en el bus del sistema. En segundo lugar, gracias a la continua reducción de dimensiones de los componentes de los procesadores, es fácil encontrar procesadores que incorporan la caché L2 en el propio chip, con la consiguiente mejora de prestaciones.

La mejora potencial del uso de una caché L2 depende de las tasas de aciertos en ambas cachés L1 y L2. Varios estudios han demostrado que, en general, el uso de un segundo nivel de caché mejora las prestaciones (*véase* por ejemplo [AZIM92], [NOVI93], [HAND98]). No obstante, el uso de cachés multinivel complica todos los aspectos de diseño de la caché, incluyendo el tamaño, el algoritmo de sustitución y la política de escritura; en [HAND98] y [PEIR99] se discuten estos temas.

Con la creciente disponibilidad de superficie para caché en el propio chip, en la mayoría de los microprocesadores modernos se ha llevado la caché L2 al chip del procesador, y se añade una caché L3. Inicialmente, la caché L3 era accesible a través del bus externo, pero más recientemente los microprocesadores han incorporado una L3 *on-chip*. En cualquiera de los casos, añadir un tercer nivel de caché parece suponer una mejora de las prestaciones (*véase* por ejemplo [GHAI98]).

**Caché unificada frente a cachés separadas.** Cuando hicieron su aparición las cachés *on-chip,* muchos de los diseños contenían una sola caché para almacenar las referencias tanto a datos como a instrucciones. Más recientemente, se ha hecho normal separar la caché en dos: una dedicada a instrucciones y otra a datos.

Una caché unificada tiene varias ventajas potenciales:

- Para un tamaño dado de caché, una unificada tiene una tasa de aciertos mayor que una caché partida, ya que nivela automáticamente la carga entre captación de instrucciones y de datos. Es decir, si un patrón de ejecución implica muchas más captaciones de instrucciones que de datos, la caché tenderá a llenarse con instrucciones, y si el patrón de ejecución involucra relativamente más captaciones de datos, ocurrirá lo contrario.
- Solo se necesita diseñar e implementar una caché.

A pesar de estas ventajas, la tendencia es hacia cachés separadas, particularmente para máquinas super-escalares tales como el Pentium y el PowerPC, en las que se enfatiza la ejecución paralela de instrucciones y la precaptación de instrucciones futuras previstas. La ventaja clave del diseño de una caché partida es que elimina la competición por la caché entre el procesador de instrucciones y la unidad de ejecución. Esto es importante en diseños que cuentan con segmentación de cauce (*pipelining*) de instrucciones. Normalmente el procesador captará instrucciones anticipadamente y llenará un buffer con las instrucciones que van a ejecutarse. Supongamos ahora que se tiene una caché unificada de instrucciones/datos. Cuando la unidad de ejecución realiza un acceso a memoria para cargar y almacenar datos,