



Mejora de Performance de la Cache

Hemos visto las tres Cs que definen los distintos miss: *Compulsory*, *Capacity* y *Conflict*.

Para reducir los miss de **capacidad** la alternativa, obviamente, emplear caches más grandes, lo cual podrá conspirar contra el tiempo de acceso, el *hit time*. Aumentar la *asociatividad* también conduce a reducción de los miss, en este caso de conflicto, pero también afectará el *hit time*. En un extremo tenemos el esquema *Full Associative* con 0 miss de conflicto pero costosa y lenta; en el otro tenemos *Mapeo Directo* simple y rápida, pero con máximo miss de conflicto. Mejores alternativas se alcanzan con esquemas *Set Associative* con un determinado grado de asociatividad. A manera de comparación, una cache con mapeo directo de un cierto tamaño tendrá un hit ratio aproximadamente igual a una cache de la mitad de tamaño, pero que sea 2-way set associative.

Para los **compulsivos**, de la primera vez, aumentando el tamaño del bloque se mejora como se ha visto pero sin dejar de lado que su aumento incrementará el *miss penalty*, y que por otro lado el aumento del tamaño no debería incrementar los miss de conflicto. Ambas alternativas, bloques más grandes y mayor asociatividad, fueron alternativas exploradas desde las primeras caches. Más recientemente surgieron alternativas que apuntan a reducir el miss rate sin afectar al período de reloj (o tiempo de acceso) o al miss penalty. Una de ellas es el empleo de *Victim Cache*, una pequeña memoria full asociativa que se intercala entre la cache y el bus al nivel inferior. Contiene sólo los bloques que han sido descartados de la cache por causa de un miss -victims- y serán chequeadas para ver si tienen el dato buscado, evitando el miss. De encontrarse allí, el *victim block* y el *cache block* son intercambiados entre sí.

Otra técnica para mejorar el miss rate sin afectar el clock rate es hacer prefetch de items antes que sean requeridos por el procesador. Instrucciones y datos pueden traerse en avance, bien sea directamente en la cache o en un buffer externo a ella, el que podrá ser accedido más rápidamente que memoria principal. Para las instrucciones típicamente se las almacena en un buffer externo, denominado *instruction stream buffer*, donde se recuperan dos bloques de memoria principal en cada miss. Existen organizaciones que automáticamente disparan el prefetch a partir de comportamientos que se den en la cache. El prefetching también podrá ser por software, con arquitecturas que implementen instrucciones al efecto. Estas a su vez podrán ser *faulting* o *no faulting* según continúe en caso de page fault, o simplemente se transformen en nops respectivamente. Para que esto funcione la cache debe poder seguir suministrando instrucciones y datos mientras se resuelve el prefetch (no bloqueante). El prefetch, tanto el de hard como el de soft, propende a superponer la ejecución con la búsqueda anticipada, de forma de encontrar el dato en cache al momento de requerirlo, evitando el miss.

Por otro lado, la penalización de los miss se podrá reducir de distintas maneras. Una de ellas es dar prioridad a los read miss sobre las escrituras. Disponer de un *write buffer*, que como se apuntan a mejorar la performance trabajando con *write through*, podría emplearse para satisfacer dicho requerimiento. Una segunda alternativa es trabajar con *sub-block*. Disponer de grandes bloques por un lado reduce el costo de almacenamiento de los tags y mejora el hit rate, pero el incremento en el miss penalty podría tornar al uso de bloques grandes inconveniente. Una solución al problema del almacenamiento de los tags es dividir el bloque en *sub-bloques*, a cada uno se le asocia un bit de válido, por lo cual ante un miss sólo el sub bloque deberá ser traído desde memoria. Un dado bloque, identificado por un tag, podrá tener sub bloques en estado de válido y otros inválido. A este esquema respondió la primer

memoria cache implementada en un sistema S/370 habida cuenta de la limitación que se tenía en el tamaño de la memoria rápida (pues empleaba tecnología bipolar). Otras alternativas plantean *Early Restart*: ni bien la palabra requerida arribe a cache debe ser enviada al **CPU** para continuar la ejecución, o bien *Critical Word First*: ir a buscar primero la palabra que originó el miss, y luego continuar completando el bloque. Cabe acotar que esta última no parece ser adecuada a las características organizativas de las memorias **DRAM** actuales, las que resultan altamente eficientes transmitiendo en bloques.

Por último se tienen dos alternativas más para reducir el miss penalty: *cache no bloqueantes*, o *lockup-free cache*, y organizaciones de *cache multinivel*. La primera plantea que ante un miss la cache continúe suministrando datos, esquema *hit under miss*, con lo cual se solapa la búsqueda del dato en el nivel inferior con cómputo, reduciendo la penalización efectiva del miss. Más aún, aunque más complejo, lo que se hace actualmente es *hit under multiple miss*, vale decir admitir varios miss en curso mientras se continua suministrando datos al **CPU**. Es de observar que esta última forma resulta vital en cualquier arquitectura contemporánea que cuente con ejecución fuera de orden, dado el ancho de las ventanas de instrucciones (próximos al centenar) y la frecuencia de los accesos a memoria vuelve esencial posibilitar su despacho sin causar un *stall* del pipeline.

La segunda, una organización multinivel con dos o tres niveles de cache, posibilita resolver adecuadamente el compromiso velocidad-capacidad. Hay una demanda de caches más grandes que se origina en el mayor tamaño de la memoria principal, pero a su vez cache grandes atentan contra el hit time. La solución es organizar a la cache jerárquicamente, esto es un primer nivel de tamaño reducido, que posibilita una tasa de acierto razonable, el cual condiciona directamente el temporizado del **CPU**; luego los niveles inferiores serán de mayor tamaño, tolerando mayor tiempo de acceso, con el resultado trabajar, en conjunto, a una velocidad próxima a la del primer nivel y contar con una capacidad adecuada. Tendremos miss locales, en cada nivel de cache, y un miss global que será el número de miss en cache como un todo, esto es accesos que deben ir a memoria dividido por el número total de referencias a memoria que produce el **CPU**:

$$T . \text{AccesoProm} . = \text{HitTime}_{L1} + \text{MissRate}_{L1} \times \text{MissPenalty}_{L1}$$

por lo cual quedará:

$$T . \text{AccesoProm} . = \text{HitTime}_{L1} + \text{MissRate}_{L1} \times \left(\text{HitTime}_{L2} + \text{MissRate}_{L2} \times \text{MissPenalty}_{L2} \right)$$

Finalmente, el Miss Rate Global estará dado por el producto $\text{MissRate}_{L1} \times \text{MissRate}_{L2}$.

Veamos ahora alternativas para reducir el hit time. El tema es particularmente crítico en el primer nivel de cache por lo que habrá que mantenerla relativamente simple y controlar su tamaño. Otro aporte lo constituirá el evitar el mecanismo de traslación en el indexado a cache. Vale decir en lugar de hacer que partiendo de la dirección virtual el **MMU** suministre la dirección física y recién allí se acceda a la cache, se puede acometer esta tarea de búsqueda usando directamente la dirección virtual. Se habla así de cache virtual en contraposición a cache física (tradicional). Este tipo de cache logra evitar el tiempo de traslación en el caso de producirse un hit en cache. A continuación analizaremos cuáles son las dificultades que plantea este esquema que a primera vista parece muy ventajoso.

Bajo un esquema de cache virtual se originan dos tipos de problemas, en primer lugar de ambigüedad, una misma dirección lógica para distintas locaciones de memoria, y en segundo lugar de sinónimos (o alias), distintas direcciones lógicas para una misma física. Cuando se cambia de proceso debería hacerse flush de toda la cache, asociado con las ambigüedades, lo cual se podría atenuar si se asociase a cada línea en cache una identificación del proceso, ASI

(*Address Space Identifier*), el cual no debe confundirse con el PID (*Process Identifier Tag*). El ASI es un pequeño entero, podría ser un byte que permitiría en tal caso la coexistencia de hasta 256 procesos distintos sin requerir el flush de la cache entre cambios de contexto. Los sinónimos se originan en que tanto el usuario como el sistema operativo podrán usar dos direcciones virtuales distintas para la misma dirección física, lo cual podrá resultar en dos copias de una misma locación en cache, si una es modificada la otra podrá tener un valor incorrecto. Con cache físicas esto no podría ocurrir dado que se traslada la dirección lógica a la física previo al acceso. La solución de hardware es asegurar que no podrá haber en cache dos bloques que mapeen a una misma dirección física, y se la conoce como *anti-aliasing*. El software puede resolver este problema de forma más simple forzando a los alias a compartir algunos bits de dirección. La versión de UNIX de Sun Microsystems, por ejemplo, requiere que todos los alias sean idénticos en los últimos 18 bits de dirección; esta restricción se llama *page coloring*. Esta restricción implica que una cache con mapeo directo por 256KB o menos nunca tendrá duplicada una dirección virtual para un dado block.

Otra cuestión que surge con las direcciones virtuales es la **I/O**. Normalmente la I/O emplea direcciones físicas, requiriendo luego un mapeo a dirección virtual para interactuar con la cache virtual. Una alternativa para las caches virtuales es trabajar con tags físicos. Esto por un lado elimina los problemas de ambigüedades, al requerir la dirección física para la comparación, pero requiere que en cada referencia a memoria trabaje el mecanismo de traslación, cosa que ocurre solo con un miss para el caso de la cache virtual. El problema de alias no desaparece por trabajar con tags físicos, dado que se la indexa virtualmente, debiendo por ende recurrirse o bien al hardware con el anti-aliasing (por caso la arquitectura Opteron de AMD) o al software. Excepción a lo anterior, con una complejidad idéntica a la de cache física, es que en el indexado intervengan solo los bits de la dirección virtual que no cambian, su número está dado por el tamaño de la página (recordar que es *offset* que se concatena con el *frame pointer*). Con mapeo directo de la cache el tamaño de ésta podrá ser de máximo el de una página, y si aumentamos la asociatividad el tamaño podrá aumentar. Esto es así porque para igual número de bloques, con n-way set associative se requiere un campo de *index* $\log(n)$ bits más pequeño que con mapeo directo.