Jerarquía de memoria

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Arquitectura de Computadores
Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid

1. Estructura del módulo

Este módulo está estructurado en tres lecciones:

1. **Memoria caché básica**. Realiza un repaso por los conceptos básicos de memoria caché y presenta un conjunto de optimizaciones básicas de memoria caché.

2. **Optimizaciones avanzadas de memoria caché**. Presenta un conjunto de optimizaciones asociadas a la memoria caché más avanzadas.

3. **Virtualización y jerarquía de memoria**. Revisa conceptos de soporte hardware de memoria virtual y presenta los mecanismos de soporte hardware para monitores de máquinas virtuales.

2. Memoria caché básica

Esta lección tiene la siguiente estructura general:

1. Introducción.

2. Políticas y estrategias.

3. Optimizaciones básicas.

   a) Reducción de tasa de fallos.

   b) Reducción de penalización de fallos.

   c) Reducción de tiempo de acierto.

2.1. Introducción

Si se define el rendimiento como la inversa de la latencia y se observa el periodo de 1980 a 2010, el rendimiento de los procesadores se ha incrementado en varios órdenes de magnitud con respecto al rendimiento del sistema de memoria. Esta barrera se ha incrementado todavía más con la aparición de los procesadores multi-core que colocan más presión sobre el sistema de memoria. Una de las respuestas a este problema ha sido el uso de memorias cachés.

Las memorias cachés se basan en el principio de localidad. Dicho principio tiene dos versiones: el principio de localidad espacial y el principio de localidad temporal. En ambos casos el uso de una memoria temporal más pequeña pero más rápida ayuda a mejorar significativamente el rendimiento del sistema de memoria.

Una memoria caché contiene un conjunto de bloques de memoria o líneas. Cuando se accede a un dato, si este se encuentra en la memoria cachés se dice que se ha producido un acierto. En caso de que el dato no se encuentre en la memoria caché se dice que se ha producido un fallo y es necesario traer el bloque que contiene el dato del siguiente nivel de la jerarquía de memoria.

Por tanto, el tiempo medio de acceso al sistema de memoria depende del tiempo de acceso a la memoria cachés t_A, la tasa de aciertos h y el tiempo de penalización por fallo t_F.

Esto hace que el número de ciclos necesarios para ejecutar cada instrucción se vea incrementado por el número de ciclos de detención debidos a esperas por el sistema de memoria caché. Éste último depende del número de accesos a memoria realizados, la tasa de aciertos y la penalización por fallo.

2.2. Políticas y estrategias

En general, el diseño de un elemento de la jerarquía de memoria depende de la respuesta a cuatro preguntas básicas. Esto es aplicable tanto a cualquier nivel de memoria caché como al caso de la memoria virtual.

¿Dónde se ubica un bloque en el nivel superior? En memoria caché esta pregunta da lugar a la **política de ubicación de bloque**. La política de *correspondencia asociativa por conjuntos* establece un compromiso entre la política de *correspondencia directa* y la de *correspondencia totalmente asociativa*.

¿Cómo se localiza un bloque en el nivel superior? En memoria cachés esta pregunta da lugar a la **identificación de bloque**. La dirección de memoria se divide en tres campos: etiqueta, índice y desplazamiento.

¿Qué bloque debe remplazarse en caso de fallo? En memoria caché esta pregunta da lugar a la **política de remplazo de bloque**. Esta política es relevante cuando se usa correspondencia asociativa o correspondencia asociativa por conjuntos. Las políticas más habituales son FIFO y LRU.

¿Qué ocurre en caso de escritura? En memoria caché esta pregunta da lugar a la **estrategia de escritura**. Aquí, las dos alternativas son escritura inmediata y post escritura. En el caso de la escritura inmediata, todas las escrituras van a memoria. En el caso de la post-escritura, solamente se envía a memoria un bloque cuando se remplaza en la memoria caché.

2.3. Optimizaciones básicas

2.3.1. Reducción de tasa de fallos

El **aumento de tamaño de bloque** da lugar a una menor tasa de fallos, y en consecuencia, mejora el aprovechamiento de la localidad espacial. No obstante, también da lugar a una mayor penalización por fallo.

El **aumento del tamaño de la caché** también reduce la tasa de fallos al poder almacenar más bloques en la memoria caché. No obstante, se puede incrementar el tiempo de acierto.

El **incremento de la asociatividad** también puede reducir la tasa de fallos, al reducir el número de conflictos por poderse usar más vías en un mismo conjunto.

2.3.2. Reducción de penalización de fallos

El uso de **cachés multinivel** permite establecer un equilibrio entre el incremento del tamaño de la caché y la reducción del tiempo de acceso a la caché.

La **priorización de fallos de lectura sobre escrituras** evita que un fallo de lectura tenga que esperar a que una escritura termine.

2.3.3. Reducción de tiempo de acierto

Evitar la traducción de direcciones durante el indexado permite acelerar el tiempo de acceso en caso de aciertos. Para ello, se usan las direcciones virtuales para indexar en la caché.

3. Optimizaciones de memoria caché avanzadas

Esta lección tiene la siguiente estructura general:

1. Introducción.
2. Optimizaciones avanzadas.

3.1. Introducción

Las optimizaciones avanzadas de memoria caché intentan reducir métricas como el tiempo de búsqueda, la tasa de fallos o la penalización por fallos, o bien aumentar el ancho de banda de la memoria caché.

3.2. Optimizaciones avanzadas

El uso de **cachés pequeñas** permite reducir el tiempo de búsqueda. El proceso de búsqueda incluye la selección de la línea usando el campo de índice, la lectura de la etiqueta de la línea y la comparación con la etiqueta de la dirección. Al usar una caché más pequeña el hardware de búsqueda necesario es más simple y además se permite que la caché pueda integrarse en el chip del procesador.

La **predicción de vía** se puede llevar a cabo almacenando bits adicionales para predecir la vías que se seleccionará en el próximo acceso. De esta manera se puede realizar el acceso al bloque por adelantado y comparar con una única etiqueta. Si posteriormente se detecta que se ha producido un fallo se realiza la comparación con las demás etiquetas.

Para aumentar el ancho de banda de la caché se puede **segmentar** el acceso a la caché en varios ciclos. De esta manera, se puede iniciar un nuevo acceso en cada ciclo consiguiéndose un mayor ancho de banda, aunque también se incrementa la latencia.

Los fallos de caché generan detenciones hasta que se obtiene el bloque requerido. Para ocultar las detenciones se suele recurrir a la ejecución fuera de orden, lo que provoca un problema si se está resolviendo el fallo. Una **caché no bloqueante** puede permitir accesos con acierto mientras se espera la resolución del fallo. Una solución más completa pasa por permitir fallos solapados, aunque la complejidad es mayor en este caso.

Otra técnica para mejorar el ancho de banda, consiste en permitir accesos simultáneos a distintas posiciones de la memoria caché. Esto se puede conseguir usando **cachés multi-banco**, dividiendo la memoria en múltiples bancos.

Normalmente el procesador requiere una única palabra del bloque de caché. Una mejora puede ser entonces no esperar a recibir todo el bloque de memoria para servir el dato. Las alternativas para conseguir esto son servir la **palabra crítica primero** o el **reinicio temprano**.

Para reducir la penalización de los fallos de escritura se puede usar un **búfer de escritura**. En este caso, tan pronto como el dato se ha escrito en el búfer, se considera la escritura como efectuada.

Existe una variedad de optimizaciones de caché que se pueden llevar a cabo por el **compilador**. Para reducir los fallos de instrucciones se puede realizar reordenación de procedimientos, alineación de bloques de código o linearización de saltos. Para reducir los fallos en accesos a datos se puede realizar fusión de arrays, intercambios de bucles, fusión de bucles o accesos por bloques.

Por último el uso de **lecturas adelantadas** de instrucciones o de datos permite traer a la caché instrucciones o datos antes de que se genere el acceso.

4. Virtualización y jerarquía de memoria

Esta lección tiene la siguiente estructura general:

1. Memoria virtual.
2. Políticas.
3. Tabla de páginas.
4. Máquinas virtuales.
5. MMV: Monitores de máquinas virtuales.
6. Soporte hardware para virtualización.
7. Tecnologías de virtualización.

4.1. Memoria virtual

La memoria virtual surge como mecanismo para resolver los problemas de limitaciones sobre el espacio de direcciones, además de ofrecer mecanismos de protección, traducción y compartición. De esta forma la memoria virtual permite superar el límite de memoria física ejecutando los programas en un espacio de direcciones virtuales normalizado.

Al contrario que en el caso de la memoria caché, en este caso es necesaria la colaboración del hardware y el sistema operativo. De este modo, el hardware realiza la traducción de direcciones, que es gestionada por el sistema operativo.

4.2. Políticas

En el caso de la memoria virtual, su diseño se puede definir respondiendo a las mismas preguntas planteadas para el caso de la memoria caché en la lección previa. En este caso, el bloque o unidad de transferencia es la página, de modo que los procesos se dividen en páginas y la memoria se organiza en marcos de página.

La **política de ubicación de página** es siempre totalmente asociativa. Esto es cada página se puede ubicar en cualquier marco de página de la memoria principal. El objetivo final es minimizar, cuanto sea posible, la tasa de fallos, puesto que la penalización por fallo es muy alta por las propias características de los dispositivos de almacenamiento secundario.

Para la **identificación de páginas** se mantiene una tabla de páginas por proceso. Esta tabla mantiene la correspondencia entre los identificadores de páginas y los identificadores de tabla de página. En principio, se requeriría un acceso adicional a la memoria principal para consultar dicha tabla. Para reducir el tiempo de traducción se suele utilizar un búfer de traducción: el *Translation Lookaside Buffer* o TLB, que mantiene las traducciones más frecuentemente usadas.

La **política de remplazo de páginas** suele definirse en el sistema operativo y la más utilizada es LRU (*Least-recently used*). No obstante, la arquitectura debe ofrecer soporte al sistema operativo.

Por último la **estrategia de escritura** es siempre post-escritura (o *write-back* puesto que el coste de las escrituras en disco es muy alto

4.3. Tabla de páginas

Es conveniente recordar que la tabla de páginas es realmente una estructura de datos del sistema operativo. No obstante requiere soporte hardware.

Por una parte, la unidad de gestión de memoria mantiene un registro que apunta a la dirección de comienzo de la tabla de páginas (el PTBR o *page table base register*). Cada vez que se produce un cambio de contexto, el sistema operativo es responsable de actualizar el registro con la dirección de comienzo de la tabla de páginas del proceso correspondiente.

Por otra parte, también se debe ofrecer una TLB para mantener las traducciones más frecuentes, ya que en otro caso los accesos a memoria tendrían un coste prohibitivo.

En cualquier caso, cuando los espacios de direcciones de los procesos son muy grandes, las tablas de página necesitan muchas entradas, pudiendo estar una gran cantidad de las páginas sin usarse. Para solventar este problema, las soluciones más típicas son el uso de tablas de páginas invertidas o bien de tablas de páginas multinivel.

4.4. Máquinas virtuales

La idea de máquina virtual surgió en la década de los años 60 cuando se comenzó a usar en entornos *mainframe*. Sin embargo, la idea fue prácticamente ignorada en entornos monousuario hasta finales de los 90.

Hay varias razones que han hecho que recuperen su popularidad. Por una parte, la creciente importancia del aislamiento, las seguridad y la fiabilidad, así como la necesidad de compartir un único computador por múltiples usuarios o bien múltiples subsistemas. Por otra parte el gran incremento en las prestaciones de los procesadores ha hecho que el uso de monitores de máquinas virtuales sea más aceptable.

Una máquina virtual ofrece la ilusión de que los usuarios tienen un computador completo a su disposición, incluyendo su copia del sistema operativo. Un computador puede ejecutar varias máquinas virtuales, cada una con sus sistema operativo, mientras que todos los sistemas operativos comparten

el uso del hardware. En este contexto, se suele denominar *host* a la plataforma hardware subyacente y *guest* a cada una de las máquinas virtuales que se ejecutan.

4.5. MMV: Monitores de máquinas virtuales

Se conoce como hipervisor o monitor de máquina virtual al software que da soporte a las máquinas virtuales. De esta forma las distintas máquinas virtuales se ejecutan sobre el hipervisor, que es el único que tiene acceso directo al hardware. El MMV determina la correspondencia entre los recursos virtuales y los recursos físicos compartidos.

Dependiente del recurso físico concrete se puede recurrir a la compartición en el tiempo (p. ej. CPU), el particionamiento (p. ej. disco) o la emulación por software (p. ej. tarjetas de red virtuales).

La sobrecarga derivada del uso de una máquina virtual depende del tipo de carga de trabajo. Para cargas muy ligadas a cómputo la sobrecarga suele ser prácticamente nula. Por el contrario para los programas intensivos en E/S la sobrecarga de la virtualización puede ser muy alta.

Además de la protección las máquinas virtuales presentan otras utilidades como la gestión de software permitiendo el despliegue combinado de sistema operativo y aplicaciones o la gestión de hardware permitiendo la ejecución de pilas de software separadas sobre un mismo hardware.

4.6. Soporte hardware para virtualización

Si la máquina virtual se tiene en cuenta al diseñar el juego de instrucciones del procesador es relativamente sencillo reducir el número de instrucciones que debe ejecutar el MMV. En otro caso el MMV debe interceptar las instrucciones problemáticas e introducir recursos virtuales que deben ser los únicos visibles para las máquinas virtuales.

Un caso de la virtualización de recursos es la memoria. El monitor virtual introduce el concepto de memoria virtual que es una capa intermedia entre la memoria virtual y la memoria física. En principio esto forzaría a dos etapas de traducción. Para evitarlo, los MMV suelen usar la técnica de usar una tabla de páginas en la sombra. Otro caso más complejo es el de la virtualización de la entrada/salida. En este caso se deja la parte general del driver en el huésped manteniendo la parte específica en el MMV.

4.7. Tecnologías de virtualización

La virtualización impura es una solución que se ha utilizado para aquellas arquitecturas que no son virtualizables. Existen dos alternativas: paravirtualización y traducción binaria. Con la paravirtualización se porta el código del sistema operativo huésped a una ISA modificada incluyendo invocaciones al MMV. Con la traducción binaria, se sustituyen las instrucciones no virtualizables por código de emulación o bien llamadas a la MMV.

Otra alternativa que han usado fabricantes de procesadores como Intel o AMD es la extensión de su juego de instrucciones para soportar la virtualización. Intel proporciona Intel-VT (*Intel Virtualization Technology*) y AMD proporciona AMD SVM (*Secure Virtual Machine*). Ambas tecnologías se basan en ofrecer un modo de ejecución diferenciado para el MMV.