Fundamentos de Arquitectura de Ordenadores Ingeniería Técnica en Informática Sistemas

PRÁCTICA VI
Análisis de la Jerarquía de Memoria

Objetivos

- Comprender la terminología y los conceptos manejados en el estudio de las memorias cachés: localidad espacial y temporal, bloque, tipos de fallos, políticas de escritura, de reemplazo, etc.
- Analizar la influencia de los distintos parámetros de diseño en las prestaciones de la jerarquía de memoria: cachés unificadas/separadas, tamaños de bloque, políticas de reemplazo, escritura, etc..
- Identificar las distintas causas que hacen que se produzcan los fallos en la memoria caché.
- Utilizar un simulador software de jerarquías de memorias caché para analizar el rendimiento de un determinado diseño.

Introducción

En este tema simularemos el comportamiento de un sistema de memoria que se compone de una jerarquía de dos niveles: Una memoria principal y una memoria caché. La caché es una memoria más pequeña y rápida que la principal y que se sitúa entre ésta y el procesador con el fin de acelerar el tiempo de acceso a memoria. El programa a ejecutar se caracterizará por su traza o secuencia de petición de direcciones. La simulación permite seleccionar el tamaño de la caché y de sus bloques, así como algunas de las políticas de gestión de la caché que se implementan en microprocesadores reales.

Entorno de trabajo

Simuladores dinero y Xcache16

El programa Xcache16 es un interfaz amigable para Windows que nos permite la simulación de este tipo de sistemas, ofreciendo de forma gráfica la evolución de los algoritmos. *Xcache16* realiza la simulación mediante la llamada al programa *dinero.exe* que corre bajo MSDOS.

El punto de partida de la simulación lo constituye un fichero que contiene la traza de la ejecución de un programa. Esta traza, nos muestra las referencias a memoria que el programa efectuó durante su ejecución. El fichero de traza para el programa *dinero* se compone de líneas de texto que constituyen referencias a memoria. Estas líneas constan de dos números que indican, respectivamente, el tipo de referencia y la dirección de memoria a la cual se hace referencia. Los ficheros de traza se guardan con la extensión `.din`.

Existen cinco tipos de referencias a memoria. Cada uno de ellos se identifica por un valor según se indica en la siguiente tabla:

Tipo de referencia	Identificador
Lectura de datos	0
Escritura de datos	1
Búsqueda de instrucción	2
Acceso desconocido	3
Limpieza (flush) de caché	4

Se puede acceder a direcciones de memoria comprendidas en el rango `0 - FFFFFFFF H`. Las palabras de memoria se direccionan por bytes. Un fragmento de archivo de traza sería por ejemplo:

```
0 000000
1 000000
1 000001
1 000002
2 0000A4
2 0000A5
```

donde el primer número es el tipo de operación y el segundo la dirección a la que se accede.

El procedimiento para realizar la simulación de la traza de un programa sería:

1. En la opción SIMULATION elegir RUN DINERO. Aquí indicaremos el archivo de entrada (.din) el archivo de salida (.out) y los parámetros de la caché: tamaño, algoritmo de reemplazo, ...

2. Una vez ejecutado dinero, tenemos dos opciones: ver el resultado global de la simulación (FILE - OPEN DINERO OUTPUT FILE), o bien ejecutar paso a paso la simulación (FILE - OPEN STREAM INPUT FILE). En el primer caso, hay que abrir el fichero `.out`, y el el segundo, el fichero `.din`.

En la ejecución paso a paso, la aplicación nos presenta cinco ventanas donde se muestran, respectivamente, la memoria principal, la caché, la traza de ejecución, diferentes estadísticos e información de tiempo de ejecución.

En caso de elegir cachés separadas para datos e instrucciones, las ventanas de memoria y de caché se desdoblan en dos para mostrar el acceso a instrucciones y datos por separado.

En la ventana de memoria principal (data/instruction memory status) se representa en cada casilla una palabra de memoria seguida de su ubicación. A medida que realizamos la simulación veremos cómo son leídas y/o escritas las diferentes posiciones, así como si están o no presentes en memoria.

En la ventana de caché (data/instruction caché), se representan los diferentes bloques de las que consta la caché. Cada bloque almacenará varias palabras de memoria principal. Estos bloques se organizan por conjuntos. Y un conjunto se compone de varios bloques a los que se accede de forma asociativa.

Para las simulaciones se utilizarán tanto trazas existentes de programas de prueba estandar, como trazas que generaremos nosotros a partir de un programa ensamblador de un computador DLX mediante el simulador *dlxsim* que funciona sobre Linux.

Especificación de la caché a simular

Para determinar la memoria caché que queremos simular hay que proporcionar dos tipos de información al simulador: Los **parámetros**, que determinan el tamaño de cada uno de los elementos de la caché, y las **políticas**, que indican la estrategia a seguir durante la búsqueda, actualización y reemplazo de sus datos.

Parámetros

Una caché está organizada en bloques, que son las cantidades de información mínima que se transfieren entre caché y memoria principal. El tamaño del bloque indicará el número de palabras que hay en ella, y a su vez, cada palabra se compondrá de un número de bytes.

En general, el simulador tiene una serie de parámetros fijos y otros que el usuario puede seleccionar. Son los siguientes:

- **Parámetros fijos:**
  - El tamaño de la memoria principal, $2^n bytes$. Al ser la memoria direccionable byte a byte en el rango desde 0h a FFFFFFFFh, esto nos dá un valor de $n = 32$ y una memoria de 2^{32} bytes (4 Gigabytes).
  - El tamaño de la palabra de memoria, 2^b bytes. Con palabras de 32 bits como las consideradas aquí, `b=2` y la memoria principal tiene un total de 2^{30} palabras.
- **Parámetros configurables:**
  - El tamaño de la memoria caché, 2^t bytes.

- El tamaño de bloque de caché, 2^w palabras.
- El nivel de asociatividad, 2^m, o número de bloques que tiene cada uno de los conjuntos en los que puede organizarse la memoria caché. Este parámetro determina también el número de conjuntos, 2^c, en base a la siguiente fórmula:

$$\boxed{2^c = \frac{2^t}{2^b \cdot 2^w \cdot 2^m}}$$

Políticas

Organización

Para agilizar el proceso de búsqueda de una palabra en la caché, ésta suele implementarse mediante una memoria asociativa en la que la dirección a localizar se compara simultáneamente con la dirección base de cada una de los bloques, obteniendo de inmediato el bloque buscado.

Sin embargo, el alto coste de una memoria asociativa hace que la organización más utilizada no sea **totalmente asociativa**, sino **asociativa por conjuntos**, donde la caché se divide en 2^c conjuntos de 2^m bloques cada uno. A partir de la dirección de memoria solicitada, se obtiene de forma directa el conjunto de la caché que tiene asignado, y dentro del conjunto se busca ahora asociativamente el bloque correspondiente.

Si tenemos conjuntos de un solo bloque, entonces se dice que la caché tiene una organización **directa**, ya que el bloque se obtiene directamente a partir de la dirección de memoria. Para indicar esta opción al simulador, seleccionaremos un nivel de asociatividad, 2^m, igual a la unidad (m=0). Por el contrario, para especificar una caché totalmente asociativa, seleccionaremos un nivel de asociatividad que vendrá determinado por los valores de `b`, `t` y `w`, según se indica en la siguiente fórmula:

$$\boxed{2^m = \frac{2^t}{2^w \cdot 2^b}}$$

Actualización

Cuando se solicita una operación de escritura a memoria de una celda cuyo valor se encuentra en caché, hay dos formas básicas de proceder:

- **Escritura directa** o *write-through*: El valor se actualiza en caché y en memoria principal de manera simultánea.
- **Post-escritura** o *write-back*: El valor se actualiza únicamente en la caché y la memoria principal se actualiza cuando ese bloque sea reemplazada por otra en la caché. Esta estrategia es más rápida que la anterior, pero a costa de no asegurar la consistencia de los datos de caché con sus homólogos de memoria principal.

Reemplazo

El objetivo de una memoria caché consiste en almacenar las palabras de memoria que más se referencian, con el fin de maximizar el índice de aciertos a ésta.

Cuando se accede a una palabra que no está en caché, puede introducirse en ella reemplazando a otra que ya estaba. El bloque a sustituir se selecciona mediante una política de reemplazo. Los criterios que más se utilizan para este propósito son:

- Random: Se reemplaza un bloque al azar. Es el criterio menos eficaz, pero el más barato de implementar.
- LRU (Least Recently Used): Se reemplaza el bloque menos recientemente utilizada. Es el método más eficiente, pues está basado en los *principio de localidad espacial y temporal* que caracterizan la secuencia de direcciones solicitadas por un programa. No obstante, resulta también muy caro de implementar.
- FIFO (First In First Out): Se reemplaza el bloque que más tiempo ha permanecido en la caché, independientemente de cuánto o cuándo se haya utilizado. Es un compromiso de coste y eficiencia intermedios respecto a las dos anteriores.

Precarga

Las técnicas de precarga tratan de maximizar el índice de aciertos a caché por medio de una anticipación, es decir, introduciendo las palabras de memoria en caché antes de que sean solicitadas por el procesador.

Las políticas de precarga están también basadas en los principios de localidad espacial y temporal. Las más utilizadas son:

- **Siempre:** Cuando se solicita una dirección de memoria principal, se lleva a caché su bloque y lo(s) siguiente(s).
- **Bajo fallo:** Se carga en caché lo(s) bloques siguiente(s) sólo si la dirección solicitada produjo un fallo en la caché.
- **Por demanda:** No se efectúa ningún tipo de precarga.

Visualización de la simulación

Una vez seleccionados los distintos parámetros y alternativas de diseño del sistema caché, procederemos a la ejecución de la simulación en sí.

El simulador muestra cada una de las direcciones de memoria accedidas en base al siguiente formato:

direccion base	c	w	b

donde con `c`, `w` y `b` calculamos 2^c conjuntos, 2^w palabras por bloque de caché y 2^b bytes por palabra, respectivamente, según se ha explicado anteriormente.

Además, el simulador mantiene información para cada bloque de caché, la cual puede visualizarse pulsando dos veces sobre el bloque. Esta información se compone de:

- La dirección base del bloque que se usa para localizarla en la búsqueda asociativa que se realiza dentro del conjunto en el que se encuentra.
- El contenido de las 2^w palabras de que se compone el bloque.
- Información de acceso que permite saber si el bloque ha sido leída y/o escrita.

A medida que avanza la simulación, las posiciones de la caché se van marcando con diversos colores, en función del tipo de suceso que ocurrió sobre cada una de esas posiciones la última vez que fueron referenciadas. Dichos colores nos indican:

- **Bloque no ocupado**: no se ha cargado nada aún en ese bloque.
- **Éxito en el acceso** (*caché hit*): La palabra accedida se encontraba con un valor válido en la caché.
- **Bloque no valida** (*invalid*): porque en ese bloque se ha producido una escritura y aparece un problema de inconsistencia de datos con la memoria principal.
- **Fallo de caché**: los motivos de los fallos de caché se pueden clasificar entre alguno de los siguientes:
  - **Fallo forzoso** (*Compulsory*): la primera vez que se referencia un dato que no está en caché, produce un fallo de este tipo en caché, que da lugar a una transferencia del bloque de memoria principal a caché.
  - **Fallo por capacidad** (*Capacity*): si la caché no puede contener todas los bloques que referencia durante la ejecución de un programa, aparecen fallos de "capacidad". Los nuevos datos que se referencien deben reemplazar a alguna de los bloques que había en caché (los bloques que se reemplazan dependen de la política de reemplazo elegida).
  - **Fallo por conflicto** (*Conflict*): si la organización de la caché es directa o asociativa por conjuntos, los fallos por "conflicto" ocurrirán cuando el fallo provoque el reemplazo de un bloque por otro sin que la caché esté completamente llena. Esto ocurre cuando demasiados bloques que referencia el programa, se mapean en el mismo conjunto. A este tipo de fallo se le llama también fallo con "colisión" o fallo con "interferencia".

Ejemplo

Simularemos el comportamiento de un micro con 2 cachés separadas, una para datos y otra para instrucciones. Considerad cada memoria caché de 64 Kilobytes con un tamaño de bloque de 1024 palabras. Las palabras son de 4 bytes y el tamaño de la memoria principal es de 4 Gigabytes, tal y como asume el simulador.

Se utilizan políticas de escritura directa, reemplazo FIFO y precarga bajo fallo. En cuanto a la organización de la caché, considerar las tres alternativas siguientes:

(a) Asignación directa.

(b) Asignación totalmente asociativa.

(c) Asignación asociativa de 4 conjuntos (c=2).

Se pide realizar la simulación para cada uno de los casos anteriores a partir de siguiente traza de direcciones de memoria (archivo `ex1.din`):

```
2 0
0 0
2 4
0 40
2 8
0 80
2 c
0 c0
2 10
0 100
2 14
0 0
1 0
2 18
0 40
2 1c
0 80
2 20
0 c0
2 24
0 100
2 28
```

Ejercicios

1. Obtener el estado final de la caché de datos y de instrucciones para cada una de las políticas de organización propuestas.

2. Justificad el número de fallos en la caché de datos, para cada una de dichas políticas.

3. Cambiando el algoritmo de reemplazo al LRU, haced una comparativa del número de fallos para la organización (c).

4. Representad para cada política de organización, en una gráfica la evolución del número de fallos, cuando se varía el tamaño del bloque (entre 1 palabra a 8 palabras por bloque).

Solución

Los parámetros fijos de la simulación son n=32 y b=2. Los parámetros configurables son en este caso 2^{16} para el tamaño de la caché en bytes (t=16), 2^{10} para el tamaño del bloque en palabras (w=10), y un número de conjuntos que ya depende de cada uno de los tres casos considerados.

Las direcciones aparecen descompuestas en los siguientes campos:

Direccion base	c	w	b

Resulta muy importante hacer notar que todas las políticas mencionadas no se aplican sobre las direcciones de memoria solicitadas, sino sobre sus bloques. Por tanto, lo primero que habrá que hacer será calcular el bloque en el que se encuentra cada dirección, y a partir de ahí trabajar únicamente con bloques.

a) Asignación directa

En una asignación directa hay tantos conjuntos como líneas, es decir, el nivel de asociatividad es 1 (m=0). Por tanto, al tener cada conjunto un solo bloque, en este caso, el bloque a reemplazar ya está determinada de antemano y el algoritmo FIFO no entra en juego.

El número de conjuntos puede calcularse a partir de `t`, `b`, `w` y `m` como ya vimos:

$$2^c = \frac{2^t}{2^b \cdot 2^w \cdot 2^m} = \frac{2^{16}}{2^2 \cdot 2^{10} \cdot 2^0} = 2^4 = 16\ conjuntos\ (c = 4)$$

Posiciones de los bits que forman parte de cada campo:

3322222222221111	1111	1100000000	00
1098765432109876	5432	1098765432	10

Descomposición de la traza:

Acceso 1:	0000000000000000	0000	0000000000	00

Fallo. Mete ese bloque en caché y precarga el bloque siguiente:

Precarga	0000000000000000	0001	0000000000	00

Acceso 2:	0000000000000000	0001	0000000000	00

Acierto (se accede al bloque precargado anteriormente), y no hay precarga.

Acceso 3:	0000000000000000	0001	0000000000	01

Acierto (se accede al mismo bloque), y tampoco hay precarga.

Acceso 4:	0000000000000000	1111	0000000000	00

Fallo. Mete ese bloque en caché y precarga el bloque siguiente:

Precarga	0000000000000001	0000	0000000000	00

Este bloque precargado sustituye en el conjunto 0 a la que haya.

Acceso 5:	0000000000000001	0000	0000000000	00

Acierto, pues se accede al bloque que acaba de precargarse.

Acceso 6:	0000000000000000	0000	0000000000	00

Fallo. Mete ese bloque en caché y precarga el bloque siguiente:

Precarga	0000000000000000	0001	0000000000	00

Pero como este último bloque ya estaba en la caché no hay que precargarla de nuevo.

Acceso 7:	0000000000000011	0000	0000000000	00

Fallo. Mete ese bloque en el conjunto 0 (sustituyendo a la que había) y precarga el bloque siguiente (cuya dirección base es también 0000000000000011) en el conjunto 1.

Acceso 8:	0000000000000010	0000	0000000000	00

Fallo. Mete ese bloque en el conjunto 0 sustituyendo a la anterior y precarga el bloque siguiente en el conjunto 1, con dirección base 0000000000000010.

Acceso 9:	0000000000000000	1000	0000000000	00

Fallo. Mete ese bloque en el conjunto 8 y precarga el bloque con dirección base 0000000000000000 en el conjunto 9.

El estado final de la caché en cada uno de sus 16 bloques o conjuntos, es el siguiente:

Línea	Dirección base
0	0000000000000010
1	0000000000000010
2	Vacía
3	Vacía
4	Vacía
5	Vacía
6	Vacía
7	Vacía
8	0000000000000000
9	0000000000000000
10	Vacía
11	Vacía
12	Vacía
13	Vacía
14	Vacía
15	0000000000000000

b) Asignación totalmente asociativa

En este caso, existe un único conjunto donde se almacenan todos los bloques. Por tanto, c=0 y el campo `c` del formato de la dirección desaparece. Dado que tenemos 16 bloques en caché, y sólo realizamos 9 accesos a memoria, no se va a producir ningún reemplazo.

Descomposición de la traza a realizar como ejercicio.

c) Asignación asociativa por conjuntos

Como dice la especificación del problema el número de conjuntos va a ser 4 (c=2), de forma que cada conjunto tendra 4 bloques.

Queda como ejercicio realizar la traza.

Desarrollo de la práctica

Para cada uno de los ejercicios siguientes, se realizará un estudio empírico de la ejecución dealgunas trazas sobre distintas configuraciones de jerarquía de memoria, con el fin de medir el impacto de una determinada característica en el rendimiento de de las mismas. Para cada uno de los ejercicios, se pide un resumen numérico y gráfico (p.e. diagramas de barras) de los resultados obtenidos, junto con algunos comentarios y conclusiones que pueden extraerse de cada estudio. Aparte de las simulaciones propuestas, los alumnos podrán realizar cualesquiera otras que crean convenientes, variando las trazas y/o los parámetros de las jerarquías de memoria simuladas, con el fin de ilustrar las conclusiones extraídas.

Las trazas `cc1.din`, `spice.din` y `tex.din` tienen muchísimas referencias a memoria, por lo que es demasiado lento ver cómo se accede a cada una de ellas. Cuando se tengan que simular estas trazas, se hará mediante la opción de RUN DINERO, y luego se abrirá aparte el fichero de resultados con un editor de textos.

1. **Principio de localidad**

   Un primer contacto con el simulador nos permitirá comprobar que existen programas condistinta localidad. Para ello, simularemos las trazas `cc1.din`, `spice.din` y `tex.din` en una jerarquía de memoria común, con el fin de comparar las distintas tasas de fallos obtenidas

   La jerarquía de memoria en la que simularemos las trazas será: Cachés separadas de datos e instrucciones, de 8 KB cada una, con tamaño de bloque de 32 bytes, y grado de asociatividad de 2 vías. La política de escritura es de escritura directa, y el algoritmo dereemplazo utilizado el LRU.

2. **Tamaño de caché**

   La siguiente evaluación que realizaremos medirá el impacto de variar el tamaño de la caché. Para ello, conservaremos todas las características de la jerarquía de memoria del ejercicio

1, pero ejecutaremos las pruebas con una sóla traza (la que queráis), y ejecutamos una simulación para cada uno de los tamaños de caché 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB y 256KB.

3. **Cachés unificadas/separadas**

   Para comprobar la diferencia de rendimientos entre una caché unificada y un nivel de caché dividido en una caché de instrucciones y otra de datos, compararemos los resultados del ejercicio 1 para las mismas trazas, con los obtenidos al simularlas en una jerarquía de características análogas, sólo que con una caché unificada de 16 KB.

4. **Tamaño de bloque**

   Nos interesa también medir la influencia del tamaño de bloque, una vez que se ha elegido el tamaño de la caché. Por tanto, en la jerarquía de memoria inicial, y sin cambiar el tamaño de las cachés, evaluar las mismas trazas que en el apartado anterior con los tamaños de bloque siguientes: 4, 8, 16, 32, 64, 128 y 256 bytes.

5. **Grado de asociatividad**

   Evaluar también la influencia en la tasa de fallos del grado de asociatividad de la caché. Sobre la base de la jerarquía del ejercicio 1, simular la ejecución de una traza cualquiera, variando el grado de asociatividad entre, 1 (correspondencia directa), 2, 4, 8 y 256 (totalmente asociativa) vías. Comprobar también en qué medida se cumple la regla del 2:1 simulando la traza sobre una caché unificada de 8 KB de dos vías y una de 16 KB de una sola vía.

6. **Políticas de reemplazo**

   Para las trazas del apartado I, y con esa misma jerarquía de memoria, comprobar el efecto en la tasa de fallos al usar las distintas políticas de reemplazo LRU, FIFO y RANDOM.

7. **Políticas de escritura**

   Idéntico al apartado anterior, pero contrastando la políticas de escritura directa y sin búsqueda de bloque contra la de postescritura y con búsqueda de bloque.