Practica5

15 March 2025 09:07

Laboratorio Sesión 05: Repaso de memoria cache

Objetivo

El objetivo de esta sesión es recordar conceptos de memoria cache vistos en EC. Para hacerlo programaréis un simulador de cache básico que simule dos caches de lectura.

Características de la memoria cache

En esta sesión programaremos una memoria cache con las siguientes caractarísticas:

- Las direcciones son de 32 bits (para simplificar asumiremos que todos los accesos son a bytes)
- La cache es de sólo lectura (asumiremos que todos los accesos son lecturas)
- La cache será de mapeo directo o 2 asociativa con reemplazo LRU
- Tamaño de la cache: 4 Kbytes
- Tamaño de la línea de cache: 32 bytes

Toma de contacto con el entorno simulador

El simulador se compone de 3 ficheros: `CacheSim.o`, `CacheSim.h` y `MiSimulador.c` El programa principal y algunos componentes del simulador ya están programados y se encuentran en el fichero `CacheSim.o`. Este fichero se encarga de generar las secuencias de test, de imprimir los resultados de la simulación por pantalla con un formato agradable y de comprobar el correcto funcionamiento de vuestro simulador. Antes de que empecéis a programar el simulador, es interesante hacer algunas pruebas con este entorno. Para comenzar, compilad el simulador (`MiSimulador.c` no funciona correctamente, pero compila).

```
$> gcc -m32 CacheSim.o MiSimulador.c tiempo.c -o sim
```

El programa tiene 3 tests:

- Test 0: Genera la secuencia de 20 referencias de la tabla del trabajo previo
- Test 1: Genera accesos secuenciales a un vector de enteros (1000 referencias)
- Test 2: Genera los accesos de un producto de matrices de 25x25 (62500 referencias)

Para pasar cualquiera de los tests, sólo es necesario poner el nº de test como parámetro del simulador. Por ejemplo, para pasar el test 0 escribiríamos:

```
$> sim 0
Test 0 FAIL :-(
$>
```

Evidentemente el test ha fallado, ya que aun no hemos programado el simulador. En caso de que el simulador falle, nos interesará ver qué está pasando. Para ello podemos utilizar la opción v (de verbose) en el simulador (la v debe aparecer como primer parámetro):

```
$> sim v
eca130 -> 1 MP:400136d0 1 MC:bffff3e0 TAG:78e530f byte:804822c MISS -> 804816c
eca131 -> 1 MP:400136d0 1 MC:bffff3e0 TAG:78e530f byte:804822c MISS -> 804816c
ec2172 -> 1 MP:400136d0 1 MC:bffff3e0 TAG:78e530f byte:804822c MISS -> 804816c
...
Test 0 FAIL :-(
```

Esta opción nos dará una salida parecida a la anterior. Como podéis ver las columnas corresponden básicamente a la tabla del ejercicio previo. De esta forma, comparando la salida y la tabla podemos ver dónde está el problema. Dado que en los tests 1 y 2 el número de referencias es muy alto, os recomendamos que no los probéis hasta que os funcione perfectamente el test 0. Con la opción `v`, los tests 1 y 2 se paran tan pronto aparece el primer error para ayudar a su identificación.

Programación del módulo MiSimulador.c

Para programar vuestro simulador de cache tenéis que programar 3 secciones del fichero `MiSimulador.c`:

1. **Estructuras globales** En esta sección tenéis que declarar las estructuras de datos globales necesarias para mantener el estado de la cache. Es necesario que sean globales, ya que la parte principal del simulador es la rutina `reference` que se ejecuta una vez por referencia y, como ya sabéis, su estado desaparece una vez se ejecuta.

2. **Inicialización de la cache** La rutina `init_cache` se llama antes de pasar cada test para inicializar las estructuras de datos globales necesarias. El objetivo es dejar la cache en un estado inicial correcto (cache vacía).

3. **Simulación de referencias** La simulación de las referencias tenéis que hacerla en la rutina `reference`. Esta rutina se llama una vez por cada referencia a simular. Sólo es necesario que generéis el valor correcto de las 7 variables locales que ya tenéis declaradas al inicio de la subrutina y que se corresponden básicamente a las columnas de la tabla del trabajo previo (excepto el booleano `replacement`, que no era necesario en el trabajo previo).

   ```
   void reference (unsigned int address)
   {
     unsigned int byte;
     unsigned int bloque_m;
     unsigned int linea_mc;
     unsigned int via_mc; // esta parámetro sólo se usa en el fichero
                          // MiSimulador2.c para la cache 2 asociativa
     unsigned int tag;
     unsigned int miss; // booleano que indica si es miss
     unsigned int replacement; // booleano que indica si
                               // se reemplaza una línea válida
     unsigned int tag_out; // TAG de la línea reemplazada
   ```

   En otras palabras, lo que tenéis que hacer es implementar el algoritmo que, de forma intuitiva, habéis hecho servir manualmente para rellenar la tabla del estudio previo.

Después de vuestro código, la rutina acaba con una llamada a la rutina `test_and_print` (o `test_and_print2` si es la cache 2 asociativa) para comprobar si los valores de las variables son correctos e imprimirlos por pantalla en caso de tener la opción `v` activada.

Estudio Previo

1. Rellenad la tabla de la hoja de respuestas indicando para cada referencia de la secuencia de referencias la información siguiente (en hexadecimal) para el caso de la cache directa:

   - el byte de la línea a que se accede (byte)
   - el bloque de memoria (bloque M)
   - la línea de memoria cache donde se mapeará la referencia (línea MC)
   - la etiqueta (TAG) que se guardará de esta referencia
   - si el acceso es HIT o MISS,

- y en caso de que se reemplace una línea válida, el TAG de la línea reemplazada (TAG out)

2. Rellenad la tabla de la hoja de respuestas indicando para cada referencia de la secuencia de referencias la información siguiente (en hexadecimal) para el caso de la cache 2 asociativa con reemplazo LRU:

   - el byte de la línea a que se accede (byte)
   - el bloque de memoria (bloque M)
   - el conjunto de memoria cache donde se mapeará la referencia (conj MC)
   - la vía de memoria cache donde se mapeará la referencia (VIA)
   - la etiqueta (TAG) que se guardará de esta referencia
   - si el acceso es HIT o MISS,
   - y en caso de que se reemplace una línea válida, el TAG de la línea reemplazada (TAG out)

   Suponed que el primer acceso en fallo a un determinado conjunto de cache siempre se guarda en la via 0.

3. Dado el siguiente código en C:

```
int vector[1024*10];

for (i=0;i<10240;i++)
   total=vector[i]+i;
```

   Calculad cuantos aciertos y fallos en la cache directa obtendremos al ejecutarlo suponiendo que las variables `i` y `total` se almacenan en registros y que la dirección de inicio de la variable `vector` es 0x20f2e120.

4. Repetid el cálculo suponiendo que la cache es 2 asociativa con reemplazo LRU.

5. Dado el siguiente código en C:

```
int vector[1024*10];
int vector2[1024*10];

for (i=0;i<10000;i++)
   total=vector[i]+vector2[i]+i;
```

   Calculad cuantos aciertos y fallos en la cache directa obtendremos al ejecutarlo suponiendo que las variables `i` y `total` se almacenan en registros y que la dirección de inicio de la variable `vector` es 0x20f2e120.

6. Repetid el cálculo suponiendo que la cache es 2 asociativa con reemplazo LRU.

Trabajo a realizar durante la Práctica

1. Programad una primera versión del simulador de cache directa en el fichero `MiSimulador.c`. Cuando funcione entregad en el Racó de la asignatura el fichero `MiSimulador.c`.

2. Programad un simulador de la cache 2 asociativa con reemplazo LRU en el fichero `MiSimulador2.c` y probadlo con el programa almacenado en el fichero `CacheSim2.o`. Suponed que el primer acceso en fallo a un determinado bloque de cache siempre se guarda en la via 0. Cuando funcione entregad en el Racó de la asignatura el fichero `MiSimulador2.c`.

3. Medid (usando la opción `test 2`) cuántas instrucciones ejecuta todo el programa en la primera versión que habéis programado.

4. Medid (usando la opción `test 2`) cuántas instrucciones ejecuta todo el programa en la segunda versión que habéis programado.

5. Calculad el número de aciertos y fallos de cache que se obtienen en el `test 2` con la cache directa.

6. Calculad el número de aciertos y fallos de cache que se obtienen en el `test 2` con la cache 2 asociativa.

Nombre: _____ Grupo: _____

Nombre: _____

Hoja de respuesta al Estudio Previo

1. Rellenad la siguiente tabla (en hexadecimal):

@	byte	bloque M	línea MC	TAG	HIT/MISS	TAG out
20f2e110	10	1079708	8	20F2E	M	—
20f2e111	11	1079708	8	20F2E	H	—
20f26152	12	107930A	A	20F26	M	—
20f2e113	13	1079708	8	20F2E	H	—
20f27155	15	107938A	A	20F27	M	20F26
20f27155	15	107938A	A	20F27	H	—
20f2f116	16	1079788	8	20F2F	M	20F2E
20f2e117	17	1079709	8	20F2E	M	20F2F
20f26158	18	107930A	A	20F26	M	20F27
20f2f119	19	1079788	8	20F2F	M	20F2E
50f2e210	10	2879710	10	50F2E	M	—
20f2e111	11	1079708	8	20F2E	M	20F2F
20f26152	12	1079 30A	A	20F26	H	—
50f2e213	13	2879710	10	50F2E	H	—
20f27155	15	107938A	A	20F27	M	20F26
20f27155	15	107938A	A	20F27	H	—
20f2f116	16	1079788	8	20F2F	M	20F2E
50f2e217	17	2879710	10	50F2E	H	—
20f26258	18	1079312	12	20F26	M	—
20f2f119	19	1079788	8	20F2F	H	—

Tq. TAG dif. al que hi havia

2. Rellenad la siguiente tabla (en hexadecimal):

@	byte	bloque M	conj MC	VIA	TAG	HIT/MISS	TAG out
20f2e110	10	1079708	8	1	41E5C	M	—
20f2e111	11	1079708	8	1	41E5C	H	—
20f26152	12	167930A	A	1	41E4C	M	—
20f2e113	13	1079708	8	1	41E5C	H	—
20f27155	15	107938A	A	2	41E4E	M	—
20f27155	15	107938A	A	2	41E4E	H	—
20f2f116	16	1079788	8	2	41E5E	M	—
20f2e117	17	1079708	8	1	41E5C	H	—
20f26158	18	107930A	A	1	41E4C	H	—
20f2f119	19	1079788	8	2	41E5E	H	—
50f2e210	10	2879710	10	1	A1E5C	M	—
20f2e111	11	1079708	8	1	41E5C	H	—
20f26152	12	107930A	A	1	41E4C	H	—
50f2e213	13	2879710	10	1	A1E5C	H	—
20f27155	15	167938A	A	2	41E4E	H	—
20f27155	15	107938A	A	2	41E4E	H	—
20f2f116	16	1079788	8	2	41E5E	H	—
50f2e217	17	2879710	10	1	A1E5C	H	—
20f26258	18	1079312	12	1	41E4C	M	—
20f2f119	19	1079788	8	2	41E5E	H	—

3. Para el primer código C, la cache directa obtiene:

Aciertos: 10240*4 - 1280 = 39680 Fallos: 1280

Agrupts només per 1r B de cada L
it 1: 0x20F2E120 ⇒ .0001.0010.0000
model 1: 1M 7H
10240 inst
L=20 9 B=0
120 121 122 123 124 125 126 127 128 129 12A 12B 12C 12D 12E 12F 130 131 132 133 134 135 136 137 138 139 13A 13B 13C 13D 13E 13F 140
L=0 A
⇒ .0001.0100.0000

∇ Cada vec[i] són 4B i en cache només podem anar de 1B en 1B

4. Para el primer código C, la cache 2 asociativa con reemplazo LRU obtiene:

Aciertos: 39680 Fallos: 1280

Igual que 1-assoc pq anem seq. i no aprofitem el que teníem guardat.

vec1[i] 0x20F2E120 + 0x00004000 = 4kB = 0x20F38120 vec2[i]
vec1[i] M+H+H+H
it 1: 0x20F2E120 ⇒ L=9 TAG=0x20F2E
0x20F38120 ⇒ L=9 TAG=0x20F38
it x: 2M 6H
M+H+H+H vec2[i]

5. Para el segundo código C, la cache directa obtiene:

Aciertos: 60000 Fallos: 20000

Es sobre mateixa L pq TAG diff.

6. Para el segundo código C, la cache 2 asociativa con reemplazo LRU obtiene:

Aciertos: 77500 Fallos: 2500

it 1: 0x20F2E120 → Conj = 9 TAG = 41E5C via 1 M+H+H+H
0x20F38120 → Conj = 9 TAG = 41E70 via 2 M+H+H+H
it 9: 0x20F2E140 → Conj = A TAG = 41E5C via 1 M+H+H+H
0x20F38140 → Conj = A TAG = 41E70 via 2 M+H+H+H

10000 it / 8 it · 2 Miss · 2500

model 1: 8 it → 2·4·8 accs
64 accs
2M .. 62 H

Nombre: _____ Grupo: _____

Nombre: _____

Hoja de respuestas de la práctica

1. La primera versión (cache directa) funciona correctamente (S/N): S
2. La segunda versión (cache 2 asociativa) funciona correctamente (S/N): S
3. Con la primera versión de la rutina (cache directa), el programa completo ejecuta: 27.334.003 instrucciones con la opción `test 2`.
4. Con la segunda versión de la rutina (cache 2 asociativa), el programa completo ejecuta: 27.147.684 instrucciones con la opción `test 2`.
5. Calculad el número de aciertos y fallos de cache que se obtienen en el `test 2` con la cache directa:
   Aciertos: 58272 Fallos: 4278

   ./sim v 2 | grep -c HIT
6. Calculad el número de aciertos y fallos de cache que se obtienen en el `test 2` con la cache 2 asociativa:
   Aciertos: 59432 Fallos: 3068
7. Recordad entregar los ficheros `MiSimulador.c` y `MiSimulador2.c` en el Racó de la asignatura. Debéis entregar sólo los dos ficheros fuentes, sin comprimir ni cambiarles el nombre, y sólo una versión por pareja de laboratorio (es indistinto que miembro de la pareja entregue).