AOC2 Práctica 4 - Análisis de Organizaciones de Memoria Cache

Tenemos que averiguar el tamaño de bloque idóneo para una cache que ejecute un cierto tipo de programas. Para ello se utiliza DineroIV, un simulador de memorias cache que clasifica las referencias a memoria de un programa según sean accesos a instrucción, lectura de dato o escritura de dato, y nos dice los fallos de cada tipo para una cache de un determinado tamaño, tamaño de bloque, asociatividad y política de actualización en escritura. ***Ya utilizamos la salida de este simulador en uno de los problemas****. Tras el trabajo experimental hay que desarrollar, a partir del algoritmo que describe el comportamiento de cada tipo de cache, el modelo analítico para calcular los costes temporales, es decir, las expresiones de tiempo efectivo de acceso por referencia y de tiempo de ejecución. Supone obtener el tiempo de ejecución (el CPI) a partir de este modelo de trabajo (simulador de fallos + descripción comportamiento de la cache).* ***Todo esto ya lo hemos explicado en teoría y lo hemos hecho en diferentes problemas.*** *Después se introducen los datos de simulación en el modelo y se comparan los diseños.*

Parámetros de la cache base y tiempo de acceso

Vamos a considerar una memoria cache (Mc) unificada de 8KB, organización en un banco, tecnología 45 nm y asociatividad 2. Ejecutando los programas que se citan más adelante, el procesador tiene un CPI = 4 con un sistema ideal de memoria (cualquier búsqueda de instrucción, lectura o escritura de dato cuesta un ciclo). Suponemos que el tiempo de ciclo del procesador viene condicionado por el tiempo de acceso a la Mc. El tiempo de acceso a la cache hay que averiguarlo introduciendo los parámetros de la cache en la interfaz Web de CACTI[1] (http://quid.hpl.hp.com:9081/cacti/): observa que cada tamaño de bloque (Line Size según terminología de Cacti) supone un tiempo de acceso diferente. Puedes comprobar por ejemplo una cache de 8 KB, 1 banco, s=1 y tamaño de bloque 16 B con tecnología de 45 nm, tiene un tiempo de acceso de ~ 0.313 ns.

Modelado del acceso a memoria

La Memoria Principal (Mp) se accede desde el chip mediante un bus con anchura de datos de 8 bytes. Independientemente de la frecuencia de reloj del procesador, la memoria principal suministra el primer fragmento de 8 bytes en 30 ns y los restantes (en caso de transferencia de bloque) a razón de uno cada 4 ns. La escritura sigue una temporización similar pero al revés: 4 ns por trozo de 8 bytes excepto el último que requiere 35 ns. Puesto que la frecuencia del procesador no es múltiplo de la de bus, es difícil estimar el tiempo de cada transacción medido en ciclos de procesador; por tanto, haced la siguiente aproximación: *para cada interacción que implique al bus (lectura de bloque, escritura de bloque sucio, etc), calculad en primer lugar el tiempo total en nanosegundos y después contad el número de ciclos de procesador equivalentes (parte entera por arriba)*.

Para disminuir el impacto negativo de las escrituras se añade a la Mc un buffer que oculta parcialmente el tiempo de escritura de Mc a Mp. Tanto si la política es *copy-back* (CB) como *write-through* (WT) *suponemos que tal buffer disminuye en un 45 % el tiempo que invertiría cada política en descargar contenidos de Mc hacia Mp*.

En la página 2 se facilitan los algoritmos que describen el comportamiento de esta cache.

[1] Más información sobre CACTI en http://www.hpl.hp.com/research/cacti/

Descripción del comportamiento de la cache:

Write-Through con buffer de escrituras (Bufu)	Tiempos
look-up(@x);	[1]
if miss(@x) **{** Mp{rB,@x}; **waitfor** Mp; Mc+X; **}**	[CrB(Mp)+1]
MRU(X);	[0]
switch(proc_r/w)	
case proc_r: **ret** x';	[0]
case proc_w:	[0]
{ Mc+x'; Bufu{wW,X'}; **waitfor** Bufu; **ret**; **}**	[CwW(bufu)]

Copy-Back con buffer de escrituras (Bufu)	Tiempos
look-up(@x);	[1]
if miss(@x) **{** U=LRU(); Mc-U;	[0]
if sucio(u) **{** Bufu{wB,U}; **waitfor** Bufu; **}**	[CwB(bufu)]
Mp{rB,@x}; **waitfor** Mp; Mc+X; **}**	[CrB(Mp)+1]
MRU(X);	[0]
switch(proc_r/w)	
case proc_r: **ret** x';	[0]
case proc_w: **{** Mc+x'; sucio(x)=true; **ret**; **}**	[1]

Modelo de Mc: simulador dineroIV

Disponemos de un simulador de Mc (`dineroIV`) instalado en `hendrix.cps.unizar.es` que acepta por la entrada estándar una traza en *formato dinero*[2] (`.din`), y genera por la salida estándar una estadística de tipos de acceso, fallos y tráfico de la Mc simulada. Para poder utilizar este simulador hay que añadir la siguiente línea al fichero `.software` que se encuentra en el $HOME de cada usuario[3]:

```
dinero          # dinero IV (simulador de caches)
```

Este cambio tendrá efecto en posteriores aperturas de sesión.

En la orden de invocación de `dineroIV` se pueden especificar numerosas opciones (dimensionado, políticas de escritura etc). Por ejemplo, la orden:

```
zcat dinero/tex.din.Z | dineroIV -informat d -l1-isize 16K
    -l1-dsize 8K -l1-ibsize 32 -l1-dbsize 32
```

simula una Mc separada de 16KB para instrucciones y 8KB para datos de correspondencia directa (es el valor por defecto) con tamaño de bloque de 32B. Las trazas contienen alrededor de un millón de referencias y están comprimidas (`.Z`) para ahorrar espacio de disco.

Experimentación y métricas de comparación

1. Realizar los **experimentos variando el tamaño de bloque** (16B, 32B, 64B y 128B) y la política de actualización en escritura (WT y CB).

Experimentaremos con la traza de ejecución de uno de estos tres programas: `tex`, `spice` ó `cc1` (elegid uno de ellos). Se encuentran en el siguiente directorio:

```
/export/home/practicas/Practicas/ac/traces/dinero/
```

Puedes ponerte de acuerdo con otros compañeros para experimentar cada uno con un programa diferente y comparar los resultados después, o bien sacar medias y analizar los resultados de forma conjunta.

2. Realizad una **tabla y una gráfica con los valores del *TEApr*** *(Tiempo efectivo de acceso = tiempo medio de acceso por ref en ns)* en todos los casos propuestos.
3. Realizad una **tabla y una gráfica con los valores del CPI y el TPI** *(tiempo por instrucción)* para cada tamaño de bloque. Explicad la forma de las curvas y decidid si aparece lo esperado o no.

[2] Formato documentado en el man de dineroIV en el que cada línea contiene:<*tipo*><*dirección* > donde *tipo* codifica si es acceso a instrucción, lectura de dato o escritura de dato, y *dirección* es la de la referencia a memoria.

[3] `.software` Es un fichero oculto. Para listarlo: `ls -la`

Notación recomendada para las fórmulas del cálculo de tiempos.

Además de las medidas citadas, *TEApr*, CPI y TPI (tiempo medio por instrucción en ns), si se precisa alguna métrica adicional se utilizará la siguiente notación:

TEApi: **Tiempo efectivo** (= tiempo medio) de acceso a memoria ***por instrucción*** en ns. O sea el tiempo medio de acceso a la jerarquía ***por instrucción***. $TefPI = CefPI * Tc$

CEApi: Ciclos efectivo (=ciclos medios) de acceso a memoria *por instrucción*.

TRM: Total de referencias a memoria

m_x tasas de fallos en tanto por uno.

x = nada. Cuenta global: $m = \#fallos/\#ref$

u Cache unificada (no la utilizamos en esta práctica)

i Cache de sólo instrucciones. $m_i = \#fins/I$

d Cache de sólo datos. $m_d = (\#fesc + flec\#)/(\#esc + \#lec)$

I Total de instrucciones ejecutadas *(dynamic count of instructions)*

f_{wt} , f_{rt} porcentaje de accesos de escritura (o lectura) sobre el total de referencias a memoria.

p.e: $f_{wt} = \#esc / (I + \#esc + \#lec)$

f_w , f_r porcentaje de accesos de escritura (o lectura) sobre el total de referencias a datos.

p.e: $f_w = \#esc / (\#esc + \#lec)$

Modelo temporal (resumen del Tema 9 – Comportamiento)

En primer lugar introducimos una sintaxis de descripción de componentes cercana al lenguaje C, añadiendo algún nuevo elemento. Nuestro modelo supone que cada componente es independiente de los demás, recibe peticiones de un cliente al que debe servir y puede enviar nuevas peticiones a otros componentes, quedando o no bloqueado hasta recibir respuesta.

Notación: variables y acciones

x'	palabra accedida por el procesador.
x	bloque completo de **x'**
@x', @x	dirección de palabra y número de bloque.
X	pareja **<x,@x>**.
X'	pareja **<x',@x>**.
u	bloque víctima, expulsado de Mc para dejar sitio a otro.
look-up(@x)	selección contenedor y comprobación fallo/acierto.
MRU(@x)	**x** pasa a la cabeza de la lista LRU. Pasa a ser el *Most Recently Used*.
m(cmd,param)	Se pasa al componente **m** el comando **cmd** con unos parámetros **param** Ejemplo: **Mp(rB,@x)** petición de lectura del bloque **x** a memoria principal. Esta petición no implica el bloqueo del módulo demandante. Comandos: **rW** lee palabra **rB** lee bloque **wW** escribe palabra **wB** escribe bloque (en una expulsión de Mc a Mp)
waitfor m	espera respuesta del componente **m**.
ret	despierta al componente cliente
ret resul	despierta al componente cliente y le devuelve **resul**.
m + bloque	guarda bloque en **m**. Ejemplo: **Mc+x**, copia **x** en **Mc**.
m - bloque	Elimina bloque de **m**.
m + pal	guarda palabra en **m**. Ejemplo: **Mc+x'**, escribe **x'** en **Mc**.

Notación: flujo de control y retardos

`if` `cond accs` si se cumple la condición `cond` se ejecutan las acciones `accs`. Si hay más de una acción, se encierran entre llaves `{}`.

`switch(``caso`**`)`** según el valor de `caso` se selecciona la opción correspondiente en las líneas siguientes **`case`**.

`[n]` tiempo asociado (**n** ciclos) a las acciones y/o preguntas de la línea. En caso de ser una línea con ejecución condicional SOLO se invierte el tiempo si se cumple la condición.

`linea1;` líneas consecutivas se ejecutan secuencialmente,

`linea2;` transcurrido el tiempo asociado a cada línea.

`{` `acciones`**`}`** las acciones entre llaves se ejecutan en paralelo hasta llegar a una sentencia **`waitfor`**. Por ejemplo:

```
{ acc1;acc2; waitfor m;acc3;acc4;acc5;}          [n]
```

implica la ejecución en paralelo de las acciones 1 y 2; la espera hasta que el componente `m` termine su servicio, y la posterior ejecución en paralelo de las acciones 3, 4 y 5. Todo ello consume `n` ciclos.

Notación: costes (en ciclos)

Ciclos de transferencia de bloque desde Mp hacia Mc	`CrB(Mp)`
Ciclos de transferencia de palabra desde Mc hacia Mp	`CwW(Mp)`
Ciclos de transferencia de bloque desde Mc hacia Mp	`CwB(Mp)`
Ciclos de copia de palabra en `bufu`.	`CwW(bufu)`
Ciclos de copia de bloque en `bufu`.	`CwB(bufu)`

Algoritmos de las opciones a comparar en la práctica

Las dos primeras opciones (sin *buffering*) no se utilizan en esta práctica, pero al ser las que se explican en las clases teóricas pueden servir para entender la notación y comparar con las opciones con *buffering*.

Relación entre tiempo de ejecución y ciclos efectivos (CEApr)

Para conocer el tiempo de ejecución de un programa (**Tex**) de referencia (*benchmark*) con diferentes caches no podemos evidentemente ejecutar ese programa de forma nativa cambiando cada vez el procesador (¡no lo hemos fabricado aún!). Sin embargo podemos saberlo si disponemos de dos informaciones. Por una parte de una descripción de las acciones de una cache determinada ante cada tipo de referencia y evento, con el coste en ciclos en cada caso (lookup, petición de bloque a memoria en caso de fallo etc). Por otra de un programa (un simulador) que extraiga sólo las referencias a memoria del *benchmark*, y que simule sólo los fallos y aciertos en la cache, sin tener en cuenta su coste temporal. Es lo que estamos haciendo en esta práctica.

A partir de la descripción de la cache podemos establecer un modelo analítico (la expresión del CEApr). A partir del simulador podemos conocer los valores de fallos y tipos de referencia (I, $f_{wt,}$ $f_{rt)}$. A partir de otros datos (normalmente un simulador ciclo a ciclo) podemos conocer el CPI con memoria ideal.

PROBLEMA: Cómo averiguamos el Tex a partir del CEApr y conociendo el CPI con memoria ideal?

INDICACIONES:

- Conocemos el CEApr una vez hallada su expresión a partir de la descripción de la cache y a partir de los datos de simulación.

- Va mos a llamarle *Cai* a los *ciclos de acceso por referencia con memoria ideal* (una cache infinita en la que acierto siempre).

- Vamos a llamarle *CPIpen* al cociente entre el *total de ciclos de penalización con memoria no ideal* (por tener fallos en cache) partido por I. Es decir, a los ciclos medios por instrucción perdidos por fallos en cache.

1. Teniendo en cuenta que:

CEApr = (*Ciclos de acceso con memoria ideal* + *ciclos de penalización con memoria no ideal*) / TRM

Ciclos de acceso a memoria ideal = TRM * Cai

CPI real = CPI con memoria ideal + CPIpen

TRM = $I + f_{wt} + f_{rt} = I + f_{wrt}$

2. Expresa CEApr en funión de CPIpen y f_{wrt} y despeja *CPIpen*
3. Ya puedes calcular Tex