FACULTAD DE INGENIERIA
Universidad de Buenos Aires

TP2 - Memorias caché

[66.20] Organización del Computador
Segundo cuatrimestre de 2020

Mariotti, Franco	102223	fmariotti@fi.uba.ar	Franco Mariotti
Fabbiano, Fernando	102464	ffabbiano@fi.uba.ar	Fernado Fabbiano
Alasino, Franco Federico	102165	falasino@fi.uba.ar	Franco Alasino

Índice

1. Introducción

En el presente trabajo se nos solicitó simular el funcionamiento de una caché. El mismo nos permitió poner en practica toda la teoría vista, obligándonos a adentrarnos en el paso a paso de como los datos van y vienen de memoria principal.

De la teoría, sabemos que la memoria caché surge de la necesidad de acortar la brecha entre la velocidad del procesador, y la velocidad con la que se acceden a datos de memoria principal. Si bien uno se puede conformar con esto, poder adentrarse y simular el funcionamiento de una caché, hace entender realmente el porque de la existencia de la misma.

En este caso, el enunciado pide simular una memoria Write Back, Write Allocate. Nuevamente, nos encontramos en un inicio ante conceptos mas que nada teóricos. Luego de repasar un poco esta teoría, y hacer algunos seguimientos con lápiz y papel de lo que debería pasar al haber una lectura, una escritura, o un miss, se procedió a implementar dichos métodos.

Consideramos que para este Trabajo Práctico, la implementación en C no fue de gran dificultad, como si quizás lo fue entender que debía suceder en cada uno de los casos.

2. Diseño e Implementación

El programa consiste en un módulo principal `-main-` que se encarga de handlear las distintas opciones que se le brindan al usuario, sabiendo como responder a los comandos requeridos en el enunciado. Esta función main la podemos encontrar en el archivo `'main.c'`. A su vez, podemos encontrar la función `start_simulation`, que se encarga de leer el archivo de entrada, parsearlo e iniciar la simulación de la caché.

En el archivo `'cache.c'` se pueden encontrar la implementación de todas las funciones necesarias para simular la Caché pedida, tanto aquellas requeridas en el enunciado, como aquellas privadas propias de la implementación de los desarrolladores.

Se tomó un espacio de direcciones de 16 bits, lo que lleva entonces a tener una memoria principal de 64 KB. Luego, se procedió a declarar un arreglo de longitud igual 32768. Este número representa la cantidad total de palabras de 2 bytes que entran en los 64KB de nuestra memoria.

3. Proceso de Compilación

Para la compilación del programa, basta con ingresar en una consola, en el directorio correspondiente al proyecto, el comando `'make'`.

Esto generará automáticamente los ejecutables, tanto el principal que se guardará en el directorio actual, como el de pruebas que guardará dentro de la carpeta tests. El primero se guarda con el nombre de `'tp2'` mientras que el segundo con el nombre de `'tests_tp2'`.

Para la ejecución y posterior utilización del programa, se debe ingresar `'./tp2 -comando archivoDeEntrada'` donde `'archivoDeEntrada'` es el archivo desde el cual se leeran los comandos a ejecutar y `'comando'` puede ser una de la siguientes opciones:

- -V, - -version ⟶ Print version and quit
- -h, - -help ⟶ Print información de ayuda acerca de los comandos
- options archivo ⟶ `archivo` contiene una serie de comandos que se deben parsear, interpretear y ejecutar. `options` es un conjuntos de opciones de la siguiente lista:
  - -o, –output ⟶ Archivo de salida
  - -w, –ways ⟶ Cantidad de vías
  - -c, –cachesize ⟶ Tamaño de la caché en kilobytes
  - -b, –blocksize ⟶ Tamaño de bloque en bytes

4. Casos de Prueba

4.1. Archivos de prueba

En esta sección adjuntaremos las salidas del programa, para los archivos prueba1.mem y prueba2.mem para las siguientes cachés: [4 KB, 4WSA, 32bytes] y [16KB, una via, 128 bytes].

Además adjuntamos la salida para ambas caches utilizando un archivo creado por nosotros llamado prueba3.mem, cuyo contenido es el siguiente:

```
W 0, 123
W 1024, 234
W 2048, 33
W 3072, 44
W 4096, 55
W 8192, 100
W 16384, 101
R 0
R 1024
R 2048
R 3072
R 4096
R 8192
R 16384
MR
```

4.1.1. Salida archivo prueba1.mem con cache: [4 KB, 4WSA, 32bytes]

```
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
Value:255
HIT DE LECTURA
Value:254
HIT DE LECTURA
Value:248
HIT DE LECTURA
Value:96
HIT DE LECTURA
Miss rate:0.500000
```

4.1.2. Salida archivo prueba1.mem con cache: [16 KB, 1WSA, 128bytes]

```
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
Value:255
MISS DE LECTURA
Value:254
MISS DE LECTURA
Value:248
MISS DE LECTURA
Value:96
```

```
MISS DE LECTURA
Miss rate:1.000000
```

4.1.3. Salida archivo prueba2.mem con cache: [4 KB, 4WSA, 32bytes]

```
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
Value:123
MISS DE LECTURA
Value:234
MISS DE LECTURA
Value:33
MISS DE LECTURA
Value:44
MISS DE LECTURA
Value:55
MISS DE LECTURA
Miss rate:1.000000
```

4.1.4. Salida archivo prueba2.mem con cache: [16 KB, 1WSA, 128bytes]

```
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
Value:123
HIT DE LECTURA
Value:234
HIT DE LECTURA
Value:33
HIT DE LECTURA
Value:44
HIT DE LECTURA
Value:55
HIT DE LECTURA
Miss rate:0.500000
```

4.1.5. Salida archivo prueba3.mem con cache: [4 KB, 4WSA, 32bytes]

```
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
Value:123
```

```
MISS DE LECTURA
Value:234
MISS DE LECTURA
Value:33
MISS DE LECTURA
Value:44
MISS DE LECTURA
Value:55
MISS DE LECTURA
Value:100
MISS DE LECTURA
Value:101
MISS DE LECTURA
Miss rate:1.000000
```

4.1.6. Salida archivo prueba3.mem con cache: [16 KB, 1WSA, 128bytes]

```
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
MISS DE ESCRITURA
Value:123
MISS DE LECTURA
Value:234
HIT DE LECTURA
Value:33
HIT DE LECTURA
Value:44
HIT DE LECTURA
Value:55
HIT DE LECTURA
Value:100
HIT DE LECTURA
Value:101
MISS DE LECTURA
Miss rate:0.642857
```

4.2. Pruebas unitarias

Al utilizar el comando make ya se nos creara el ejecutable de las pruebas unitarias llamado `'tests_tp2'`. Para correrlas, debemos igresar al directorio tests con el comando `'cd tests'` y luego ingresar `'./tests_tp2'`. Al correr las pruebas obtendremos la siguiente salida:

```
---Incio pruebas de inicializacion de la cache --

La cache se inicializa correctamente: OK

---Incio pruebas de escritura en cache --

Al escribir un dato que no esta en cache hay miss y se escribe en cache : OK
```

```
Al escribir un dato que esta en cache hay hit y se escribe en cache : OK
Al quitar un bloque de cache que habia sido escrito (dirtybit == 1),
este se escribe en mem ppal : OK

---Incio pruebas de lectura en cache --

Al leer un dato que no esta en cache hay miss y se lee el dato correctamente : OK
Al leer un dato que esta en cache hay hit y se lee el dato correctamente : OK

---Incio pruebas de calculo de miss rate --

El Miss Rate se calcula correctamente: OK
```

5. Conclusiones

Este trabajo nos ayudo a comprender mejor como funciona una cache y a poner en practica los conocimientos teóricos previos acerca de memorias caché en general.

En particular fue interesante poder aplicar los conceptos de Write back y Write allocate, dado que pudimos ver con mas claridad la utilidad de estos. Del mismo modo, pensar e implementar la política de reemplazo LRU, nos resultó de gran interés y utilidad para fijar conceptos. A simple vista, esto último parecía algo bastante complicado de implementar pero luego de releer la teoría, y de conversar e intercambiar opiniones y puntos de vista entre los desarrolladores, pudimos sacarlo adelante con éxito y sin mayores complicaciones.

Un detalle que notamos interesante destacar es que al correr el archivo de prueba 1 con 4 vías, 8k btyes de cache y bloques de 16 bytes, obtuvimos una tasa de miss del 50 % mientras que al reducir las vías a 2 obtuvimos una tasa de miss del 100 %. Esto nos demuestra como un cambio , que podría parecer pequeño a simple vista como reducir las vías de 4 a 2, en la estructura de la caché puede hacer que su performance varíe y pueda hasta resultar inútil, como en el caso mencionado anteriormente.

Una vez corridas las pruebas otorgadas en el enunciado, pudimos realmente observar como el hecho de agregar una memoria caché en nuestras computadoras, puede mejorar notablemente la performance de los procesos. Si extrapolamos las observaciones y conclusiones de este Trabajo Práctico, y pensamos en una caché de tres niveles como las que existen hoy en día, podemos intuir que las mejoras en tiempos de acceso a memoria son notables, lo cual es necesario para reducir la brecha mencionada con anterioridad en la Introducción.

Referencias

[1] Repositorio de Github,
https://github.com/FrancoMariotti/tp2-OrgaDeComputadoras