UNIVERSIDAD DE COSTA RICA

Escuela de Ciencias de la Computación e Informática

CI-0120 Arquitectura de computadares

Grupo 1 profesor Francisco Arroyo

Enunciado segunda etapa del proyecto

(cuenta por un 15% de la nota de proyecto, un 3.75% de la nota final)

Fecha de entrega: Viernes 2021/May/21
Modalidad: individual

a) Memorias cache

En esta etapa vamos a desarrollar un simulador para memorias cache, a fin de estudiar y comparar la efectividad de configuraciones distintas para este tipo de memorias.

La memoria cache podrá ser configurada de acuerdo con los siguientes parámetros:

- Número de conjuntos en la cache (**Sets** positivo y potencia de dos)
- Número de bloques en cada conjunto (**Blocks-Set** positivo y potencia de dos)
- Número de bytes en cada bloque (**Bytes-Block** positivo y potencia de dos, mayor o igual a cuatro)
- Bandera para "**write-allocate**" o "**no-write-allocate**"
- Bandera para "**write-through**" o "**write-back**"
- Algoritmo para reemplazo (**LRU**, **FIFO**, **random**)
- Tiempo **T** de acceso a la memoria cache, medida en ciclos de procesador para mover un bloque de información
- Tiempo $\mathbf{T_n}$ de acceso a la memoria del siguiente nivel, para acumular cuando hay que buscar los datos en ese nivel
- Nombre de la memoria, por ejemplo "L1", "L2", "L3" o "L4"

Algunas combinaciones válidas:

- Una cache con *N* conjuntos de *1* bloque es "**direct-mapped**"
- Una cache con *N* conjuntos de *M* bloques es "**m-way set-associative**"
- Una cache con *1* conjunto de *N* bloques es "**fully associative**"

La cache más pequeña que se puede definir es un conjunto de un bloque de 4 bytes, puede recordar una posición de memoria de 4 bytes y nada más, puede ser útil para realizar pruebas funcionales del programa (test cases).

El parámetro "**write-allocate**" determina lo que hay que hacer en caso de que tengamos un fallo durante un "store". Si la bandera está levantada (**true**), el bloque de memoria

involucrado es guardado en la memoria cache antes de que el "store" sea completado. Si no está levantada, un fallo no modificaría la cache. Además, si la bandera es "**write-through**" está levantada un "store" escribiría en la memoria cache y también en el siguiente nivel; si la bandera es "**write-back**" un "store" escribe únicamente en este nivel de la cache y marca el bloque como sucio (*dirty*), cuando este bloque sea elegido por el algoritmo de reemplazo (**eviction** o **desalojo**), deberá ser escrito a la memoria del siguiente nivel antes de ser reemplazado.

Hay parámetros que no tiene sentido combinarlos, por ejemplo, "**no-write-allocate**" con "**write-back**" por que no seremos capaces de escribir en la memoria cache el dato indicado. Su programa debe controlar este tipo de inconsistencias.

El parámetro para el algoritmo de reemplazo es relevante solo para las caches asociativas, en las "direct-mapped" no hay escogencia del bloque a reemplazar:

- LRU (Least recently used) desaloja el bloque que se utilizó hace más tiempo
- FIFO desaloja el bloque que lleva más tiempo en la cache
- Random desaloja un bloque al azar, sin tomar en cuenta el tiempo que lleve en la memoria

Hay muchos más detalles que intervienen en un procesador real, por ejemplo, "write-buffers", "smart ways to fill cache blocks" y otros que no es necesario implantar en nuestro simulador.

Este simulador leerá de la entrada estandar un archivo con una **traza simplificada** de accesos a memoria, simulará el acceso a la memoria cache y generará estadísticas a la salida estandar. En el archivo de la traza cada acceso a memoria viene en una línea por separado. Cada línea contiene un primer caracter que indica si se trata de un "load" ('l') o un "store" ('s'), un espacio en blanco y una dirección de 32 bits en hexadecimal para la operación indicada (`l 0x00ecc100` o `s 0x00ecc100`). No son importantes los datos en las instrucciones ya que no van a ser almacenados en la memoria, por lo que se puede ignorar cualquier otro elemento adicional que aparezca en una línea.

Un ejemplo de corrida sería:

```
./cacheSim.out 256 4 16 write-allocate write-back lru <archivoTraza
```

este simularía una cache con 256 conjuntos de cuatro bloques cada uno (aka 4-way set-associative cache), con bloques de 16 bytes cada uno; la cache utiliza las estrategias de "write-allocate" y "write-back" (o no "write-through") y desaloja el bloque que no ha sido utilizado más recientemente (lru). El tamaño de esta cache sería 16KB para el almacenamiento de los datos, sin contar los elementos adicionales para su operación, sin embargo en nuestro simulador no vamos a almacenar datos, solo contabilizar los ciclos gastados por los accesos. Se espera una salida similar a la siguiente:

Total “loads”:	318597
Total “stores”:	195486
Load hits:	315798
Load misses:	33599
Store hits:	187250
Store misses:	7236
Total CPU cycles:	9364783

Vamos a desarrollar este simulador de memoria cache en varias partes, no deben perder de vista que la memoria cache debe funcionar correctamente con todas las opciones de configuración descritas, a pesar de que lo desarrollaremos en partes.

Para esta segunda etapa del proyecto la entrega comprendería una memoria cache funcional con:

1. “direct-mapped”, “write-through” y “no-write allocate”
2. “fully associative”, “write-through” y “no-write-allocate”

En las demás etapas desarrollaremos los elementos faltantes.

Para referencia pueden utilizar el simulador que está instalado en el servidor:

https://os.ecci.ucr.ac.cr/ParaCache/fa.html

Referencias

- Libro de Stallings, capitulo 4
- Libro de Hennessy, capítulo 2