Arquitetura de Computadores 2018/19

TPC 4

Prazo de entrega: 23:59 de 3 de junho de 2019

Tópico: *Caches.*

Este trabalho de casa consiste em dois exercícios a resolver individualmente. Pode discutir ideias gerais com colegas, mas a solução e a escrita do código deve ser estritamente individual. A entrega será via Mooshak do DI. Todas as resoluções serão comparadas de forma automática e os casos de fraude serão punidos de acordo com os regulamentos em vigor.
NOTA: só são aceites as primeiras 10 submissões ao mooshak de cada problema, sendo avaliada a com mais pontos no mooshak. Se submeter mais vezes, serão ignoradas!

Introdução

Este trabalho baseia-se na matéria das aulas teóricas sobre caches (ver slides aulas 20 e 21). A matéria encontra-se também no capítulo 6 do livro *Computer Systems: A Programmer´s Perspective 2nd Ed*. O exercício recorre a um simulador, em C (cache.c), que procura simular o comportamento de uma cache de uma determinada arquitetura perante uma sequência de acessos a endereços de memória. O código necessário para este trabalho é fornecido via CLIP.

Simulador de cache

Considere o simulador de cache no ficheiro `cache.c`. Este recebe da linha de comandos um ficheiro de texto contendo a sequência de endereços de memória que determinado CPU usou ao executar determinado programa. O ficheiro com o traço dos endereços de memória acedidos é um ficheiro de texto com o seguinte formato (exemplo):

```
0041f7a0 R
31348900 W
```

Em cada linha, o primeiro valor é um inteiro (hexadecimal) com o endereço de memória a que o CPU acede. A letra representa o tipo de operação realizada naquele endereço de memória: R (leitura) ou W (escrita). Estão disponíveis dois ficheiros que contêm sequências de referências à memória feitas por dois programas reais, nomeadamente:

`gcc.trace`	O compilador GNU CC
`bzip.trace`	O compressor BZIP

A simulação não está completa. Veja o código fornecido. Pretende-se que a termine para simular o funcionamento de uma cache associativa por grupos (ou conjuntos) com as seguintes características:

- linhas de cache com 32 bytes;
- 512 linhas organizadas em grupos de 4 linhas (ou seja, um total de 128 grupos).

A quando de um miss, quer numa leitura quer numa escrita, o bloco de memória será sempre lido para cache. Este ficará na primeira linha livre do respetivo grupo (Cvalid==0), se existir. Se todas estiverem ocupadas, será lida para a primeira linha que não foi acedida recentemente (Clru==0). Este bit funciona como no pseudo-LRU exemplificado na aula 21.

Deve sempre atualizar os contadores de números de acessos a memória e *cache hits*, de acordo com as operações realizadas na cache:

`memR` = número de leituras efetivas de memória
`memW` = número de escritas efetivas na memória
`hitR` = número de Hits nos acessos de leitura
`hitW` = número de Hits nos acessos de escrita

Existem ainda os seguintes contadores que já são atualizados na função main:

`R` = número de acessos de leitura
`W` = número de acessos de escrita

Note: além da correta implementação do funcionamento e gestão da cache, deve garantir que estes contadores são incrementados de acordo com os respetivos acontecimentos. A avaliação no mooshak depende destes valores estarem corretos.
O programa que submete ao mooshak deve escrever exatamente as mensagens já existentes, sem alterações e apenas essas.

Exercício 1 (Prob A – 70%)

Considera-se que as escritas são imediatas (*write-through*) e não necessita assim de marcar as linhas alteradas (não use o Cdirty e conte sempre cada escrita como uma escrita na memória).
O resultado obtido para os endereços em `bzip.trace`, uma execução do compressor de ficheiros *bzip*, é:

```
16384 bytes in cache as 128 sets, 4 lines per set with 32 bytes (block size)
Reads:   877581,  868915 hits (99.0%)
Writes:  122419,  119350 hits (97.5%)
Total:  1000000,  988265 hits (98.8%)
MemR: 11735, MemW: 122419 (total 134154, 13.4%)
```

Exercício 2 (Prob B – 30%)

Altere a solução anterior para suportar agora escritas diferidas (*write-back*). Recorde que tal leva a que se marque as linhas alteradas (Cdirty) e a que estas sejam apenas escritas em memória antes de substituídas por novos blocos.
O resultado obtido para a execução em `bzip.trace` é:

```
16384 bytes in cache as 128 sets, 4 lines per set with 32 bytes (block size)
Reads:   877581,  868915 hits (99.0%)
Writes:  122419,  119350 hits (97.5%)
Total:  1000000,  988265 hits (98.8%)
MemR: 11735, MemW: 3485 (total 15220, 1.5%)
```

Repare no novo valor de escritas na memória.