Centro Federal de Educação Tecnológica de Minas Gerais

Departamento de Computação
Curso de graduação em Engenharia da Computação
Organização e Arquitetura de Computadores II
Profa. Daniela Cristina Cascini Kupsch (cascini@decom.cefetmg.br)

Comece a fazer esta lista imediatamente. Você nunca terá tanto tempo para resolvê-la quanto agora.
Após a data da entrega você pode entregar a lista para ser corrigida, mas não receberá os pontos pelo trabalho.

LISTA IV

Exercício 1

O desempenho de um multiprocessador com coerência de cache por snooping depende de muitas questões de implementação que determinam a rapidez com que uma cache responde com dados em um bloco no estado Exclusivo ou Modificado. Em algumas implementações, uma falha de leitura da CPU a um bloco da cache que é exclusivo na cache de outro processador é satisfeita de forma mais rápida do que uma falha a um bloco de memória. Isso porque as caches são menores e, portanto, mais rápidas do que a memória principal. Reciprocamente, em algumas implementações, as falhas satisfeitas pela memória são mais rápidas do que aquelas satisfeitas pelas caches, porque as caches geralmente são otimizadas para "frente" ou referências da CPU, em vez de para "trás" ou acessos por snooping. Para o multiprocessador da Figura 1, considere a execução de uma sequência de operações em uma única CPU, onde

- Acertos de leitura e escrita de CPU não geram ciclos de stall.
- Falhas de leitura e escrita de CPU geram ciclos de stall $N_{memória}$ e N_{cache}, se satisfeitas pela memória e pela cache, respectivamente.
- Acertos de escrita de CPU que geram uma invalidação incorrem em $N_{invalidação}$ de ciclos de stall.
- Um write-back de um bloco, seja devido a um conflito, seja devido à solicitação de outro processador a um bloco exclusivo, incorre em $N_{write\text{-}back}$ ciclos de stall adicionais.

Considere duas implementações com as diferentes características de desempenho resumidas na Tabela 1. Considere a sequência de operações a seguir, considerando o estado da cache inicial na Figura 1. Para simplificar, considere que a segunda operação começa depois que a primeira termina (embora estejam em processadores diferentes):

P1: read 110
P3: read 110

Para a implementação 1, a primeira leitura gera 50 ciclos de stall, pois a leitura é satisfeita pela cache de P0. P1 aguarda por 40 ciclos enquanto espera o bloco, e P0 por 10 ciclos enquanto escreve o bloco de volta à memória em resposta à solicitação de P1. Assim, a segunda leitura por

P3 gera 100 ciclos de stall, pois sua falha é resolvida pela memória. Então, essa sequência gera um total de 150 ciclos de stall. Para as sequências de operações a seguir, quantos ciclos de stall são gerados por implementação? **Para simplificar, considere que cada letra abaixo utiliza a configuração inicial mostrada na Figura 1, ou seja, para cada item deve-se considerar a configuração da memória/cache sem ser modificada pelo item anterior**.

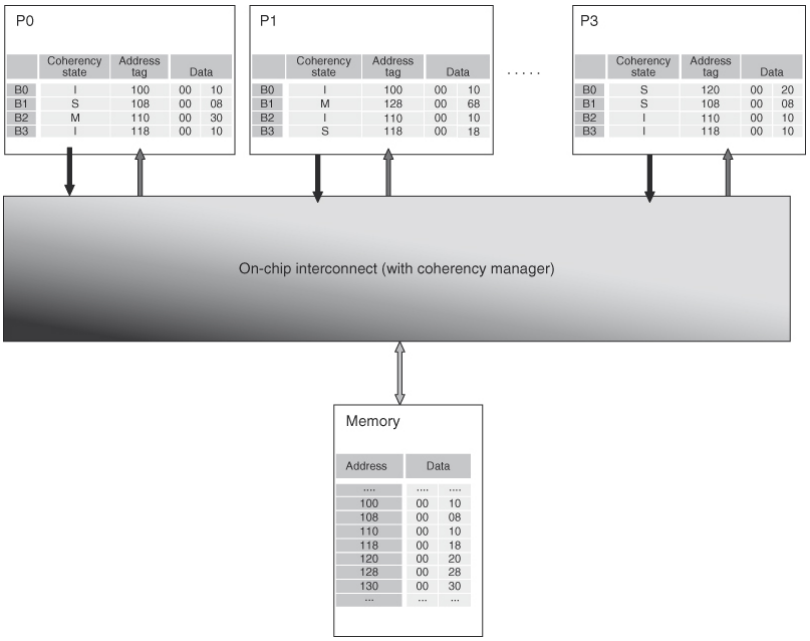

Figura 1

Tabela 1

Parâmetro	Implementação 1	Implementação 2
$N_{memória}$	100	100
N_{cache}	40	130
$N_{invalidação}$	15	15
$N_{write-back}$	10	10

a) P0: read 120
P0: read 128
P0: read 130

b) P0: read 100
P0: write 108 <-- 48
P0: write 130 <--78

c) P1: read 120
P1: read 128
P1: read 130

d) P1: read 100
P1: write 108 < -- 48
P1: write 130 <-- 78

Exercício 2

Os protocolos de diretório são mais escaláveis do que os protocolos snooping, pois enviam mensagens explícitas de solicitação e invalidação para os nós que possuem cópias de um bloco, enquanto os protocolos de snooping enviam todas as solicitações e invalidações por broadcast a todos os nós. Considere o sistema de 16 processadores ilustrado na Figura 2 e suponha que todas as caches não mostradas possuem blocos inválidos. Para cada uma das sequências a seguir, identifique quais nós recebem cada solicitação e invalidação.

a) P0,0: write 100 <-- 80
b) P0,0: write 108 <- - 88
c) P0,0: write 118< -- 90
d) P0,1: write 128 <-- 98

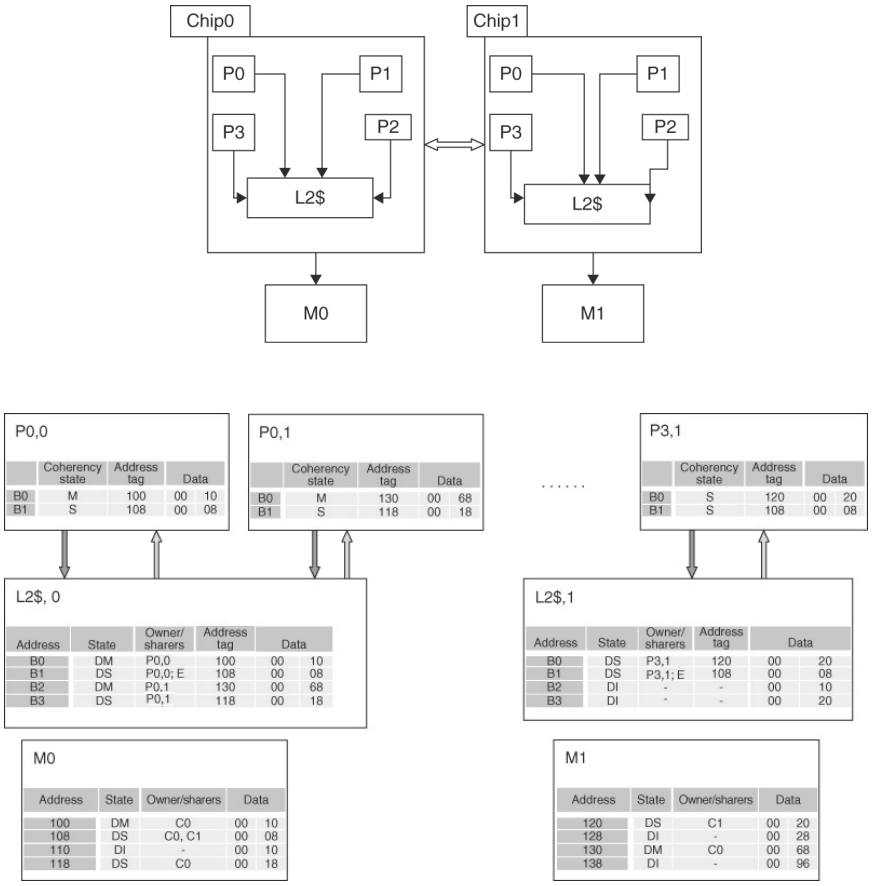

Figura 2

Exercício 3

O *spin lock* é o mecanismo de sincronismo mais simples possível na maioria das máquinas comerciais de memória compartilhada. Esse *spin lock* conta com a primitiva de troca (*exchange*) para carregar (*load*) atomicamente o valor antigo e armazenar (*store*) um novo valor. A rotina de *lock* (bloqueio) realiza a operação de troca (*exchange*) repetidamente até que encontre o *lock unlocked* (ou seja, o valor retornado é 0).

```
        DADDUI    R2, R0, #1
lockit: EXCH   R2, 0(R1)
        BNEZ   R2, lockit
```

O desbloqueio (*unlocking*) de um *spin lock* simplesmente exige um armazenamento do valor 0.

```
unlock: SW   R0, 0 (R1)
```

Suponha que os processadores P0, P1 e P3 estejam tentando adquirir um bloqueio (*lock*) no endereço 0x100 (ou seja, registrador R1). Considere o conteúdo da cache da Figura 1 e os parâmetros de temporização da implementação 1 da Tabela 1. Para simplificar, considere que as seções críticas utilizam 1000 ciclos.

Usando o *spin lock* simples, determine aproximadamente quantos ciclos de stall da memória cada processador incorre antes de adquirir o *lock* (bloqueio).

Exercício 4

Como discutido na seção 5.6, o modelo de consistência de memória fornece uma especificação de como o sistema de memória vai aparecer para o programador. Considere o seguinte fragmento de código, onde os valores iniciais são A= Flag = C = 0 ("0" em todas as variáveis)

P1	P2
A = 2000	while (Flag ==1) {;}
Flag = 1	C = A

a) No final do segmento de código, qual valor você esperaria para C?
b) Um sistema com uma rede de interconexão de uso geral, um protocolo de coerência de cache baseado em diretório, e suporte para *loads* sem bloqueio gera um resultado onde C é 0. Descreve um cenário onde esse resultado seja possível.
c) Se você quisesse tornar o sistema *sequencialmente consistente*, que restrições principais precisaria impor?

Exercício 5

Dada a natureza interativa dos WSCs, quais são alguns dos desafios enfrentados para reduzir consideravelmente o uso de energia?