Coerência de Cache

Márcia A. Santos
RA - 47748
marcia.santos@gmail.com

Resumo
Este artigo apresenta o problema de coerência de cache considerando sistemas uniprocessados e multiprocessados. São apresentados alguns dos protocolos de software e de hardware utilizados para manter a coerência da cache em sistemas multiprocessados.

Categorias e Descritores do Assunto
C0 [**Geral**]: Arquitetura de computadores

Palavras-chave
cache, *coerência*, *protocolos*.

1. INTRODUÇÃO
Existem sistemas que possuem um único processador. Tais sistemas podem fazer uso de uma ou mais caches para armazenamento de dados consultados. Para tais sistemas, a manutenção dos dados na cache pode ser facilmente realizada através da invalidação dos dados da cache, uma vez que somente um processador irá acessar ou alterar estes dados. Manter uma cache coerente é garantir que os dados armazenados estejam sincronizados com as atualizações realizadas ao longo do processamento das aplicações pelo processador. Dados podem ser alterados, removidos ou adicionados durante um acesso à cache.

Em sistemas multiprocessados, que atendem maiores demandas de processamento, cada processador pode ter sua própria cache, chamada cache privada. O uso de cache privada em sistemas multiprocessados pode gerar problemas relacionados à coerência das caches, pois podem existir os mesmos dados copiados em diferentes caches e estes dados podem ser atualizados de forma diferente pelos processadores. [1]

2. O PROBLEMA DE COERÊNCIA DE CACHE
O mecanismo de cache representa a idéia da memória virtual em que os dados armazenados ficam mais próximos do processador do que da memória principal. O objetivo principal da cache é prover os dados ao processador de uma forma mais rápida do que a memória principal.[4]

A cache em uniprocessadores possui um design mais simples do que em multiprocessadores, uma vez que os dados mantidos na cache são acessados por um único processador e somente uma cópia desse dado existirá na cache. Multiprocessadores usam várias caches, uma para cada processador (figura 1). Em um design de multiprocessadores com várias caches, múltiplas cópias dos mesmos dados podem existir em caches diferentes. Essa existência de cópias do mesmo dado é que gera o problema de coerência de cache (figura 2) [3].

Um sistema será coerente se todas as leituras por qualquer processador retornam o valor produzido pela última operação de escrita, sem importar qual processador realizou a escrita.

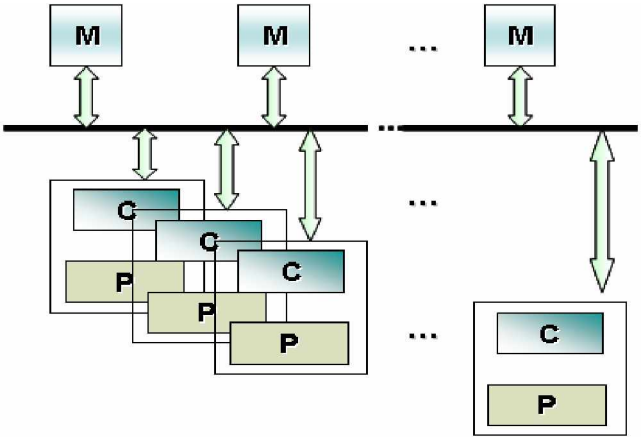

Figura 1. Cache em multiprocessadores

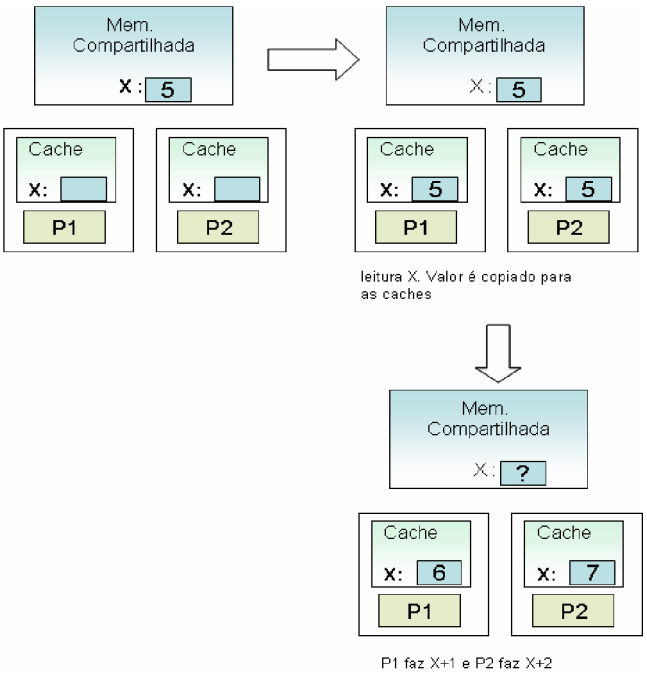

Figura 2. O problema de coerência de cache

3. COMO RESOLVER O PROBLEMA DE COERÊNCIA DE CACHE

Existem vários esquemas criados para resolver o problema de coerência da cache, alguns deles baseados em software e outros em hardware. Esquemas mais simples se baseiam na limitação do tipo de dado que pode ir para a cache, permitindo que apenas dados para leitura ou mesmo dados não-compartilhados (privados ao processador) possam ser armazenados na cache.

No caso dos esquemas baseados em software, delega-se ao compilador ou ao sistema operacional a responsabilidade de garantir a coerência. A vantagem dos esquemas baseados em software é evitar que um outro hardware seja necessário para realizar o tratamento do problema de coerência.

No caso dos esquemas baseados em hardware, as inconsistências são identificadas pelo hardware durante a execução, deixando assim que o tratamento seja transparente ao compilador ou ao sistema operacional. A coerência por hardware é a mais utilizada. Os esquemas de coerência são denominados *protocolos de coerência de cache.*

4. ESQUEMAS E PROTOCOLOS DE COERÊNCIA DE CACHE

4.1 Esquemas de coerência baseados em software

Esquemas de coerência de cache baseados em software são mais baratos e, na maioria das vezes, menos complexos do que os baseados em hardware. Porém podem demandar mais suporte por parte do programador.

Alguns esquemas são implementados como parte do compilador e, portanto, rodam em tempo de compilação. Outros são parte do sistema operacional e, portanto, rodam em tempo de execução. Também existem esquemas que operam em ambos, compilador e sistema operacional. Os esquemas que rodam em tempo de compilação são classificados como estáticos e os que rodam em tempo de execução são classificados como dinâmicos.

Alguns exemplos de esquemas de coerência de cache baseados em software são apresentados na figura 3

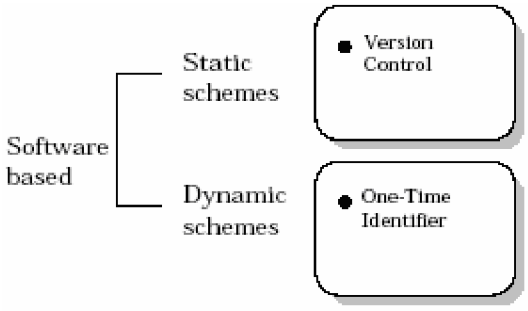

Figura 3. Esquemas de coerência baseados em software

Para um melhor entendimento e exemplificação, o funcionamento dos esquemas *One-Time Identifier* e *Version Control* serão descritos a seguir:

One-Time Identifier

No esquema *One-time Identifier*, cada página compartilhada possui um identificador único associado à página, chamado *one-time identifier*. Um campo de identificador é adicionado para cada linha da cache e para cada entrada da TLB. A cada atualização da TLB um novo valor de identificador é colocado no campo de identificador.

Quando uma linha é acessada pela primeira vez, os dados são copiados da memória principal e o valor do identificador da entrada correspondente à TLB é copiado para o campo de identificador da entrada correspondente à cache no registro de identificadores. Todos os acessos posteriores irão comparar o valor do identificador *one-time* da entrada de identificadores da TLB com o valor do identificador da entrada da cache no registro. Caso os valores sejam iguais, tem-se um *hit*. Caso contrário, tem-se um *miss*. O processador invalida a entrada da TLB quando sai da região crítica que protege o dado que está nesta entrada. Os valores de identificadores são copiados novamente para o registros de identificadores quando o próximo processador acessa os dados compartilhados. Os acessos que ocorrerem à entrada antiga da cache serão tratados como miss pois os valores dos identificadores não serão iguais.

Version Control

Este esquema utiliza versionamento como forma de garantir a coerência. Cada nova escrita de uma dado compartilhado gera uma nova versão do conteúdo. Os processadores incrementam o número da versão corrente a cada escrita que ocorra. Na cache, cada linha também possui um número de versão que é atribuído a cada alocação da cache. Para todo acesso ao dado compartilhado, os números de versão na cache e o corrente do processador são comparados. Caso o número da versão na cache seja menor, ocorre um *miss*, caso contrário, ocorre um *hit*.

4.2 Protocolos de coerência baseados em hardware

Embora sejam mais complexos, os protocolos de coerência baseados em hardware são utilizados principalmente em sistemas multiprocessadores comerciais. Tais protocolos lidam com a coerência em tempo de execução, sendo assim considerados dinâmicos.

Os protocolos podem ser classificados em Snoopy e Directory de acordo com a forma utilizada para informar os diversos processadores de que houve uma escrita de um dado, e que a cópia local pode estar desatualizada.

Alguns exemplos de protocolos de coerência de cache baseados em hardware são apresentados na figura 4.

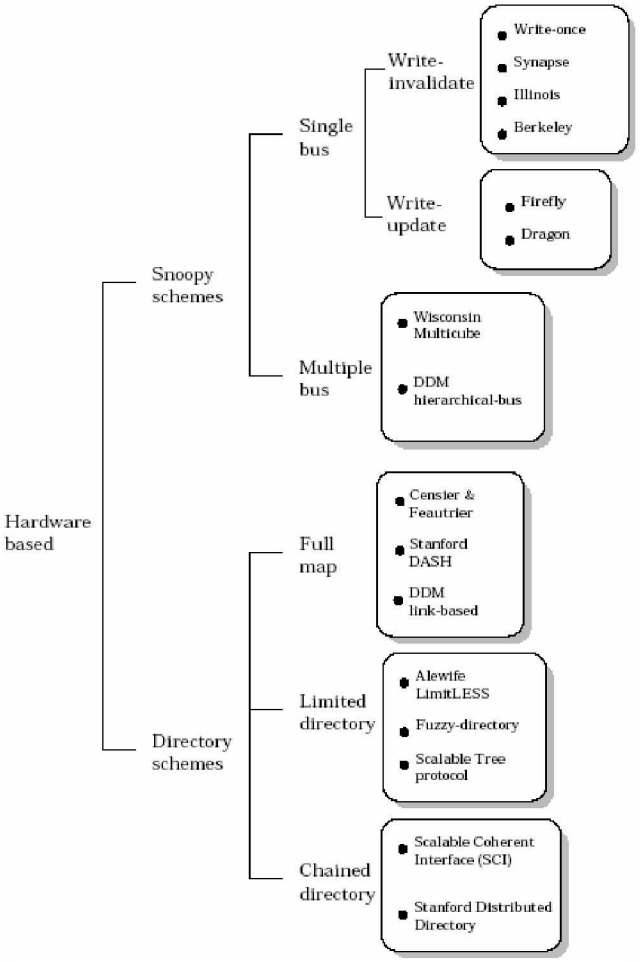

Figura 4. Protocolos de coerência baseados em hardware

Para um melhor entendimento e exemplificação, o funcionamento dos protocolos Snoopy: *Single bus com política de Write Invalidate* e *Multiple bus; e Directory: Full Map e Limited Directory* serão descritos a seguir:

- Protocolos *Snoopy*

Estes protocolos fazem uso da distribuição no tratamento do problema e coerência de cache.

Seu procedimento é baseado em ações dos controladores de caches locais e suas informações sobre o estado dos dados na cache. Utiliza-se broadcast para comunicar às demais caches as ações executadas sobre os dados compartilhados.

Como os protocolos *Snoopy* são baseados em broadcast, são mais indicados a multiprocessadores baseados em barramento, pois o barramento irá favorecer o broadcast.

Um problema detectado nos protocolos Snoopy faz com que eles sejam substituídos por outros protocolos, tais como os protocolos Directory: barramentos não são escaláveis quando se tem um número muito grande (dezenas) de processadores.[4]

Protocolos *Snoopy* podem ser do tipo *Single bus* e *Multiple bus*. Os *Single bus* podem usar políticas de *Write-invalidate* ou *Write-update*. A seguir descreveremos um exemplo relacionado a cada tipo de protocolo Snoopy:

Single bus – Write-invalidate - Synapse

A política de *Write-invalidate* permite apenas um escritor e vários leitores. Uma escrita em um dado compartilhado gera uma invalidação de todas as cópias desse dado antes que a escrita se inicie. Já a política *Write-update* atualiza todas as cópias do dado compartilhado antes de realizar a escrita.

Como um protocolo que utiliza a política Write-invalidate, Synapse faz uso de três estados: *Invalid*, *Valid* e *Dirty*. Qualquer cache que possua uma cópia em estado *Dirty* é a "dona" do bloco. Se não houver nenhum Dirty, a memória é a "dona".

Para situações de *miss* e *hit*, o Synapse possui o seguinte comportamento [5]:

- *Read Hit*: não há problemas de coerência
- *Read Miss*: o bloco é copiado para a cache e a CPU muda o estado da linha na cache para *Valid*.
- *Write Hit:* o barramento notifica todos os processadores que a linha é *Invalid*. A linha na cache é marcada como *Dirty* e todos os outros processadores marcam suas linhas correspondentes como *Invalid*.
- *Write Miss*: idêntico ao caso anterior

Multiple bus – Wisconsin Multicube

A arquitetura para o *Wisconsin* Multicube é baseada em uma família de interconexões mais gerais. Essa topologia é chamada de multicube. São x^k processadores com ligações N para N com os barramento: cada barramento conecta x processadores e cada processador está conectado com k barramentos, onde k é a dimensão do *multicube*. Um *Wisconsin multicube* é uma rede bidimensional.

O *multicube* consiste de cache de dois níveis: cache do processador e *snooping* cache. A consistência entre os dois níveis é mantida através da estratégia de write-through. Uma linha estará sempre em um dos estados globais: *unmodified* ou *modified*. Existem também alguns modos locais: *local, modified e invalid*. O protocolo possui quatro tipo de transações: leitura, leitura-alteração, allocate e *write-back*.

- Protocolos *Directory*

De uma maneira geral, estes protocolos fazem uso de um controlador central para manter a coerência e também guardam a informação de localização das cópias de dados. É utilizado um **diretório**, geralmente na memória principal, que armazena o estado global do conteúdo das várias caches. A cada requisição da cache local, o controlador central faz uma análise do diretório buscando qual a cache que possui a cópia do dado e libera a tranferência dos dados da memória principal para as caches. Este controlador realiza a atualização do estado da informação. O controlador central recebe as informações sobre tarefas que possam afetar o estado dos dados.

Um problema detectado nos protocolos *Directory* é a necessidade por espaço que tende a aumentar à medida que o número de processadores aumenta.

Directory – Full Map

Este protocolo tem sido a base para muitos outros designs que surgem. Seu funcionamento se dá através da concatenação de um vetor de bits para cada bloco de memória. Esse vetor de bits possui um bit por cache. Cada bit indica a presença ou ausência do bloco na cache correspondente. Este método possui eficiência com relação ao tempo.

Directory – Limited Directory

Com o objetivo de resolver o problema do tamanho dos diretórios, esse protocolo limita o crescimento do diretório em um fator constante de crescimento. Para isso, restringe-se o número de cópias simultâneas de bloco dados para a cache.

Quando uma cache requer a cópia de um dado para escrita, o módulo de memória invalida as demais cópias existentes nas outras caches. É utilizado um mecanismo de substituição de ponteiros. Um ponteiro para uma cache é substituído por outro que aponta para outra cache que possui o dado atualizado.

5. CONCLUSÃO

Existem vários esquemas e protocolos utilizados para resolver o problema de coerência de cache, como apresentado nas figuras 3 e 4. Porém, esses esquemas e protocolos devem ser adaptados a cada tipo de arquitetura para que a vantagem da utilização de caches não seja perdida por causa da utilização de esquemas e protocolos que degradem o desempenho ou mesmo a latência da cache.

6. REFERÊNCIAS

[1] Stenston, Peter. A survey of chache coherence schemes for multiprocessors. *Lund University, Suécia.*.

[2] Baukus, Kai and van der Meyden, Ron. A knowledge based analysis of cache coherence. School of Computer Science and Engineering, -University of New South Wales, and National ICT Australia

[3] Chaiken Lars, David. Cache Coherence Protocols for Large-Scale Multiprocessor. S.M. Thesis, Massachussetts Institute of Technology, 1990

[4] Raina, Sanjay. Virtual Shared Memory: A Survey of Techniques and Systems. Dept. of Computer Science University of Bristol Bristol BS8 1TR, U.K

[5] Geiss, Lenise Cristina. Coerência de Cache em Multiprocessadores. Instituto de Informática Universidade Federal do Rio Grande do Sul, 1998