ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES II

Coerência de Cache em Multiprocessadores

Índice

Coerência de Cache (*Cache Coherence*)

- **Diversas Arquiteturas de máquinas paralelas atuais são construídas com processadores produzidos em larga escala**
- **Objetivo**
  - **Reduzir os custos de projeto**
- **Fato**
  - **máquinas paralelas com múltiplos processadores incorporam *caches* em suas arquiteturas**
- **Problema**
  - **Presença de *caches* privadas em multiprocessadores necessariamente introduz problemas de coerência de *cache***
- **Descrição do problema**
  - **Múltiplas cópias de mesma posição de memória podem existir em diferentes *caches***
  - **Cada processador atualiza sua cópia local, não se preocupando com a existência de outras cópias em outros processadores**
  - **Cópias de mesmo endereço de memória poderão possuir valores diferentes → Caracteriza uma situação de inconsistência de dados**

Coerência de Cache

- **Alternativa para eliminar o problema**
  - Não permitir que dados compartilhados para operações de escrita sejam colocados nas *caches* do sistema
  - Somente instruções e dados privados são *cacheable*
  - Dados compartilhados são considerados *noncacheable*
- **Operacionalização**
  - Compilador é responsável por colocar etiqueta (*tag*) nos dados
  - Mecanismos de cópia de dados entre níveis de memória sabem quais dados devem ser repassados diretamente (*sem passar pelas caches*) ao processador
- **Coerência de cache em multicomputadores**
  - Esse problema não ocorre → Cada nó possui hierarquia de memória inteira → Não existe espaço de endereçamento global compartilhado por todos
- **Definição de coerência de cache**
  - Uma arquitetura multiprocessada com *caches* privadas é coerente se e somente se uma leitura de uma posição *x* de memória efetuada por qualquer processador *i* retorne o valor mais recente desse endereço
  - Toda vez que uma escrita for efetuada por um processador *i* em um endereço de memória *x*, tem que ser garantido que todas as leituras subseqüentes de *x*, independentemente do processador, forneçam o novo conteúdo de *x*

O Problema da Inconsistência de Dados

- **Necessidade de consistência de dados**
  - **É imprescindível que a implementação monoprocessada e a multiprocessada gerem o mesmo resultado para um programa**
- **Problema**
  - **Quando dois processadores compartilham mesma memória através de *caches* diferentes, existe risco de uma posição de memória não possuir valor mais recente**
- **Causas mais comuns desse tipo de inconsistência de dados**
  - **Inconsistência no compartilhamento de dados**
  - **Inconsistência na migração de processos**
  - **Inconsistência de E/S**

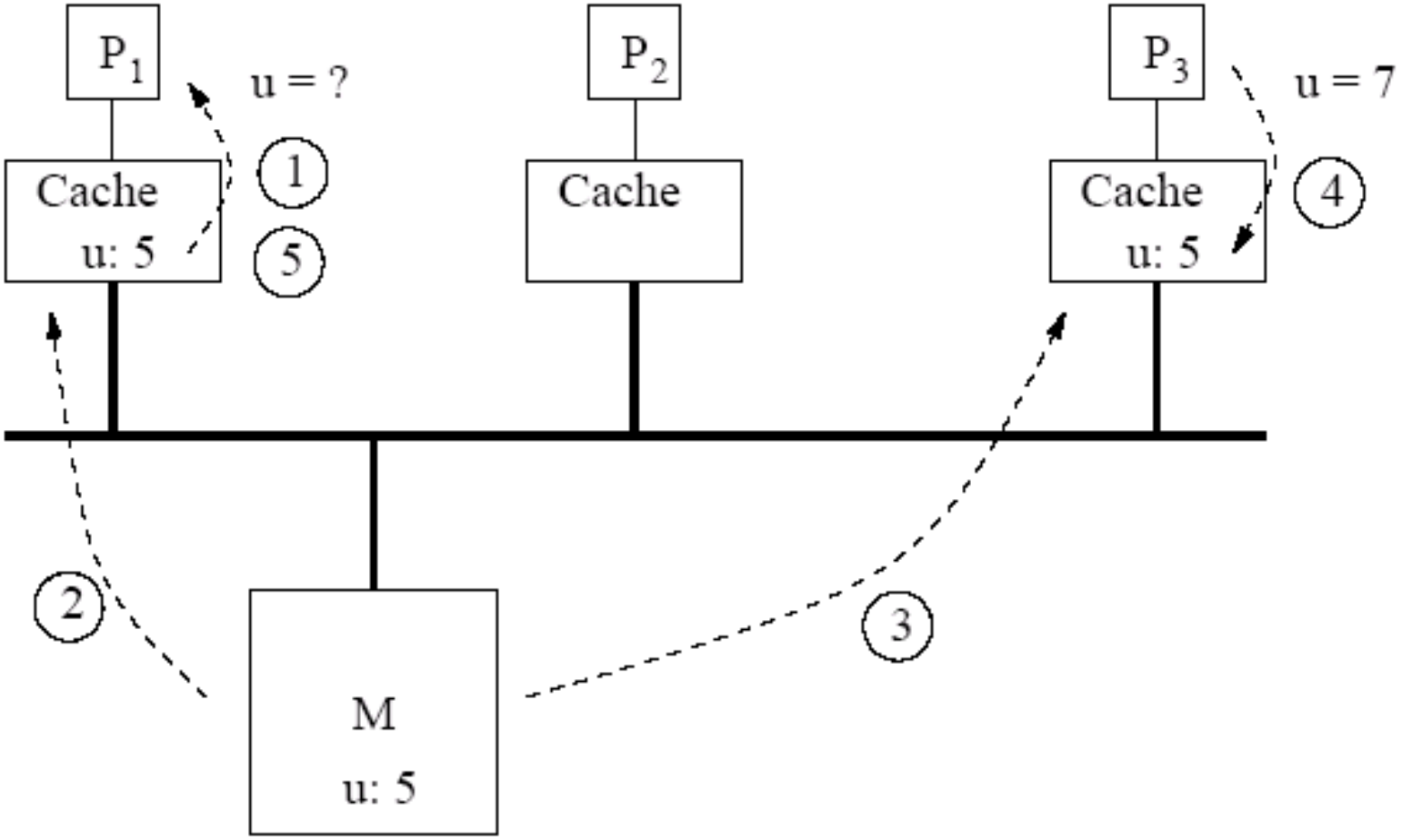
Índice

Inconsistência no Compartilhamento de Dados

- **Descrição do problema**
  - P1, P2 e P3 são processadores que possuem *caches* privadas
  - Estes estão interconectados, através de barramento, a uma memória principal compartilhada

1. P1 tenta ler *u* de sua *cache*
2. Como *u* não está presente, dado é lido da memória principal e copiado para sua *cache*
3. P3 faz o mesmo, gerando também cópia de *u* em sua *cache*
4. P3 efetua escrita em *u* e altera conteúdo da posição de 5 para 7
5. Quando P1 efetuar uma leitura de *u* novamente, ele receberá o conteúdo da posição *u* que se encontra em sua *cache* → 5, e não o valor mais recente de *u*, que é 7

Inconsistência no Compartilhamento de Dados



Inconsistência no Compartilhamento de Dados e Política de Atualização de Cache

- Esse problema ocorre independentemente da política de atualização da memória principal
- *write-through* (escrever através)
  - A cada escrita na *cache*, a posição da memória principal é atualizada também
  - A alteração de P3 teria sido repassada também à memória principal
  - Não impediria P1 de ler o valor menos recente de *u* de sua *cache*
- *write-back* (escrever de volta)
  - A alteração de P3 teria apenas marcado como sujo o bloco de *u* na *cache* (através de seu *dirty-bit*)
  - A memória principal não seria atualizada imediatamente
  - Somente em uma eventual substituição desse bloco na *cache* é que a memória principal seria atualizada
  - Se P2 efetuasse uma leitura em *u* nesse meio-tempo, copiaria para sua *cache* o valor menos recente de *u* (valor 5)

Índice

Inconsistência na Migração de Processos

- **Descrição do problema**
  - Novo processo é escalonado devido a operação de E/S
  - Não existe garantia que este vá retornar sua execução no mesmo processador de uma máquina multiprocessada
  - Escalonador associará processo a processador livre, segundo sua política de escalonamento, quando a operação de E/S estiver concluída
  - Muitas vezes, o processo volta a executar em outro processador perdendo as informações de sua antiga *cache*
- **Pergunta**
  - Se fosse garantido executar processo no mesmo processador, este problema seria resolvido. Porquê?

Índice

Inconsistência de Cache devido à Operação de E/S

- **Introdução ao problema**
  - **Problemas de inconsistência de dados também podem ocorrer durante operações de E/S (*outros componentes que não os processadores*) com DMA à memória principal**
  - **Estas operações não se preocupam se os dados estão sendo compartilhados por vários processadores, com cópias em diferentes *caches***
- **Descrição**
  - **Supondo atualização de valor *x* escrito na memória principal por *write-through***
    - **Quando controladora de E/S carregar dado *x'* na memória principal, ocorre inconsistência de dados entre *x'* e as cópias nas *caches* dos processadores que possuem antigo valor *x***
  - **Supondo atualização de valor *x* com política de *write-back***
    - **Quando valor *x'* é lido da memória principal em operação de E/S, pode ocorrer inconsistência se valor atual das cópias de memória nas *caches* ainda não tenha sido substituído e, portanto, ainda não tenha sido copiado para a memória principal**

Índice

Estratégias de Coerência de *Cache*

- Problema da coerência de *cache* resume-se
  - Existir simultaneamente múltiplas cópias de um dado, o qual pode ser alterado sem que se faça algo em relação às outras cópias
- Estratégias básicas para tratar coerência de cache
  - Escrita na *cache* resulta atualização de outras cópias desse dado nas demais *caches* (*write-update*)
  - Escrita na *cache* resulta invalidação de outras cópias desse dado nas demais *caches* (*write-invalidate*)
- Vantagens e Desvantagens
  - Invalidação tem custo menor, mas maior latência de acesso caso as cópias invalidadas sejam novamente acessadas → Necessita buscar da memória principal
  - Atualização tem custo mais alto, especialmente em máquinas com muitos processadores (muitas cópias potenciais de mesmo endereço), mas menor latência → Novo acesso a cópias é resolvido em nível de *cache*

Estratégias de Coerência de *Cache*

- **Questão**
  - **Qual das estratégias resulta melhor desempenho para sistema como um todo?**
- **Resposta**
  - **Está diretamente ligado ao compartilhamento da carga de trabalho. Ou seja, o compartilhamento dos programas que executam na máquina**
    - **Se processadores que estavam usando as cópias antes de serem atualizadas fizerem novos acessos a esses dados, vale a pena o custo das atualizações**
    - **Se processadores não utilizarem esses dados novamente, o tráfego gerado pelas atualizações penalizou o sistema. Neste caso, a invalidação eliminaria as cópias antigas e acabaria com uma situação de compartilhamento aparente**

Estratégias de Coerência de *Cache*

- ***Pack rat:*** **Fenômeno que onera a estratégia de atualizações**
  - **Conseqüência da migração de processos executada pelo SO**
  - **Quando processo perde processador por causa de uma operação de E/S, ele pode voltar a executar em outro processador**
  - **Não poder mais acessar dados de sua antiga *cache***
  - **Tem que buscá-los novamente da memória principal**
  - **Os dados antigos continuam sendo atualizados em vão até que sejam eliminados da *cache* antiga por alguma política de substituição de blocos**
  - **(*write-update* ou *write-invalidate*)**
- **É muito fácil construir casos nos quais uma determinada estratégia (*write-update* ou *write-invalidate*) vai desempenhar melhor que outra**
- **Alternativa**
  - **Implementação destas em hardware e que o sistema permita a troca entre atualização e invalidação dinamicamente em tempo de execução**
- **Momento da troca pode ser**
  - **Determinado pelo programador através de uma chamada de sistema**
  - **Seguir uma certa probabilidade alterável na configuração do sistema**
  - **Depender do padrão de acesso observado em tempo de execução**

Índice