Mechanisms for Store-wait-free Multiprocessors

Bruno Cardoso Lopes - RA023241

26 de abril de 2009

Thomas F. Wenisch, Anastasia Ailamaki, Babak Falsafi, and Andreas Moshovos. **Mechanisms for store-wait-free multiprocessors**. In ISCA '07: *Proceedings of the 34th annual international symposium on Computer architecture*, pages 266–277, New York, NY, USA, 2007. ACM.

1 Resumo

Sistemas multiprocessados com memória compartilhada possuem o desempenho afetado pela latência de *store misses*, provocados por *stalls*. Esses *stalls* ocorrem devido a capacidade insuficiente de armazenamento nos *store buffers* durante *store bursts* e também por causa dos limites de ordenação impostos, que obrigam *stalls* na execução.

A ocorrência de *stalls* varia conforme os modelos de consistência de memória : *Strongly-ordered*, *Relaxed load-to-store order* e *All order relaxed*, cujas implementações utilizadas são (na mesma ordem): *SC*, *TSO* e *RMO*. Com o aumento de relaxamento do modelo de consistência, ocorre a diminuição do número de *stalls* e o aumento da complexidade do modelo de programação.

Maneiras convencionais de diminuir tais *stalls* são elaboradas de forma a obter as melhores soluções gerais; aquelas que contemplem um bom balanceamento entre modelos de consistência de memória e complexidade de programação.

A abordagem de multiprocessadores *store-wait-free* tem o objetivo idealista de proporcionar o modelo mais simples de programação e ao mesmo tempo nunca ter *stalls* resultantes de *store misses*. Dois mecanismos podem ser utilizados em conjunto para se obter multiprocessadores *store-wait-free*: *Scalabe Store Buffer* (*SSB*) e *Atomic Sequence Order* (*ASO*).

O *SSB* resolve o problema de capacidade do *Store Buffer* eliminando a procura associativa feita por *loads* em *TSO*s, provendo os dados procurados de maneira especulativa na cache L1 (onde a visão sobre dados é local) e mantendo dados globais e commitados na cache L2. Antes de sofrerem *commit* na L2 os *writes* são guardados na *TSOB*, uma *FIFO* responsável por ordenar os *stores*.

O *ASO* fornece restrições na ordenação fornecendo atomicidade por blocos de código (ao contrário de abordagens que utilizam pequenas operações atômicas), possibilitando implementação eficiente de mecanismos de *rollbacks* baseados em *checkpoints*.

Quando ocorre uma violação o *hardware* deve voltar na execução (*rollback*) e recuperar o estado correto do processador desde o começo da seqüência violada. Assim, é preciso que o hardware guarde todas as seqüências de *writes* e as libere atomicamente em um *commit* ou as eliminem sob violação.

No caso em que algumas, mas não todas seqüências vigentes, violem e um *rollback* seja necessário, a cache L1 descarta todas as linhas escritas de maneira especulativa e essas seqüências são apagadas da *TSOB*.

A implementação da *SSB* em *hardware* é feita com a adição de *bits* de validade por palavra na L1, que permitem a leitura privada de dados enquanto um *store* está acontecendo. Um *buffer* circular também é adicionado ao *hardware*, constituindo a *TSOB*.

Para implementar a ASO são adicionados *bits* de especulação de leitura e escrita em cada linha da L1. Uma tabela de seqüências atômicas (AST) é mantida e contem a gravação das posições da *TSOB* para cada seqüência atômica (assim é possível saber quais *stores* devem ser descartados do *TSOB* caso ocorra uma violação).

A validação desta solução é realizada através do simulador *FLEXUS*, configurado para se aproximar aos recursos de um *Intel Core* 2. O *FLEXUS* suporta os modelos de consistência de memória SC, TSO e RMO.

O resultado mostra que a utilização da SSB e ASO provê um desempenho melhor que o de modelos de consistência totalmente relaxados (RMO) mantendo a simplicidade de modelos de programação mais convencionais e rígidos. Isto é provado uma vez que a utilização de SSB e ASO com os modelos SC e TSO dão desempenhos melhor que RMO em todas aplicações testadas.