UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

CIÊNCIA DA COMPUTAÇÃO

ARQ. E ORG. DE COMPUTADORES II

RELATÓRIO TÉCNICO

Simulador Protocolo MESI em uma aplicação

Prof: Sandra Cossul

DISCENTES

Nome	RA
Gustavo Giozeppe Bulgarelli	129658

Disciplina 6895/01
25/08/2024

SUMÁRIO

INTRODUÇÃO

O seguinte relatório técnico apresenta o desenvolvimento de um simulador de Protocolo MESI (*Modified*, *Exclusive*, *Shared*, *Invalid*) em uma aplicação, Sistema de Gerenciamento de uma Biblioteca. Sobre essa aplicação, diferentes processadores simulam usuários que podem acessar e modificar informações de livros armazenados na base de dados da biblioteca, permitindo que possa se estudar o comportamento de uma memória cache e do Protocolo MESI, demonstrando como o mesmo otimiza o desempenho em sistemas de memória cache. Além disso, também são discutidas as decisões de projeto para a implementação da aplicação.

OBJETIVOS

O objetivo deste trabalho é desenvolver um simulador de cache utilizando o protocolo MESI em um sistema de gerenciamento de uma biblioteca, com o propósito de estudar e demonstrar o funcionamento da coerência de cache em sistemas multiprocessadores. Por esta simulação, busca-se compreender como diferentes estados de cache impactam a leitura e a escrita de dados em ambientes onde múltiplos processadores acessam simultaneamente os mesmos recursos de memória.

FUNCIONAMENTO DA MEMÓRIA CACHE

A memória cache é um tipo de memória de alta velocidade localizada entre o processador e a memória principal, projetada para armazenar temporariamente dados que são acessados frequentemente a fim de reduzir o tempo de acesso a esses dados.

No simulador, a memória cache é utilizada para acessar os dados armazenados na RAM. Cada processador simulado possui sua memória cache correspondente, implementada como uma estrutura de dados com um número fixo de posições (‘`CACHE_SIZE`’). Tais posições armazenam blocos de dados (com tamanho fixo ‘`BLOCK_SIZE`’) da RAM, juntamente com uma *tag* que indica o bloco de memória acessado.

O funcionamento da memória cache é administrada por um mapeamento direto, onde cada posição da cache pode conter dados de um bloco específico da memória principal. Quando um processador realiza uma operação de leitura ou escrita, é verificado caso o dado solicitado já está presente na cache ou não, cache *hit* e cache *miss* respectivamente. Caso ocorra ocorra um cache miss, o bloco solicitado da memória é carregado para a cache. A posição a qual o bloco será armazenado é determinada pelo algoritmo de substituição FIFO (*First in, first out*), implementado na função ‘`get_FIFO_index`’, que também lida com a política de escrita *write-back*.

FUNCIONAMENTO DO PROTOCOLO MESI

O protocolo MESI gerencia a coerência de dados entre as caches dos processadores, sendo um protocolo de monitoramento (*snoopy*) com abordagem *write invalidate*. É baseado em estados, para cada linha de cache, que representam condições específicas dos dados em relação à memória principal e a outras caches.

No simulador, o protocolo é definido como um tipo (‘`typedef enum {MODIFIED, EXCLUSIVE, SHARED, INVALID} MesiState`’) e implementado em conjunto a estrutura de dados das posições da cache, fazendo com que cada linha possa estar em um dos quatro estados:

‘`MODIFIED`’: Uma linha da cache é modificada, quando o seu valor foi alterado na cache e ainda permanece inalterado na RAM.

- Quando a escrita de um dado é solicitada, a linha, esteja presente na cache anteriormente ou não, seu estado é alterado para ‘`MODIFIED`’.

‘`EXCLUSIVE`’: Uma linha da cache é exclusiva quando somente um processador está acessando a posição da memória RAM.

- Quando a leitura de um dado é solicitada, a linha é marcada como ‘`EXCLUSIVE`’ caso esse dado não esteja presente em outra cache.

‘`SHARED`’: Uma linha da cache é compartilhada se mais de um processador está acessando a posição da memória RAM.

- Quando a leitura de um dado é solicitada, a linha é marcada como ‘`SHARED`’ caso esse dado esteja presente em outra cache.

‘INVALID’: Uma linha da cache é inválida se outro processador alterou o valor da posição da memória RAM.

- Quando a escrita de um dado é solicitada por outro cache e esse dado está presente nesse nesse cache, seu estado é alterado para ‘INVALID’ (a linha que escreveu teria se tornado ‘MODIFIED’).

FUNCIONAMENTO DA APLICAÇÃO

A aplicação de um sistema de gerenciamento de uma biblioteca ilustra a interação entre múltiplos usuários do sistema, acessando e/ou editando dados de livros.

Para maior facilidade no manuseio do código, os dados dos livros estão salvos em um arquivo `.txt` na forma: ‘`<id>|<título>|<autor>|<ano de lançamento>`’ e cada bloco representa uma linha no arquivo. Esses dados são inicializados junto ao simulador e são salvos na RAM, mantendo o arquivo intacto.

A interface do usuário é implementada em um loop de entrada, que oferece ao usuário três operações: leitura, escrita ou sair. Tanto a operação de leitura quanto a de escrita solicitam ao usuário qual processador será usado, simulando cada usuário do sistema, como também o bloco de dados, através da tag, a ser lido ou escrito. Para a escrita também solicita o bloco a ser escrito naquela tag.

A interface também exibe os resultados das operações, como também o estado da cache após cada operação, permitindo que o usuário observe a interação entre as caches em si e entre a memória principal.

DECISÕES DE PROJETO PARA A IMPLEMENTAÇÃO

Estruturas de Dados Utilizadas

A estrutura de dados tanto da cache quanto da RAM foram implementadas por através de ‘`typedef struct`’.

A `struct` ‘`CachePos`’ representa uma posição da cache, que contém a tag da RAM acessada, o bloco de dados da memória e o estado MESI. É nessa `struct` que a maioria das interações acontecem. A `struct` ‘`Cache`’ representa a cache, contendo um *array* de posições (‘`CachePos`’). A `struct` ‘`Processor`’ representa cada processador contendo uma cache (‘`Cache`’). Essa estrutura é definida com o prefixo ‘`extern`’ para que possa ser acessada diretamente, como também é definida como um array, de tamanho fixo ‘`PROC_SIZE`’ para representar os múltiplos processadores na simulação.

A `struct` ‘`RAM`’ representa um bloco de dados da RAM, que contém o bloco de dados, como também o endereço (‘`line`’) desse bloco. Essa estrutura é acessada quando é solicitada uma leitura ou escrita de dados. A `struct` ‘`Memory`’ representa a memória principal e contém um array de blocos (‘`RAM`’) com tamanho fixo ‘`RAM_SIZE`’. Essa estrutura é definida com o prefixo ‘`extern`’ para que possa ser acessada diretamente.

Gerenciamento do Protocolo MESI

O gerenciamento é realizado diretamente nas funções de leitura e escrita, tanto para o processador que solicita, quanto para qualquer outro que for necessário (i.e., invalidar os dados acessados caso um dos processadores os modifique).

A Aplicação (Questões Relativas à Implementação)

A aplicação simula um sistema de gerenciamento de uma biblioteca, com os dados presentes na RAM sendo informações dos livros. É armazenada inicialmente em um arquivo que é acessado junto ao simulador para passar os dados para a RAM. A interface da aplicação é um menu baseado em texto, que permite que o usuário selecione as operações, como também veja os resultados de cada operação.

CONCLUSÃO

O desenvolvimento deste simulador de cache com o Protocolo MESI em um sistema de gerenciamento de uma biblioteca demonstrou a importância quanto à coerência e otimização de cache em sistemas multiprocessadores.

A simulação evidenciou como a correta gestão dos estados de cache é essencial para reduzir conflitos e aprimorar o acesso à memória. Este trabalho reforça a relevância do protocolo MESI em arquiteturas modernas, oferecendo uma base sólida para futuros estudos em gerenciamento de memória e otimização em paralelismo.

REFERÊNCIAS BIBLIOGRÁFICAS:

Stallings, William. **Computer Organization and Architecture: Design for Performance**. PEARSON/PRENTICE HALL, 11ed, 2021.

Patterson, David A. and Hennessy, John L. **Computer Organization And Design: The Hardware/Software Interface**. MORGAN KAUFMANN, 4th EDITION, 2012.

WATSON, Ian. **Cache coherence in shared-memory architectures**. 2018. Disponível em: https://www.cs.utexas.edu/~pingali/CS377P/2018sp/lectures/mesi.pdf. Acesso em: 24 ago. 2024.