

Organização de Computadores

Aula 18

Memória cache

Parte 3

Memória Cache

terceira parte

1. Mecanismos de fetch e escrita

2. Substituição de linhas

3. Hierarquia de caches

4. Caches de dados e instruções

5. Medindo desempenho

Introdução

- **Motivação**

- **Memórias cache trabalham com os princípios de localidade temporal e espacial.**
- **Assim, as memórias cache usam blocos de dados, para que durante uma busca de dado, o bloco todo seja trazido, aumentando assim a localidade espacial.**
- **Uma vez carregado o bloco na memória cache, ele tende a ser reutilizado segundo o princípio de localidade temporal, dessa maneira, é interessante deixar os dados que serão novamente usados, o maior tempo possível na memória cache.**

Mecanismo de Fetch

1. Mecanismos de Fetch e Escrita

Estratégias para fetch de palavras ou linhas da memória principal

- Fetch por demanda
- Prefetch
- Fetch por demanda
 - Fetch da linha quando ocorre *miss*
 - Estratégia mais simples, não exige hardware adicional
- Prefetch
 - Fetch da linha antes que ela seja necessária
 - Prefetch da linha $i+1$ quando a linha i é inicialmente referenciada
 - Prefetch da linha $i+1$ quando ocorre *miss* da linha i

Política de Escrita

Técnicas durante Write Hit

Política de Escrita - Introdução

- Leitura na cache não afeta conteúdo => não há discrepância entre cache e memória principal
- Escrita na cache: cópias da palavra na cache e na memória principal podem ter valores diferentes
- Valores deveriam ficar iguais e manter coerência para:
 - Acessos de E/S feitos através da memória principal
 - Acessos à memória principal por múltiplos processadores
- Tempo médio de acesso à cache é aumentado pela necessidade de atualizações da memória principal
- Mecanismos de **coerência** de escrita
 - *Write-Through*
 - *Write-Back*

Mecanismo *Write-Through*

- ***Write-through***: cada escrita na cache é repetida imediatamente na memória principal
- Escrita adicional na memória principal aumenta tempo médio de acesso à cache
- Estatisticamente apenas 5% a 34% dos acessos à memória são escritas

Mecanismo *Write-Through*

Tempo médio de acesso à cache T_{ma} é dado por

$$T_{ma} = T_c + (1 - h) T_b + w (T_m - T_c)$$

onde

T_c = tempo de acesso à cache h = hit ratio

T_b = tempo de leitura de uma linha da memória principal

T_m = tempo de acesso a uma palavra da memória principal

w = probabilidade de que acesso seja de escrita

supondo

$T_c = 1 \text{ ns}$, $T_b = T_m = 10 \text{ ns}$, $h = 0.98$, $w = 0.2$

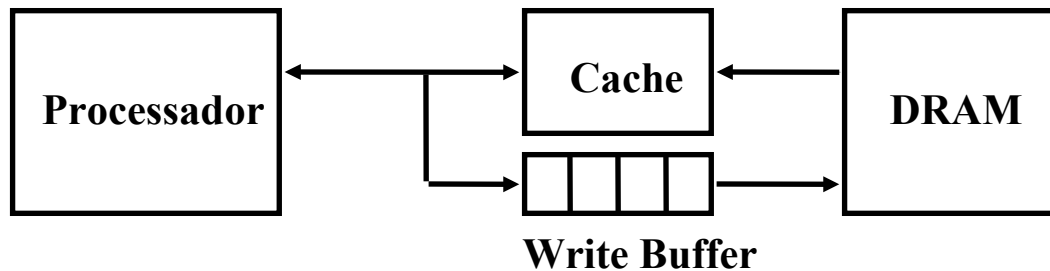
$T_{ma} = 3,0 \text{ ns}$, sendo:

1,0 ns tempo da cache

0,2 ns devido a misses

1,8 ns devido ao write-through !!!

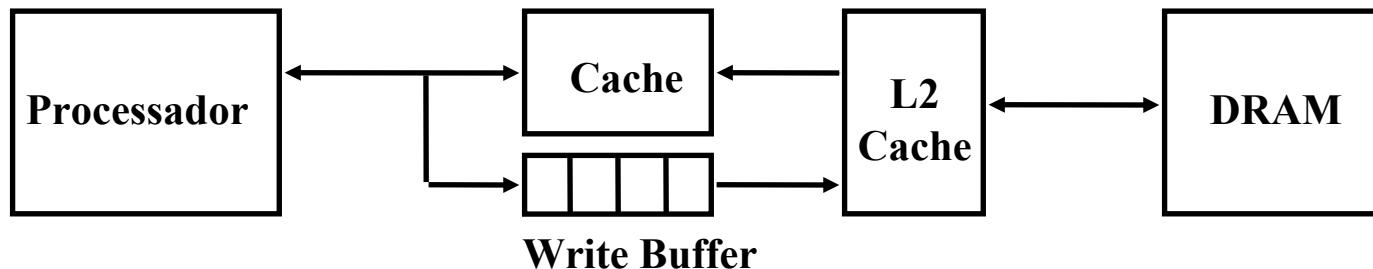
Write-Through com Write Buffer



- **Write Buffer é necessário entre cache e memória principal**
 - **processador**: escreve dados na cache e no *write buffer*
 - **controlador de memória**: escreve conteúdo do buffer na memória
- **Write buffer é uma FIFO**
 - típico número de posições ≥ 4
 - funciona bem se: frequência de escritas $\ll 1 / \text{ciclo escrita DRAM}$
- **Problema**
 - frequência de escritas $> 1 / \text{ciclo escrita DRAM}$
 - saturação do Write Buffer

Saturação do *Write Buffer*

- **Frequência de escritas $> 1 / \text{ciclo escrita DRAM}$**
 - Isso ocorre porque tempo de ciclo da CPU é rápido demais e/ou ocorrem muitas instruções *store* em sequência
 - Se esta condição existe por um longo período de tempo:
 - **Write-Buffer terá overflow, não importa quão grande ele seja**
- **Solução para saturação do Write Buffer**
 - usar cache com *write-back*
 - instalar uma cache de segundo nível (L2)



Mecanismo *Write-Back*

- ***Write-back***: linha da cache só é escrita de volta na memória principal quando precisa ser substituída
- **Estratégia mais simples**: escrita é feita mesmo que linha não tenha sido alterada

$$T_{ma} = T_c + (1 - h) T_b + (1 - h) T_b$$

onde o segundo termo $(1-h) T_b$ é devido à escrita

- **Estratégia alternativa**: só escrever de volta se linha foi alterada exige um bit de tag para indicar modificações na linha

$$T_{ma} = T_c + (1 - h) T_b + w_b (1 - h) T_b$$

onde w_b = probabilidade de que linha tenha sido alterada

Políticas de Escrita

Técnicas de Write Miss

Técnicas de Write Miss

Write Allocate

- O bloco de endereço é carregado para a memória cache na ocorrência de um "write miss", seguindo-se uma ação de "write hit".
- O "Write Allocate" é usado com frequência em caches de "Write-back",
- Write Allocate + Write-Back = Pode ajudar a reduzir tempos de acesso durante gravações sucessivas a um determinado bloco.

No Write Allocate

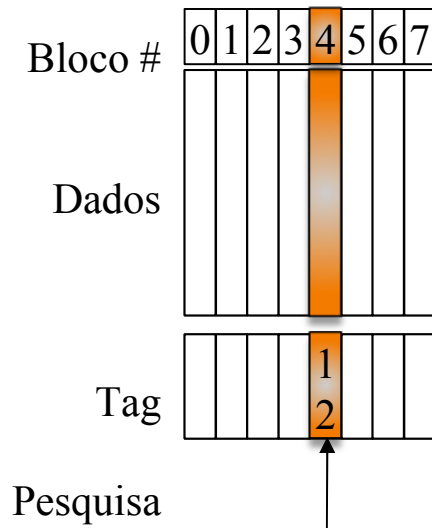
- O bloco de endereço é diretamente modificado na memória, o endereço do "write miss" não é carregado na cache.
- O "No Write Allocate" é usado frequentemente em caches de "Write Through"
- No Write Allocate + Write Through = Não compensa carregar o bloco para a memória que causou o write miss, pois todas as atividades de write serão enviadas para os níveis abaixo.

2. Substituição de linhas

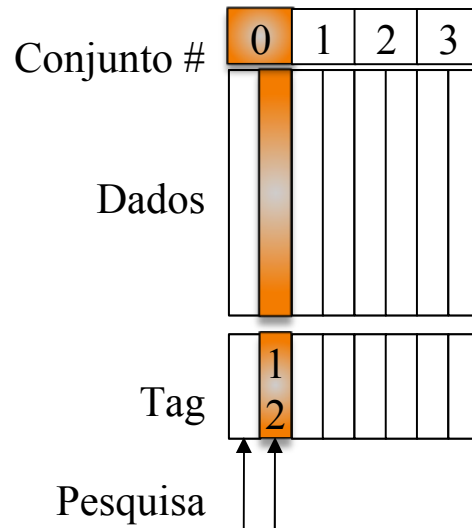
2. Substituição de linhas

- Quando ocorre um *miss*, uma nova linha precisa ser trazida da memória principal
 - Cache com mapeamento direto: não precisa escolher qual linha da cache será substituída
 - Cache completamente associativa: pode-se escolher qualquer uma das linhas
 - Cache conjunto-associativa: deve-se escolher uma linha dentro de um conjunto fixado pelo índice
-
- Algoritmo de substituição precisa ser implementado em hardware

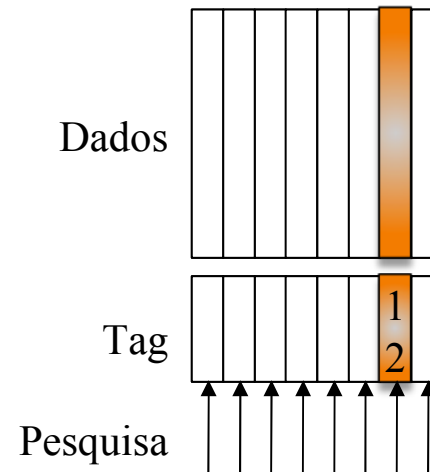
(a) Diretamente mapeada



(b) Associativa por Conjunto



(c) Totalmente Associativa



Algoritmos de substituição

Substituição randômica

- Escolha de uma linha ao acaso para ser substituída
- Exemplo de implementação em hardware: contador
 - Contador é incrementado a cada ciclo do relógio
 - Quando substituição é necessária, escolhe-se linha cujo endereço é igual ao valor atual do contador
 - Cache completamente associativa: contador de n bits se cache tem 2^n linhas
 - Cache conjunto-associativa: contador de 2 bits numa cache com associatividade = 4

First-In First-Out (FIFO)

- Remover a linha que está há mais tempo na cache
- Implementação evidente: fila de endereços de linha

Algoritmos de substituição

LRU – *Least Recently Used*

- Linha a ser substituída é aquela que há mais tempo não é referenciada
- Implementação mais simples em hardware: contador associado a cada linha
 - Quando *hit* ocorre, contador da linha correspondente é zerado
 - Demais contadores são incrementados
 - Linha com contador com valor mais alto é a substituída
- **Outras implementações**
 - Pilha de registradores
 - Matriz de referência
 - Métodos aproximativos

3. Hierarquia de caches

3. Hierarquia de caches

- **Caches integradas dentro de um processador têm limitação de tamanho**
- ***Miss penalty* na cache é muito grande, pela diferença entre os tempos de acesso da cache e da memória principal**
- **Solução: caches em dois ou três níveis**
 - **cache integrada (L1, de primeiro nível) de tamanho pequeno, p.ex. 8 Kbytes, e tempo de acesso menor**
 - **cache secundária (L2) tem tamanho maior, p.ex. 256 Kbytes, e tempo de acesso maior**
- **Processadores recentes têm cache de terceiro nível (L3)**
 - **cache L3 fora do chip do processador, cache L2 dentro**
- ***Misses* podem ocorrer em referências a qualquer nível de cache**
- **Transferências entre níveis de cache apresentam mesmos problemas e possíveis soluções já discutidos**

Reducing Cache Miss Rates

❑ Uso de múltiplos níveis de cache

- Com o avanço da tecnologia são possíveis grandes cache L1 e L2 dentro do mesmo *die* do processador.
- Normalmente uma cache L2 unificada (Inst. + Dados) e em alguns casos, até mesmo uma cache L3 unificada também.
- Exemplo, CPI_{ideal} de 2, 100 ciclos de miss penalty (memória principal), 36% load/stores, a 2% (4%) L1I\$ (D\$) miss rate, adicionando uma UL2\$ que tem 25 ciclos de miss penalty e 0.5% de miss rate

$$CPI_{stalls} = 2 + .02 \times 25 + .36 \times .04 \times 25 + .005 \times 100 + .36 \times .005 \times 100 = 3.54$$

(Comparado com 5.44 sem a cache L2\$)

Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are very different
 - Primary cache should focus on **minimizing hit time** in support of a shorter clock cycle
 - Smaller with smaller block sizes
 - Secondary cache(s) should focus on **reducing miss rate** to reduce the penalty of long main memory access times
 - Larger with larger block sizes
- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate
- For the L2 cache, hit time is less important than miss rate
 - The L2\$ hit time determines L1\$'s miss penalty
 - L2\$ local miss rate \gg than the global miss rate

Key Cache Design Parameters

| | L1 typical | L2 typical |
|-------------------------------|-------------|-----------------|
| Total size (blocks) | 250 to 2000 | 4000 to 250,000 |
| Total size (KB) | 16 to 64 | 500 to 8000 |
| Block size (B) | 32 to 64 | 32 to 128 |
| Miss penalty (clocks) | 10 to 25 | 100 to 1000 |
| Miss rates (global for L2) | 2% to 5% | 0.1% to 2% |

Two Machines' Cache Parameters

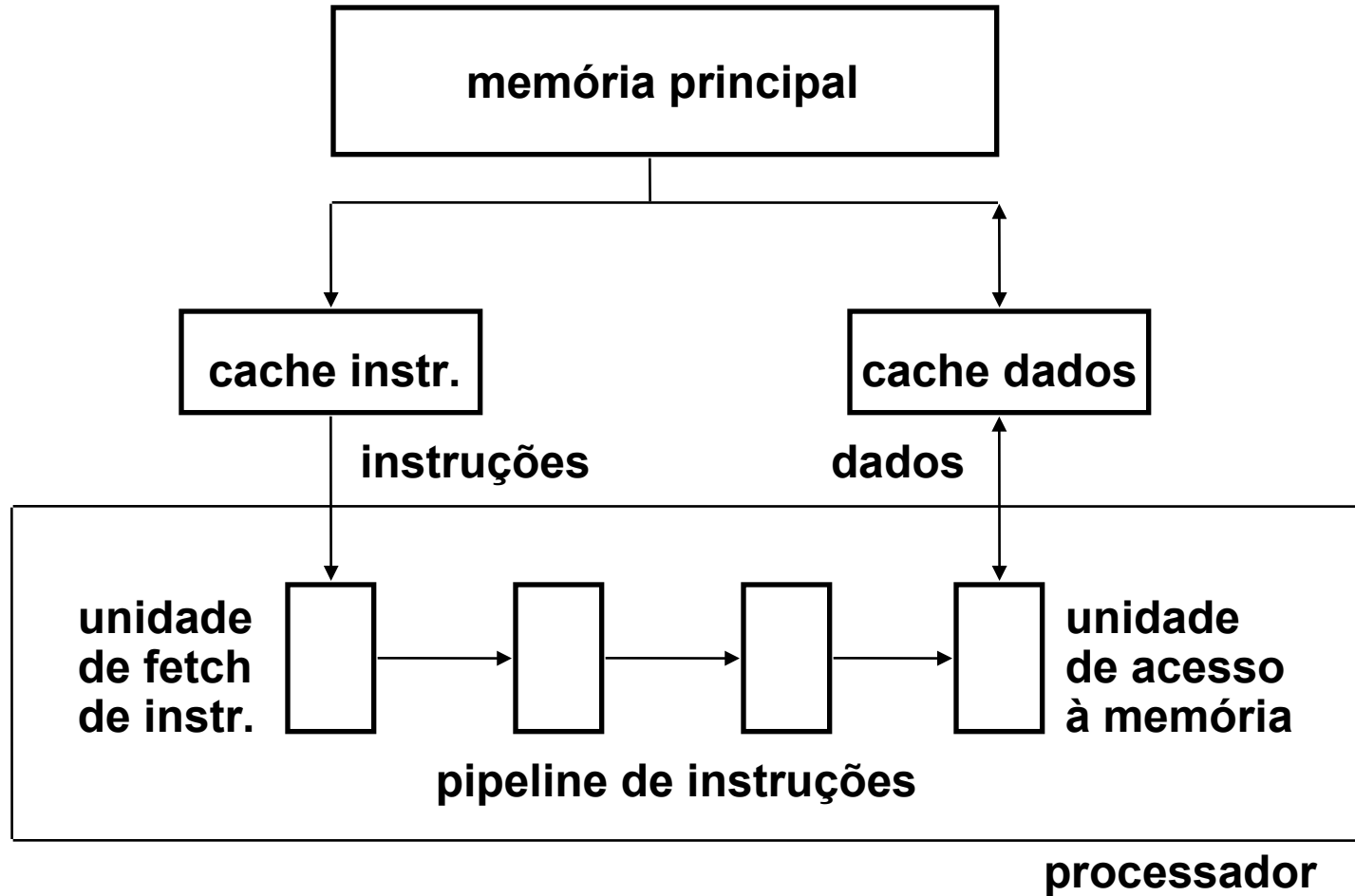
| | Intel P4 | AMD Opteron |
|------------------|--|------------------------------|
| L1 organization | Split I\$ and D\$ | Split I\$ and D\$ |
| L1 cache size | 8KB for D\$, 96KB for trace cache (~I\$) | 64KB for each of I\$ and D\$ |
| L1 block size | 64 bytes | 64 bytes |
| L1 associativity | 4-way set assoc. | 2-way set assoc. |
| L1 replacement | ~LRU | LRU |
| L1 write policy | write-through | write-back |
| L2 organization | Unified | Unified |
| L2 cache size | 512KB | 1024KB (1MB) |
| L2 block size | 128 bytes | 64 bytes |
| L2 associativity | 8-way set assoc. | 16-way set assoc. |
| L2 replacement | ~LRU | ~LRU |
| L2 write policy | write-back | write-back |

4. Caches de dados e instruções

4. Caches de dados e instruções

- **Dados e instruções: cache unificada x caches separadas**
- **Vantagens das caches separadas**
 - política de escrita só precisa ser aplicada à cache de dados
 - caminhos separados entre memória principal e cada cache, permitindo transferências simultâneas (p.ex. num pipeline)
 - estratégias diferentes para cada cache: tamanho total, tamanho de linha, organização
 - Conhecida como arquitetura de Harvard
- **Caches separadas são usadas na maioria dos processadores, no nível L1**
- **Caches unificadas nos níveis L2 e L3**

Caches de dados e instruções






Problemas com caches separadas

- **Código auto-modificável**
- **Instruções e dados colocados em posições próximas de memória =>**
duas cópias da mesma linha estarão nas duas caches
- **Necessidade de controle adicional**

5. Avaliando o desempenho

5. Medindo desempenho

- **Desempenho da cache depende ...**
 - da organização das cache
 - do mapeamento dos dados
 - do tamanho da cache e das linhas (blocos)
 - do algoritmo de substituição
 - dos mecanismos de escrita e de fetch
 - dos programas sendo executados
- **Métodos de obtenção de estimativas de desempenho**

| | | | |
|---------------------------------|--|---|---|
| – medida direta | (Caro) | (Preciso) | (Pouco genérico) |
| – simulação <i>trace-driven</i> |  |  |  |
| – modelagem matemática | (Barato) | (Estimado) | (Bastante Genérico) |

Medindo desempenho

- **Medida Direta**
 - Hardware de monitoração permite coleta das referências à memória
 - Registro é então utilizado na simulação
 - Vantagem: todas as referências são coletadas, mesmo aquelas executadas por trechos protegidos de código
- **Simulação *trace-driven***
 - Programas típicos são executados
 - Facilidade de trace do processador é usada para interromper após cada instrução e registrar endereços de memória gerados pela instrução
 - Registro é utilizado numa simulação
 - Exige-se simulação de grande n° de instruções (> 1 M)
- **Modelagem analítica**
 - Difícil modelagem matemática se o sistema for complexo
 - Bastante genérico, costuma indicar uma tendência para diversos sistemas de um mesmo tipo.

FIM