Universidade Federal do Rio Grande do Sul
Instituto de Informática

Organização de Computadores

Aula 18

Memória cache
terceira parte

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

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
  - p.ex: prefetch da linha i+1 quando a linha i é inicialmente referenciada
  - alternativa: prefetch da linha i+1 quando ocorre *miss* da linha i

Operações de escrita

- 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 em razão de:
  - 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 Tma é dado por

$$Tma = Tc + (1 - h)\,Tb + w\,(Tm - Tc)$$

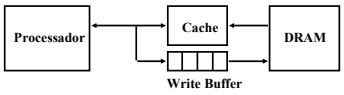
onde

Tc = tempo de acesso à cache h = *hit ratio*
Tb = tempo de leitura de uma linha da memória principal
Tm = tempo de acesso a uma palavra da memória principal
w = probabilidade de que acesso seja de escrita

supondo

Tc = 1 ns, Tb = Tm = 10 ns, h = 0.98, w = 0.2
⇨ Tma = 3,0 ns, sendo 0,2 ns devido a *misses* e 1,8 ns devido ao *write-through*

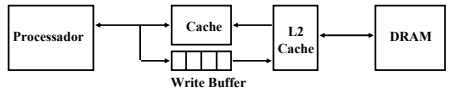
Write-Through com *Write Buffer*



- 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: freqüência de escritas << 1 / ciclo escrita DRAM
- problema
  - freqüência de escritas > 1 / ciclo escrita DRAM
  - saturação do Write Buffer

Saturação do *Write Buffer*

- freqüência de escritas > 1 / ciclo escrita DRAM
  - se esta condição existe por um longo período de tempo (porque tempo de ciclo da CPU é rápido demais e/ou ocorrem muitas instruções *store* em seqüência):
    - 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

  $Tma = Tc + (1 - h)\,Tb + (1 - h)\,Tb$

  onde o segundo termo (1–h) Tb é devido à escrita
- estratégia alternativa: só escrever de volta se linha foi alterada

  exige um bit de tag para indicar modificações na linha

  $Tma = Tc + (1 - h)\,Tb + w_b\,(1 - h)\,Tb$

  onde w_b = probabilidade de que linha tenha sido alterada

2. Substituição de linhas

- quando ocorre um *miss*, uma nova linha precisa ser trazida da memória principal para a cache
- 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

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

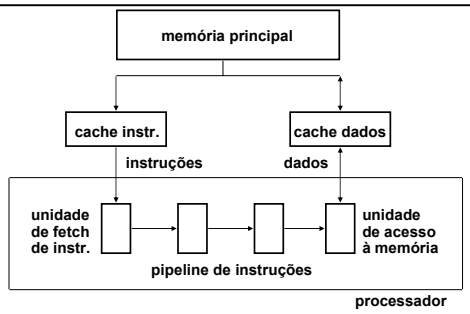
3. Hierarquia de caches

- 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) é 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

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
- 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. Medindo desempenho

- desempenho da cache depende ...
  - da organização (mapeamento)
  - do tamanho da cache e das linhas
  - do algoritmo de substituição
  - dos mecanismos de escrita e de fetch
  - dos programas sendo executados
- métodos de obtenção de estimativas de desempenho
  - simulação *trace-driven*
  - medida direta
  - modelagem matemática

Medindo desempenho

- 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 nº de instruções (> 1 M)
- 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

Exercício

- **Qual o CPI de uma maquina com pipeline, se o hit na cache de instruções é de 95%, e o da cache de dados é de 90%? 30% das instruções fazem acesso à memória, e destas, 30% são stores. A penalidade de acesso à memória principal é de 50 ciclos, e a política de escrita é de write-through sem buffer.**
- **Repita para o caso onde uma cache L2 foi adicionada. A penalidade para uma ida na L2 é de 5 ciclos. O hit-ratio na L2 é de 98% para instruções e 95% para dados.**
- **Repita ambos os exercícios assumindo que existe um write-buffer, e que somente em 10% das escritas deve-se efetivamente pagar a penalidade.**

Conclusão

- **Note como é boa a idéia do superescalar...**