Hierarquia de Memória - Parte 3 - Organização da Memória Cache

Arquitetura de Computadores

Emilio Francesquini
e.francesquini@ufabc.edu.br

2020.Q1

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

Disclaimer

UFABC

- Estes slides foram preparados para o curso de Arquitetura de Computadores na UFABC.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- O conteúdo destes slides foi baseado no conteúdo do livro *Computer Organization And Design: The Hardware/Software Interface*, 5th Edition.

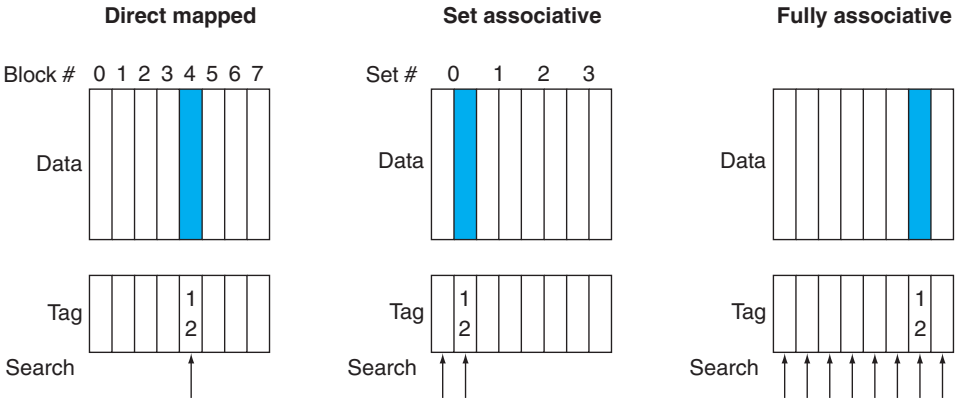
Associatividade

Caches associativas

UFABC

- Totalmente associativas (*fully associative*)
  - Permite que um bloco seja colocado em qualquer local da cache
  - Para garantir a eficiência, exige que todas as entradas da cache sejam verificadas simultaneamente
  - É necessário um comparador por entrada
- Associativa de *n* vias (*n-way associative*)
  - Divide a cache em conjuntos (*sets*)
  - Cada conjunto contém *n* entradas
  - O número do bloco indica qual é o set
    - (Nº do bloco) modulo (Nº de sets)
  - Todas as entradas de cada um dos sets é verificada simultaneamente
  - Precisa apenas *n* comparadores (mais barato)

Exemplo de caches associativas



Espectro da associatividade

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Exemplo de associatividade

UFABC

- Vamos comparar caches com 4 blocos
  - Estratégias de mapeamento direto, totalmente associativa e associativa com 2 vias
  - Sequência de acessos: 0, 8, 0, 6, 8

Exemplo: Mapeamento direto

UFABC

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

Exemplo: Associativa com 2 vias

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

Exemplo: Completamente Associativa

UFABC

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

- Não dá para fazer melhor!
- Apenas ocorrem falhas compulsórias.

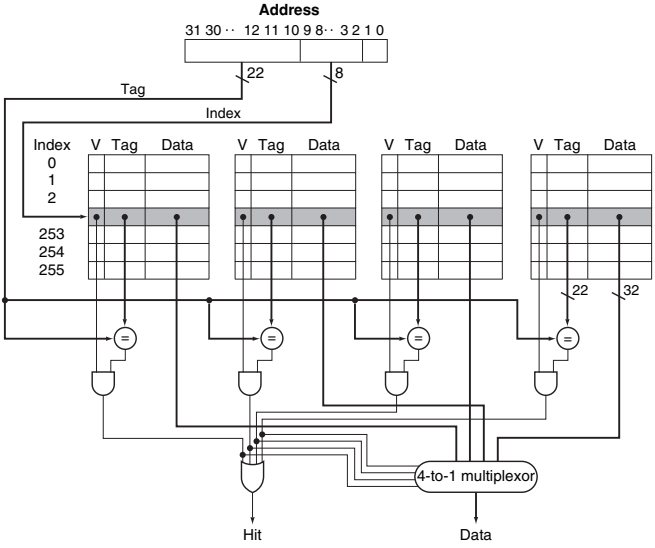
Até onde levar isso?

UFABC

- Maior associatividade $\rightarrow$ menor miss rate
  - Mas a melhora diminui rapidamente
- Em uma simulação com uma cache de dados de 64KB, blocos de 16 palavras usando o SPEC2000

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

Organização de Caches Associativas



Política de substituição

Política de substituição

UFABC

- A política de substituição (*replacement policy*) determina qual bloco da cache deverá dar lugar a outro quando é feita uma leitura de dados em um endereço que não está na cache
- A política para caches com mapeamento direto é única: o bloco pode ser colocado em uma única posição
- Em caches associativas
  - Preferencialmente utiliza-se uma entrada não válida, caso exista
  - Só então escolhe entre as outras entradas do set

UFABC

- Menos recentemente utilizada (*least-recently used*) - LRU
  - Escolhe substituir o bloco que está sem utilização pelo período mais longo de tempo
    - Simples de implementar em hardware para 2 vias, razoável para 4 vias, complicado além disto
- Aleatória
  - Escolhe um bloco aleatoriamente
  - Fornece aproximadamente o mesmo desempenho que LRU quando a associatividade é alta

Caches com múltiplos níveis

UFABC

- Cache primária, ou L1, ligada diretamente à CPU
  - Pequena, porém rápida
- Cache L2 trata misses da cache L1
  - Maior que L1, um pouco mais lenta, mas ainda bem mais rápida que a RAM
- A memória principal trata misses da cache L2
- Alguns sistemas incluem níveis adicionais, como L3 e L4

Exemplo de cache com múltiplos níveis

UFABC

- São dados
  - CPI base da CPU = 1, clock = 4 GHz
  - Taxa de misses/instrução = 2%
  - Tempo de acesso à memória principal = 100 ns
- Apenas com a cache L1
  - Miss penalty = 100ns / 0.25ns = 400 ciclos
  - CPI efetivo = 1 + 0.02 * 400 = 9

Exemplo de cache com múltiplos níveis (Cont.)

UFABC

- Agora com cache L2
  - Tempo de acesso 5ns
  - Taxa de misses = 0.5%
- Miss na L1 e Hit na L2
  - Penalty = 5ns / 0.25ns = 20 ciclos
- Miss na L1 e Miss na L2
  - Penalty extra = 400 ciclos
- CPI = 1 + 0.02 $\times$ 20 + 0.005 * 400 = 3.4
- Razão de desempenho = 9/3.4 = 2.6

Algumas considerações sobre caches com múltiplos níveis

UFABC

- Cache L1
  - Foco em respostas com o mínimo de tempo possível
- Cache L2
  - Foco em baixa taxa de misses para evitar acessos à memória principal
  - O tempo de hit tem um impacto menos importante quando comparado à L1
- Resulta em:
  - L1 é normalmente menor quando comparada com um sistema com um sistema com um único nível de cache
  - O tamanho do bloco da L1 é, em geral, menor do que o tamanho do bloco da L2

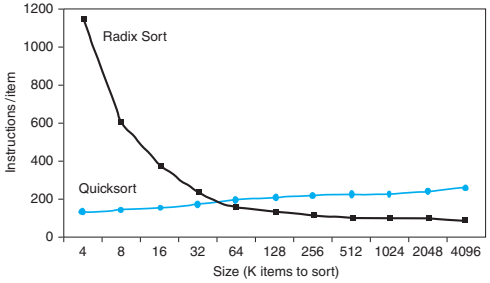
Interações da Cache com Hardware e Software

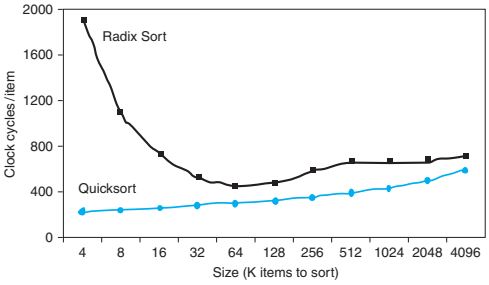
UFABC

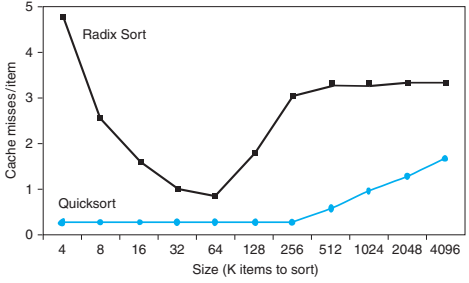
- CPUs OoO são capazes de executar instruções enquanto ocorre um cache miss
  - Stores pendentes ficam armazenados na unidade de load/store
  - Instruções com dependências ficam aguardando
  - Instruções independentes simplesmente continuam
- Efeito dos misses dependem do fluxo de dados do programa
  - Difícil fazer uma análise
  - Utiliza-se, em geral, simulações para este propósito

- Misses dependem do padrão de acesso à memória
  - Comportamento do algoritmo
  - Otimizações do compilador quanto ao acesso à memória

Exemplo: QuickSort *vs.* Radix Sort







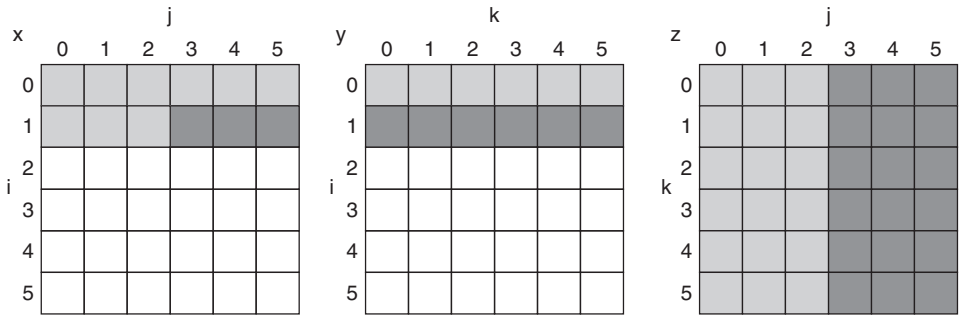
UFABC

- Objetivo principal é aproveitar o máximo os dados na cache antes que eles sejam substituídos
  - blocagem (*cache blocking*)
- Considere os laços internos da DGEMM[1]

```
1 for (int j = 0; j < n; ++j) {
2     double cij = C[i + j * n]; /* cij = C[i][j] */
3     for (int k = 0; k < n; k++)
4         cij += A[i+k*n] * B[k+j*n]; /* cij += A[i][k]*B[k][j] */
5     C[i+j*n] = cij; /* C[i][j] = cij */
6 }
```

[1] Double precision GEneral Matrix Multiplication

- C, A e = B com `N = 6` e `i = 1`

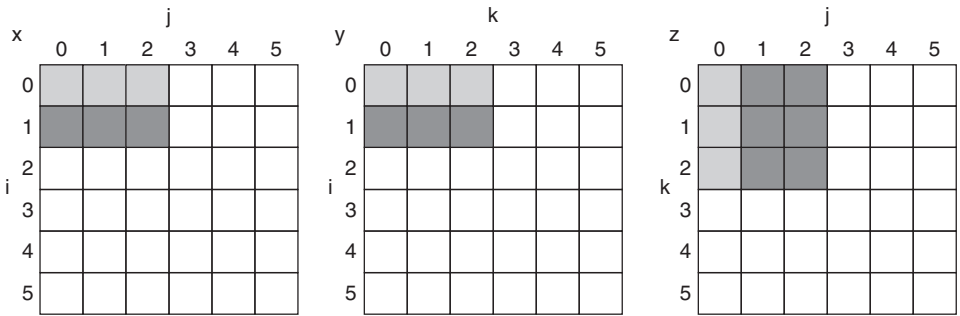


Versão blocada

UFABC

```
1  #define BLOCKSIZE 32
2  void do_block (int n, int si, int sj, int sk, double *A, double *B,
   ↪  double *C) {
3      for (int i = si; i < si+BLOCKSIZE; ++i)
4          for (int j = sj; j < sj+BLOCKSIZE; ++j) {
5              double cij = C[i+j*n];/* cij = C[i][j] */
6              for( int k = sk; k < sk+BLOCKSIZE; k++ )
7                  cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
8              C[i+j*n] = cij;/* C[i][j] = cij */
9          }
10 }
11 void dgemm (int n, double* A, double* B, double* C) {
12     for ( int sj = 0; sj < n; sj += BLOCKSIZE )
13         for ( int si = 0; si < n; si += BLOCKSIZE )
14             for ( int sk = 0; sk < n; sk += BLOCKSIZE )
15                 do_block(n, si, sj, sk, A, B, C);
16 }
```

- C, A, B, `BLOCKSIZE = 3`



Comparação DGEMM - Versão padrão *vs.* Versão blocada