Memórias cache: uma introdução

João Canas Ferreira

Dezembro de 2006

Contém figuras de "Computer Architecture: A Quantitative Approach", J. Hennessey & D. Patterson, 3ª. ed., MKP

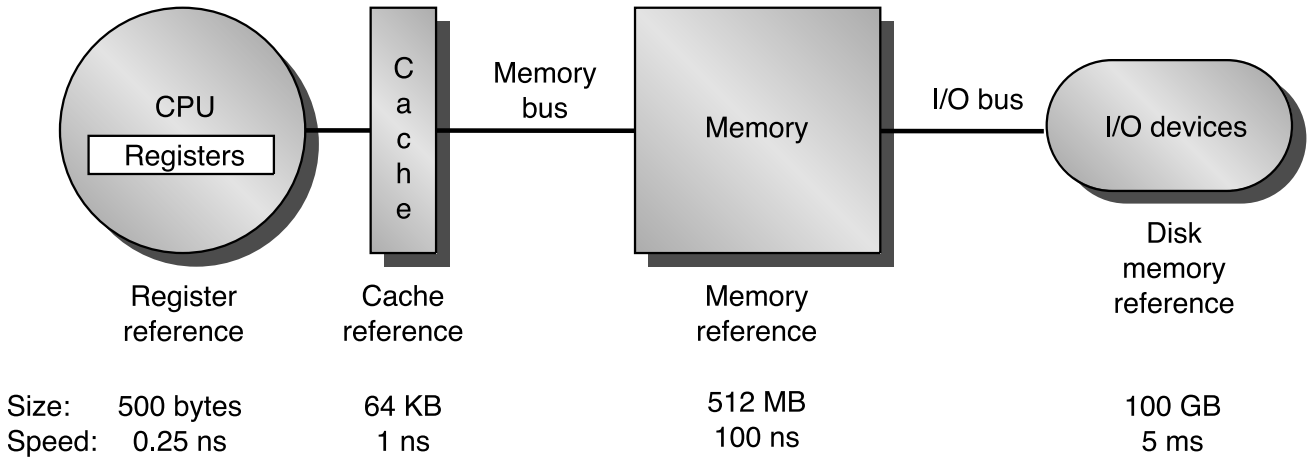
Assuntos

Hierarquias de memória: conceitos básicos

- **Princípio da proximidade:** Programas tendem a reutilizar os dados e as instruções usados recentemente. Existem 2 tipos de proximidade:
  1. *proximidade temporal*—elementos acedidos recentemente têm maior probabilidade de ser acedidos a seguir;
  2. *proximidade espacial*—elementos colocados em posições de memória próximas tendem a ser acedidos em instantes consecutivos.
- Como memória rápida é cara, a memória é geralmente organizada numa hierarquia de vários níveis: cada nível tem menor capacidade, é mais rápido e mais caro (€/bit) que o seguinte.
- Frequentemente cada nível está contido no seguinte (mas nem sempre).
- A hierarquia de memória também é responsável pela verificação de endereços (já que tem de mapear endereços nos níveis correctos).
- A importância da hierarquia de memória tem aumentado com os avanços de CPU.

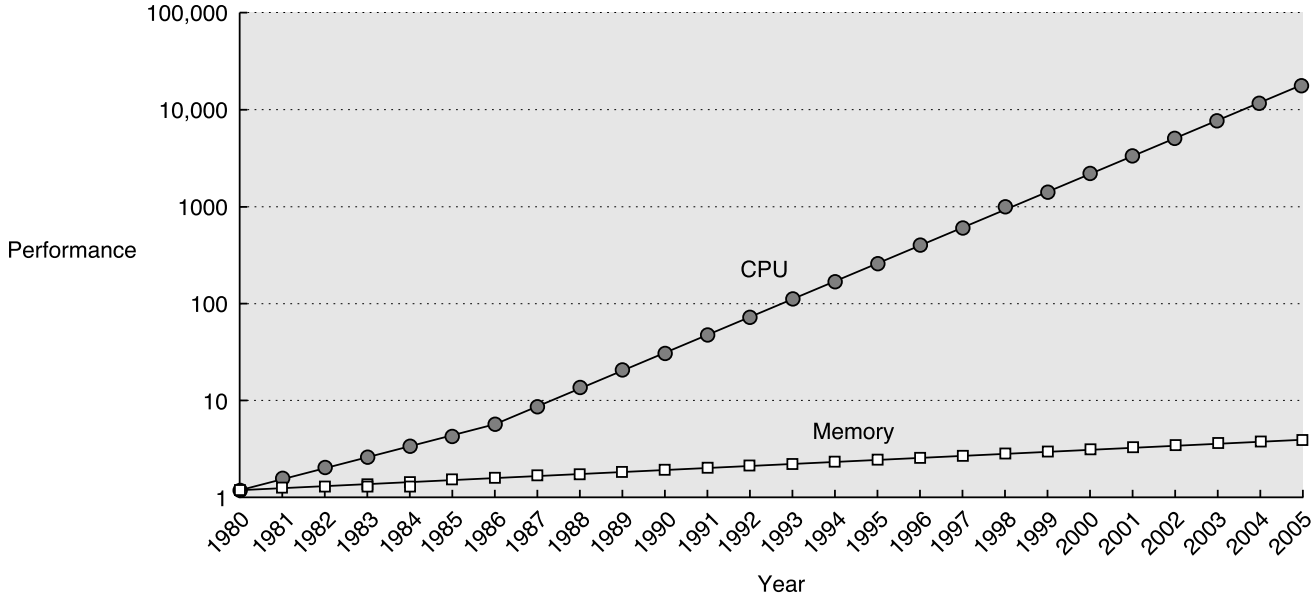
Uma hierarquia de memória simples

Exemplo:



Desempenho de CPU vs. memória

Evolução histórica do desempenho de CPU e dos circuitos de memória principal.



Características da hierarquia de memória

Nível	1	2	3	4
Nome	registos	*cache*	memória principal	memória de massa
Tamanho típico	< 1 KB	< 16 MB	< 16 GB	> 100 GB
Tecnologia	memória multi-porto, CMOS	CMOS SRAM	CMOS DRAM	disco magnético
Acesso (ns)	0.25–0.5	0.5–2.5	80–250	5×10^6
Largura de banda (MB/s)	$2 \times 10^3 - 10^5$	5000–10000	1000–5000	20–150
Gerido por	compilador	*hardware*	OS	OS/admin
Backup	*cache*	memória principal	memória de massa	CD/banda

Revisão dos termos básicos

- *cache hit*—CPU encontra item na *cache* (acerto).
- *cache miss*—CPU *não* encontra item na *cache* (falha).
- *bloco*—conjunto de dados (de tamanho fixo) que é obtido de memória e colocado em *cache* (contém o item pedido).
- A proximidade temporal implica que o item será utilizado de novo dentro de pouco tempo.
  A proximidade espacial implica que haja uma probabilidade elevada de que os outros dados do bloco sejam usados dentro de pouco tempo.
- O tempo necessário para processar uma falha depende da latência e da largura de banda da memória.
  - A *latência* determina o tempo necessário para obter o primeiro elemento do bloco.
  - A *largura de banda* determina o tempo necessário para obter o resto do bloco.

Aspectos elementares de desempenho de *caches*

Símbolos:

- Tempo de execução (CPU): t
- Ciclos de relógio (CPU): n_c
- Período de relógio: $T = 1/f$
- Ciclos de protelamento no acesso a memória: n_m
- Número de acessos a memória: n_a
- Número de falhas: n_f
- Taxa de falhas: $t_f = n_f/n_a$
- Número de instruções: n_i
- Penalidade de falha (em ciclos): p_f

Assumindo que:
(a) n_c inclui o tempo de tratamento de *cache hit*;
(b) CPU protela durante *cache miss*.

$$t = (n_c + n_m) \times T$$

$$n_m = n_f \times p_f = n_i \times \frac{n_f}{n_i} \times p_f = n_i \times \frac{n_a}{n_i} \times t_f \times p_f$$

Aspectos elementares de desempenho de *caches* (cont.)

- Os elementos da última equação podem ser medidos com relativa facilidade.
- Taxas de falhas podem ser avaliadas por simulação usando um *address trace* (listagem dos endereços ordenados por ordem temporal de acessos).
- Ter em atenção que a taxa de falhas *não* é constante!
- Alternativa para obtenção dos dados: contadores embutidos no CPU.
- Assumiu-se que taxas de falhas e penalidade de acesso são iguais para leituras e escritas; uma equação mais detalhada deve distinguir as duas situações.
- Formulação alternativa de taxa de falhas (t_f): $n_f/n_i = f_i$ e então

$$f_i = \frac{t_f \times n_a}{n_i} = t_f \times \frac{n_a}{n_i}$$

  f_i é frequentemente dada em falhas/1000 instruções.

Quatro questões sobre hierarquias de memória

As questões mais importantes sobre um nível da hierarquia de memória são:

1. **Onde é colocado um bloco?** ***[colocação]***
2. **Como determinar se um bloco está presente?** ***[identificação]***
3. **Que bloco deve ser substituído quando ocorre uma falha?** ***[substituição]***
4. **O que acontece durante a escrita?** ***[estratégia de escrita]***

Estas questões podem colocar-se a todos os níveis da hierarquia de memória. As respostas dependem de aspectos arquitecturais e tecnológicos.

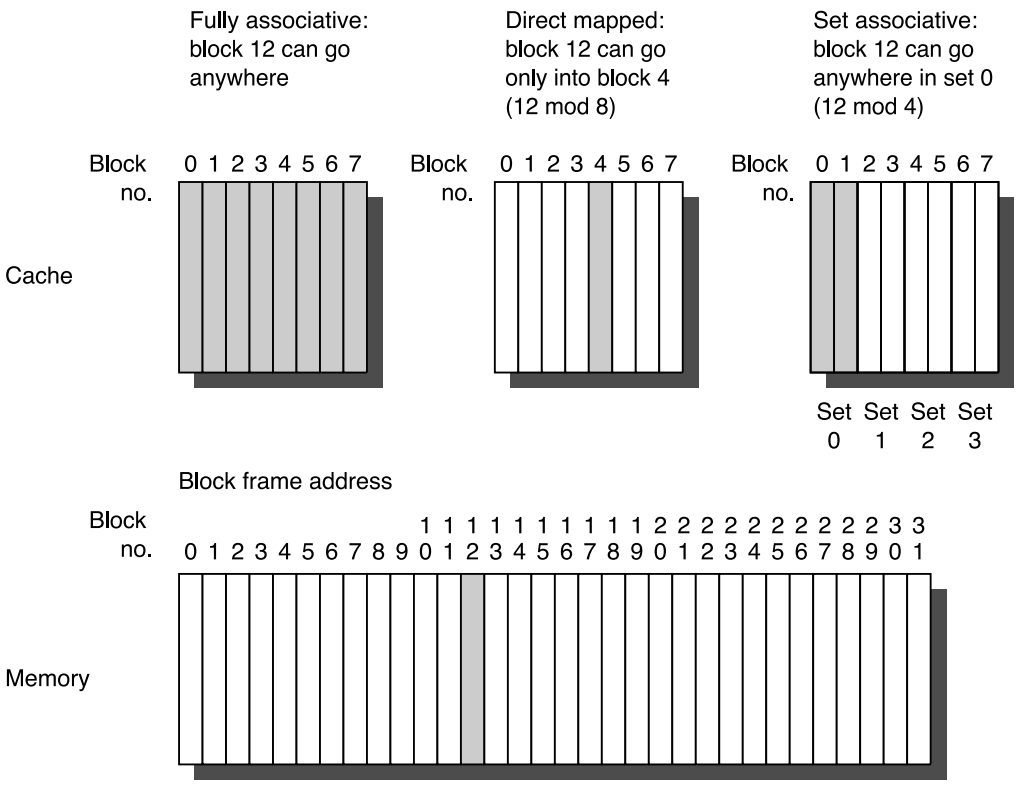
Localização dos dados

A localização dos dados em *cache* permite distinguir 3 categorias de organização:

1. Um bloco tem apenas um lugar possível: *mapeamento directo [direct mapped cache]*.
2. Um bloco pode ser colocado em qualquer posição: memória *totalmente associativa*.
3. Um bloco pode ser colocado em qualquer posição de num conjunto restrito: memória *associativa por conjuntos*. A associatividade é caracterizada por N, o número de elementos de um conjunto *[n-way set associative cache]*.

As implementações mais frequentes têm associatividade 1 (mapeamento directo), 2 ou 4.

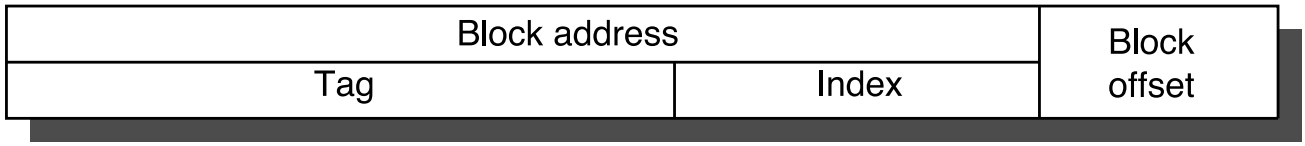
Associatividade



Identificação de um bloco em "cache"

- A cada bloco está associada uma *etiqueta* (parte do endereço).
- As etiquetas de todas as posições possíveis são verificadas em *paralelo*.
- Posições válidas são assinaladas por um bit de validade.

Estrutura de um endereço de memória:



Estratégia de substituição

- *Mapeamento directo*—Não há escolha, já que um bloco apenas pode residir numa posição.
- Para memórias associativas, a escolha do elemento do conjunto a ser substituído por ser feita:
  - ➤ *aleatoriamente*—a escolha é feita (pseudo-)aleatoriamente.
  - ➤ *least-recentely used (LRU)*—Usa o passado para predizer o futuro: o bloco não usado (acedido) há mais tempo é substituído (corolário do princípio da proximidade).
  - ➤ *first-in, first-out (FIFO)*—Aproximação a LRU: substituir o bloco mais velho.
- A implementação de LRU é a mais complicada, pelo que se usa frequentemente um método aproximado (principalmente FIFO).

Influência da estratégia de substituição no desempenho

Exemplo para *cache* de dados com blocos de 64 bytes (Alpha 21264).

	Associatividade								
	2-way			4-way			8-way		
Tam.	LRU	Aleat.	FIFO	LRU	Aleat.	FIFO	LRU	Aleat.	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.1
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

(Em falhas por milhar de instruções.)

Estratégia de escrita (1/2)

- Leituras são muito mais numerosas que escritas ($\approx$ 21% das instruções que acedem à *cache* de dados).
- O ênfase é colocado na leitura, que é a operação mais fácil.
- Existem duas grandes estratégias de escrita (assumindo que o bloco está presente em *cache*):
  1. *write through*—A informação é escrita simultanemente em *cache* e na memória do nível seguinte.
  2. *write back*—A informação é escrita apenas em *cache*: o bloco modificado é escrito em memória quando é substituído.
- É usado um bit (por bloco) para indicar se o bloco em *cache* foi modificado: *dirty bit*.
  Um bloco não modificado *(limpo)* não precisa de ser escrito em memória aquando da substituição.

Estratégia de escrita (2/2)

- Vantagens de *write-back*:
  1. escritas à velocidade da *cache*;
  2. múltiplas escrita no mesmo bloco requerem apenas um acesso à memória (reduz consumo de largura de banda e de potência).
- Vantagens de *write-through*:
  1. memória *cache* sempre "limpa" (uma falha de leitura não resulta em escritas na memória de nível seguinte);
  2. o nível seguinte mantém sempre a versão mais recente dos dados (facilita coerência de memória).
- Com *write-through*, o CPU pode protelar à espera que uma escrita termine *(write stall)*—usar um *write buffer*.
- Comportamento numa falha de escrita:
  1. *write-allocate*—um bloco é alocado para guardar o novo valor;
  2. *no-write-allocate*—apenas a memória de nível seguinte é actualizada; a *cache* não é afectada.
- *write-back & write-allocate*, *write-through & no-write-allocate*

Exemplo: Cache unificada vs. caches separadas

Tamanho (KB)	Instruções	Dados	Unificada
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61	36.9	39.4
128	0.30	35.3	36.2
256	0.02	32.6	32.9

Falhas por 1000 instruções; referências a instruções 74%; 2-way; blocos de 64 bytes.

Mais aspectos de desempenho de memórias *cache* (1/2)

- A taxa de falhas não constitui um bom índice de desempenho (o número de instruções executadas também não).
- O *tempo médio de acesso a memória* t_m é melhor:

$$t_m = t_h + t_f \times p_f$$

  em que t_h é o tempo gasta em acesso à *cache* (h=hit).
- Para CPU com execução em ordem, t_m é um bom índice de previsão do desempenho. (Convenção: o tempo t_h é incluído no tempo de execução de instruções.)
- Falhas de acesso têm um efeito duplo sobre o desempenho:
  1. Quanto menor for o CPI (de execução), tanto maior é o impacto relativo de um número fixo de ciclos de penalização por falha.
  2. Para duas hierarquias de memória idêntica, o CPU mais rápido tem uma penalidade maior (em ciclos).

Mais aspectos de desempenho de memórias *cache* (2/2)

- Para CPUs com execução fora de ordem, a situação é mais complicada. Aproximação:

$$\frac{n_m}{n_i} = \frac{n_f}{n_i} \times (L_T - L_0)$$

  L_T : atraso total da falha; L_0 : atraso "coberto" por outras operações. O tempo t_h também pode ser decomposto em duas parcelas.
- *latência total*—O que se deve considerar como início/fim de um acesso?
- *sobreposição de latência*—quando tem início a sobreposição com outras operações (quando está o processador a protelar)?
- Consideraremos o processador em protelamento num dado ciclo se o CPU não finalizar todas as instruções passíveis de finalização nesse ciclo. O protelamento é atribuído à primeira instrução não finalizada.
- A latência pode ser medida desde que a instrução está na fila de instruções OU desde que o endereço é gerado OU . . . Ser *consistente*!

Resumo das equações de desempenho de memórias *cache*

$$2^{\text{index}} = \frac{\text{Cache size}}{\text{Block size} \times \text{Set associativity}}$$

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$

$$\text{Memory stall cycles} = \text{Number of misses} \times \text{Miss penalty}$$

$$\text{Memory stall cycles} = \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

$$\frac{\text{Misses}}{\text{Instruction}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}}\right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}\right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty}\right) \times \text{Clock cycle time}$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}}{\text{Instruction}} \times (\text{Total miss latency} - \text{Overlapped miss latency})$$

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}_{L1}}{\text{Instruction}} \times \text{Hit time}_{L2} + \frac{\text{Misses}_{L2}}{\text{Instruction}} \times \text{Miss penalty}_{L2}$$

Múltiplos níveis de "cache"

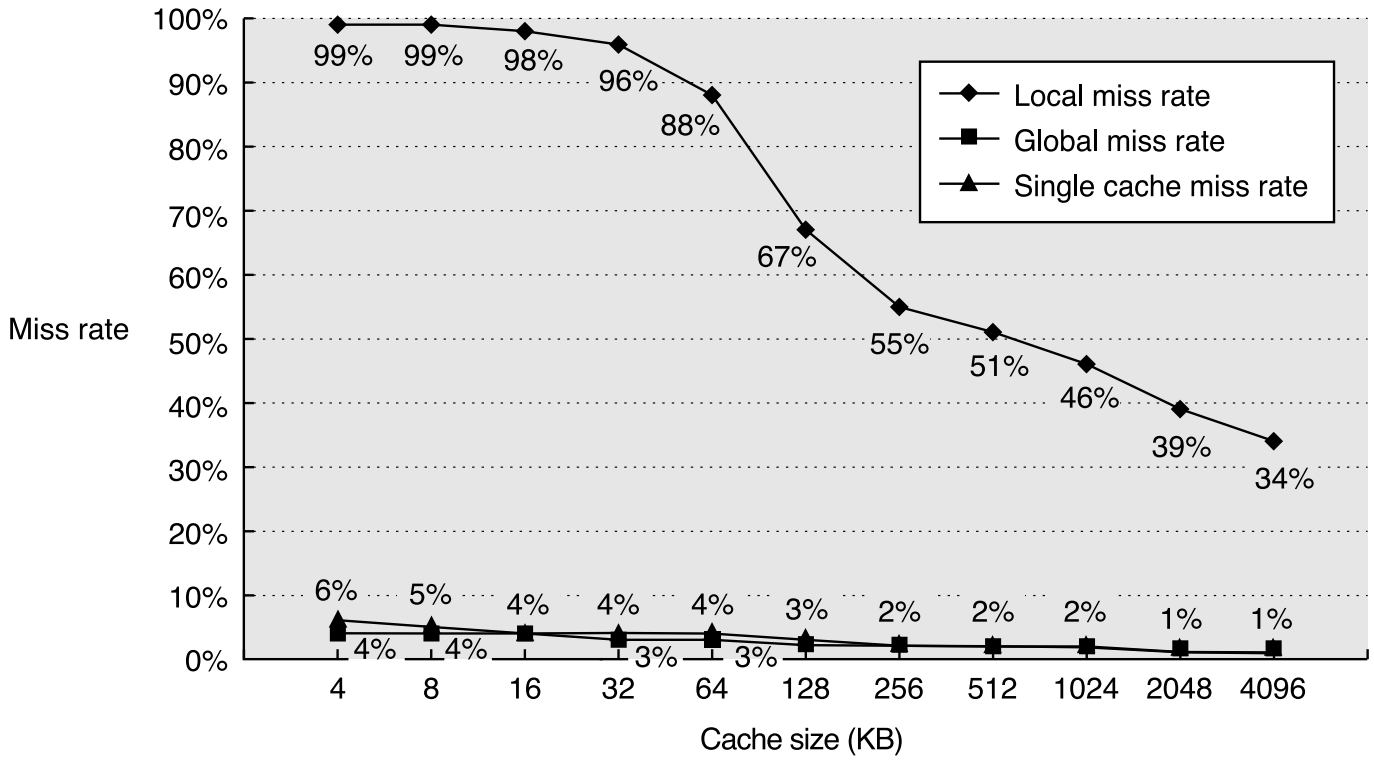
- As equações de desempenho indicam que melhorias da penalidade de falha são tão benéficas como melhorias da taxa de falhas.
- A crescente diferença de desempenho entre memórias e CPU leva a um aumento do custo relativo das penalidades.
- Técnica 1: Acrescentar níveis intermédios de memória *cache*.
- O tempo médio de acesso é:

$$t_m = t_{h1} + t_{f1} \times (t_{h2} + t_{f2} \times p_{f2})$$

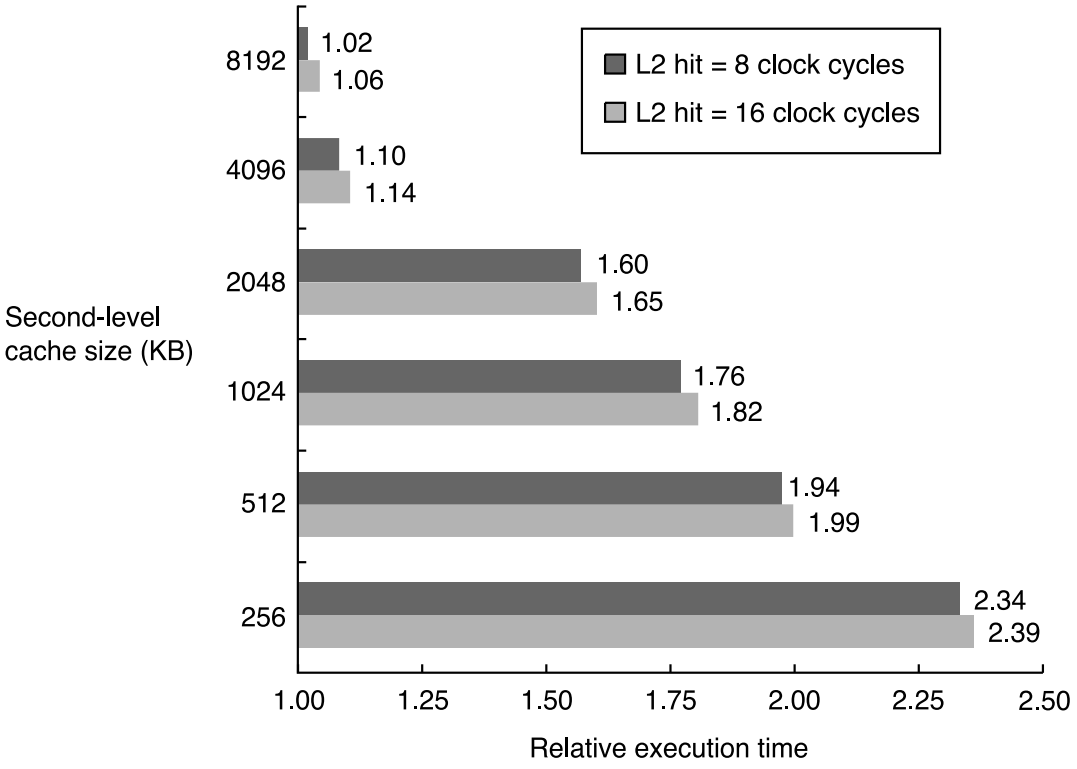
- *taxa de falhas local*—nº de falhas a dividir pelo número de acessos.
- *taxa de falhas global*—nº de falhas a dividir pelo número de acessos gerados pelo CPU.
- Número médio de falhas por instrução é

$$f_i = f_{i1} \times t_{h2} + f_{i2} \times p_{f2}$$

Medidas empíricas para *caches* multinível (1/2)



Medidas empíricas para *caches* multinível (2/2)



Classificação de falhas

- Falhas podem ser classificadas segundo os três C's:
  - *compulsórias*—O primeiro acesso a um bloco *não pode* ter sucesso (falhas de arranque a frio).
  - *capacidade*—Falhas causadas pela ausência de blocos que já estiveram em *cache* mas foram retirados por falta de espaço.
  - *conflito*—São falhas causadas pela impossibilidade de ocupação simultânea de uma posição ou conjunto (falhas de colisão ou interferência).
- Falhas compulsórias ocorrem mesmo em *caches* infinitas.
- Falhas compulsórias e de capacidade ocorrem em *caches* completamente associativas.
- Memórias associativas por conjuntos também têm falhas de conflito.

Repartição de taxas de falhas por categoria (1/2)

Cache de 4 kB:

Assoc.	Total	Compulsórias	(%)	Capacidade	(%)	Conflito	(%)
1	0.098	0.0001	0.1	0.070	72	0.027	28
2	0.098	0.0001	0.1	0.070	93	0.005	7
4	0.098	0.0001	0.1	0.070	99	0.0001	1
8	0.098	0.0001	0.1	0.070	100	0.000	0

Cache de 512 kB:

Assoc.	Total	Compulsórias	(%)	Capacidade	(%)	Conflito	(%)
1	0.008	0.0001	0.8	0.005	66	0.003	33
2	0.007	0.0001	0.9	0.005	71	0.002	28
4	0.006	0.0001	1.1	0.005	91	0.000	8
8	0.006	0.0001	1.1	0.005	95	0.000	4

Repartição de taxas de falhas por categoria (2/2)

Associatividade 1 :

Tam. (kB)	Total	Compulsórias	(%)	Capacidade	(%)	Conflito	(%)
4	0.098	0.0001	0.1	0.070	72	0.027	28
8	0.068	0.0001	0.1	0.044	65	0.024	35
16	0.049	0.0001	0.1	0.040	82	0.0009	17
32	0.042	0.0001	0.2	0.037	89	0.005	11

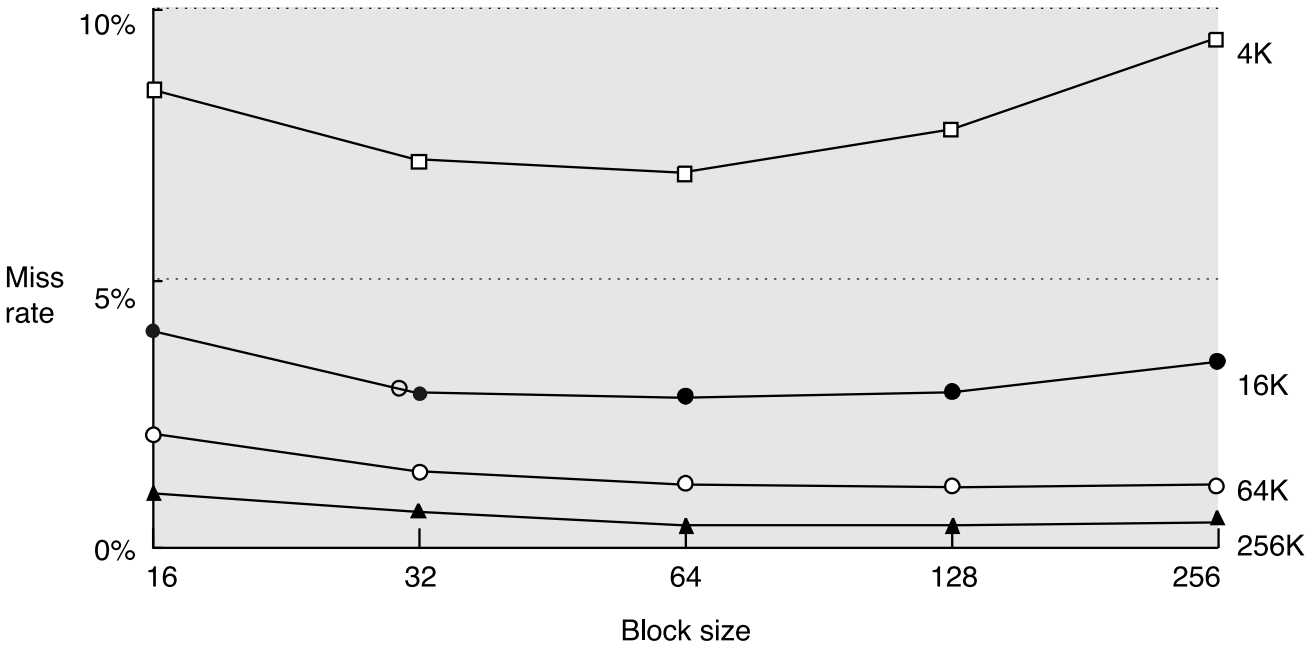
Associatividade 8:

Tam. (kB)	Total	Compulsórias	(%)	Capacidade	(%)	Conflito	(%)
4	0.071	0.0001	0.1	0.070	100	0.000	0
8	0.044	0.0001	0.1	0.044	100	0.000	0
16	0.041	0.0001	0.2	0.040	100	0.000	0
32	0.037	0.0001	0.2	0.037	100	0.000	0

Dimensão dos blocos

- A maneira mais simples de reduzir a taxa de falhas é aumentar a dimensão dos blocos (aproveitamento da proximidade espacial).
- Desvantagens:
  - aumenta a penalidade de falhas;
  - pode aumentar o número de falhas por conflito;
  - pode aumentar as falhas de capacidade (em *caches* pequenas).
- A escolha da dimensão dos blocos também depende das características do nível seguinte de memória: baixa latência $\rightarrow$ blocos pequenos.

Dimensão dos blocos: medidas empíricas



Dimensão dos blocos vs. tempo de acesso

		Tamanho da *cache*			
Bloco (B)	Penal.	4 kB	16 kB	64 kB	256 kB
16	82	8.027	4.231	2.673	1.894
32	84	7.082	3.411	2.134	1.588
64	88	7.160	3.323	1.933	1.449
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

- Técnica 2: Aumentar o tamanho da memória *cache*.
- Aumento de custo e de tempo de acesso.

Aumento de associatividade

- Aumentar a associatividade melhora a taxa de falhas mas aumenta o tempo médio de acesso.
- Regras empíricas:
  1. Para efeitos práticos, associatividade 8 = infinito.
  2. Uma memória *direct mapped* de tamanho N tem a mesma taxa de falhas que uma memória de associatividade 2 e de tamanho N/2.

	Associatividade			
Tamanho (kB)	1	2	4	8
4	3.44	3.24	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25