Organização de Computadores II
DCC007

Aula 13 – Desempenho de Caches, Melhorando o Desempenho

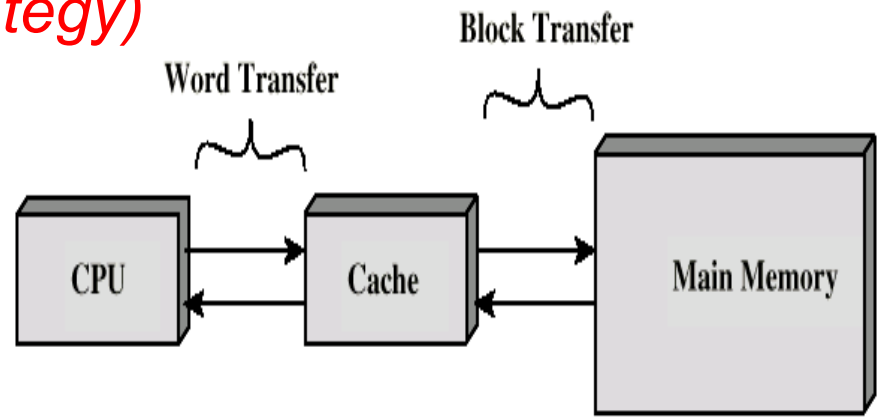
Prof. Omar Paranaiba Vilela Neto

Organização de Caches

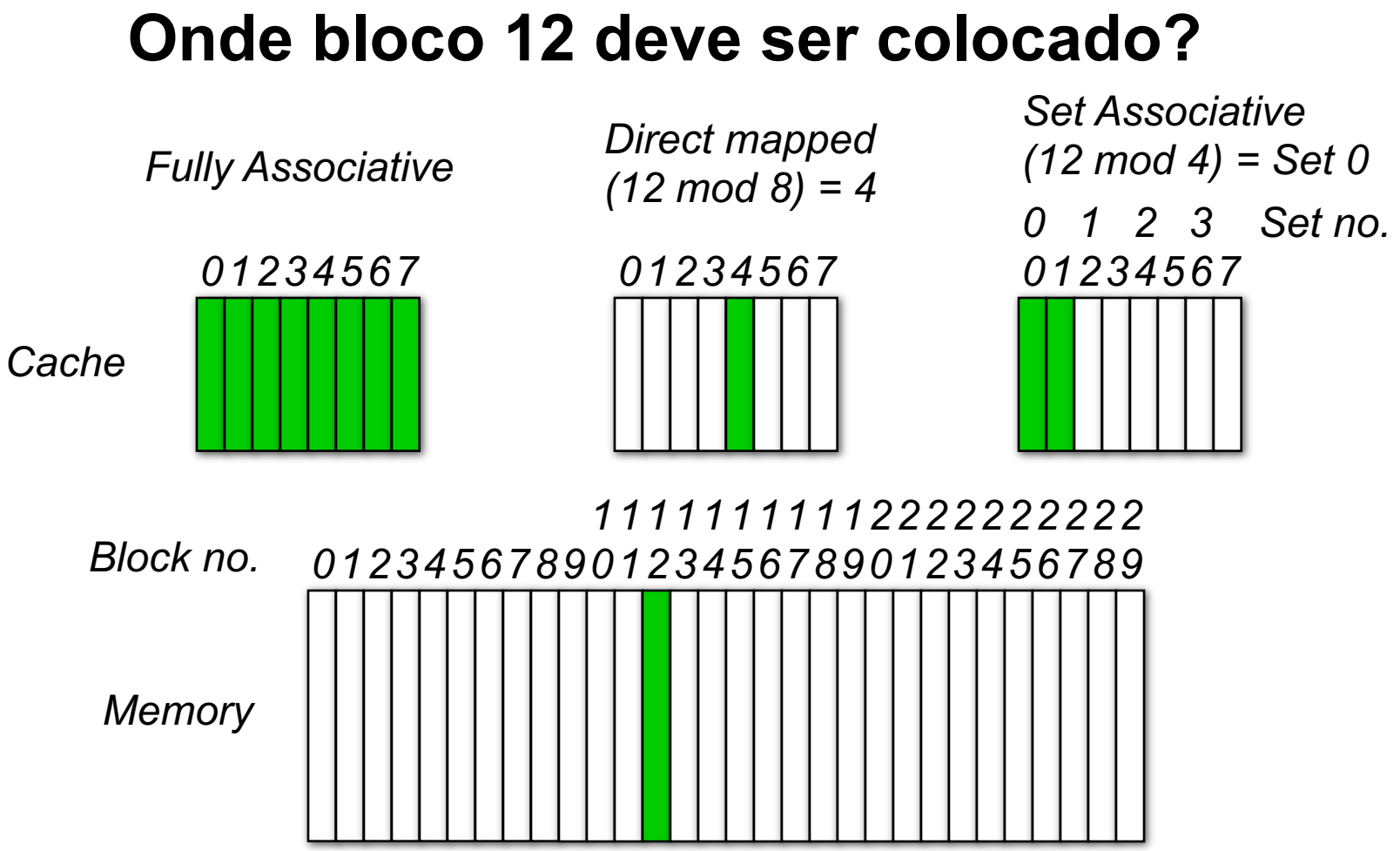
- Mapeamento Direto;
- Totalmente Associativa;
- Associativa por Conjunto.

Quatro Perguntas Básicas sobre Hierarquia de Memória

- Q1: Onde o bloco vai ser colocado na memória de nível mais alto? *(Block placement)*
- Q2: Como bloco é encontrado na memória de nível mais alto?
  (Block identification)
- Q3: Quais blocos serão trocados em um miss?
  (Block replacement)
- Q4: O que acontece em uma escrita?
  (Write strategy)



Q1: Onde Bloco Deve Ser Colocado no Nível Mais Alto?



Q2: Como o Bloco é Encontrado no Nível Mais Alto?

- Tag para cada bloco
  - Não é necessário checar índice ou offset

- Aumento de associatividade reduz índice, aumenta tag

Q3: Qual Bloco Será Trocado Durante um Miss?

- Fácil de decidir para caches de mapeamento direto
- Quando empregar cada técnica?
  - Aleatório (associatividade alta)
  - LRU (associatividade baixa)
  - FIFO (para caches menores)

Q3: Qual Bloco Será Trocado Durante um Miss?

Falhas na cache por 1000 instruções

Size	Associativity								
	2-way			4-way			8-way		
	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Q4: O Que Acontece Durante uma Escrita?

- Write through: A informação é escrita tanto para o bloco da cache quanto para a memória de nível mais baixo.
- Write back: A informação é escrita somente para o bloco da cache. Este bloco é escrito na memória quando ele for trocado.
  - Precisa de dirty bit
- Vantagens e Desvantagens:
  - WT: miss de leitura não resulta em escrita na memória
  - WB: reduz BW para memória de nível mais baixo
- WT está sempre combinada com write buffers de forma a não precisarmos esperar pelo nível mais baixo de memória

Q4: O Que Acontece Durante uma Escrita?

- Write allocate (fetch on write) : bloco é trazido para cache se ocorrer um miss de escrita, seguido por uma escrita com hit.

- No-write allocate (write around): bloco é modificado no nível mais baixo e só depois carregado na cache.

- WB é geralmente utilizado com write allocate
- WT é geralmente utilizado com no-write allocate

Desempenho de Caches

CPU time = (ciclos de CPU + ciclos de stall de Memória) x período do clock

ciclos de stall de Memória = (Reads x MR p/ Read x MP p/ Read + Writes x MR p/ Write x MP p/ Write)

ciclos de stall de Memória = número de acessos de Memória x MR x MP

Desempenho de Caches

CPUtime = IC x ($CPI_{execution}$ + acessos de Mem por instr x Miss rate x Miss penalty) x período do clock

Misses por instr = acessos de Mem por instr x Miss rate

CPUtime = IC x ($CPI_{execution}$ + Misses por instr x Miss penalty) x período do clock

Desempenho de Caches

- Na verdade, cada arquitetura de cache possuirá peculiaridades específicas
  - Cache de dados e instrução separadas
  - WT ou WB
  - WB com write buffer
- Como isso afeta a equação de desempenho das caches?

Exemplo

- Qual MR é melhor? 16 KB I-cache + 16 KB D-cache ou 32 KB unificadas? Assuma que 36% das instruções são de transferências de dados.

Tamanho	Cache de instruções	Cache de dados	Cache unificada
8 KB	8,16	44,0	63,0
16 KB	3,82	40,9	51,0
32 KB	1,36	38,4	43,3
64 KB	0,61	36,9	39,4
128 KB	0,30	35,3	36,2
256 KB	0,02	32,6	32,9

Figura 5.8 **Erros por 1.000 instruções para caches de instruções, de dados e unificadas de diferentes tamanhos.** A centagem de referências de instruções é aproximadamente 74%. Os dados são para caches associativas de duas vias com bl de 64 bytes, referentes ao mesmo computador e aos mesmos benchmarks da Figura 5.6.

Exemplo

- Qual MR é melhor? 16 KB I-cache + 16 KB D-cache ou 32 KB unificadas? Assuma que 36% das instruções são transferências de dados.

MR(16 KB I-cache) = 3.82 /1000/ 1.0= 0.004

MR(16 KB D-cache) = 40.9/1000/0.36 = 0.114

MR(split) = 0.74 * 0.004 + 0.26 * 0.114 = 0.0324

MR(32 KB) = 43.3/1000/(1+0.36) = 0.0318

Melhorando o Desempenho de Caches

$$AMAT = HT + MR * MP$$

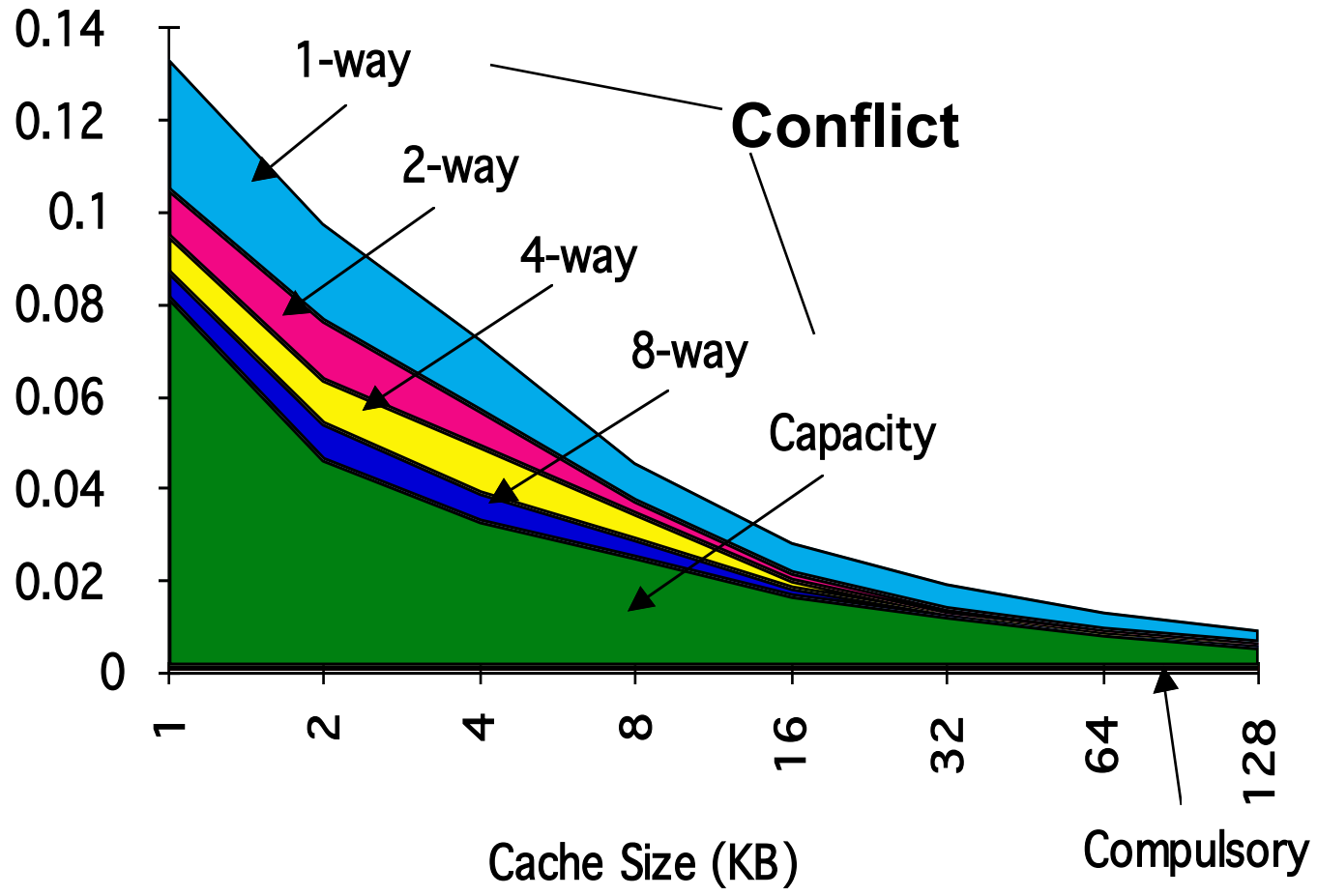
- Reduzindo MP,
- Reduzindo MR,
- Reduzindo o tempo de hit da cache.

6
Otimizações Básicas de Caches

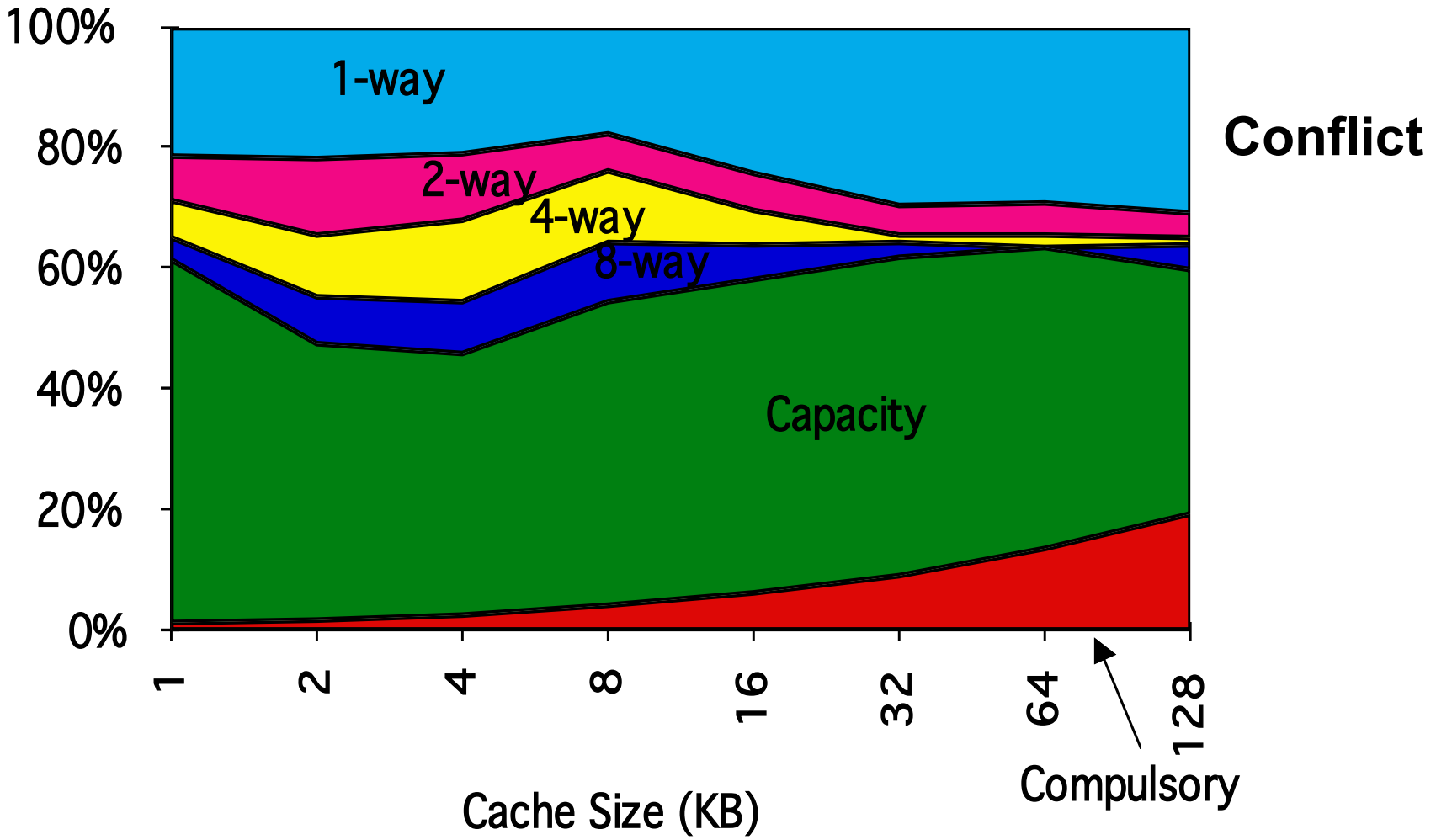
Reduzindo Misses

- Tipos de Misses: 3 Categorias
  - *Compulsório*—O primeiro acesso a um bloco que não está na cache para que o bloco possa ser trazido pela primeira vez para a cache. Também chamado de *cold start misses* ou *misses de primeira referência*. *(Misses para Caches Infinitas)*
  - *Capacidade*—Se a cache não pode conter todos os blocos necessários durante a execução do programa, misses de capacidade ocorrerão devido aos blocos terem que ser descartados e depois trazidos novamente. *(Misses em Tamanho X Cache)*
  - *Conflito*—Se a estratégia de colocação de blocos é associativa por conjuntos ou mapeamento direto, misses de conflito ocorrerão se um bloco deve ser descartado porque muitos blocos mapearam para o conjunto. Também são chamados de *misses de colisão* ou *misses de interferência*.
    (Misses em N-way Associative, Tamanho X Cache)

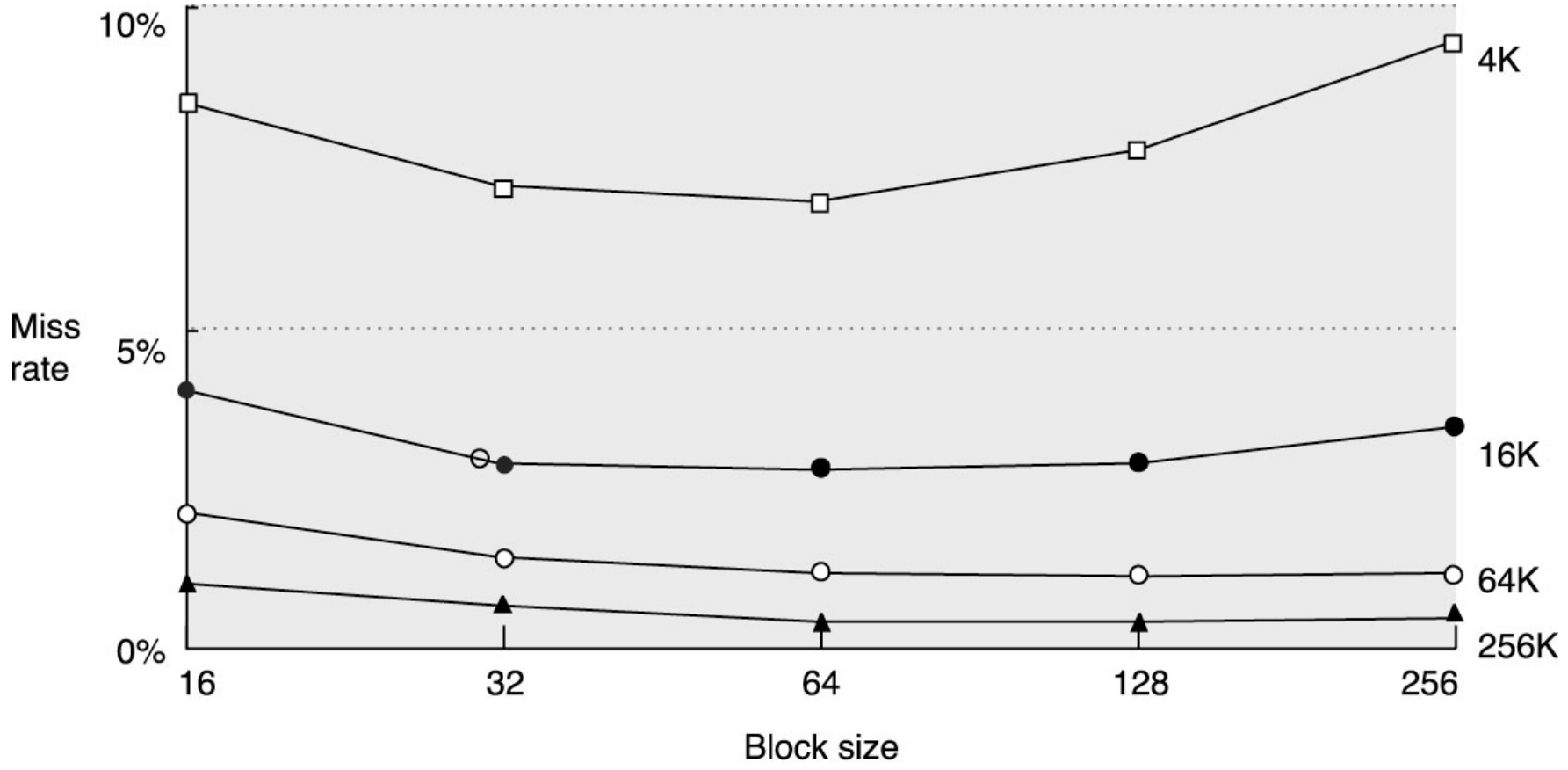
Miss Rate Absoluto p/ 3Cs



Miss Rate Relativo p/ 3Cs



1. Reduzindo Misses
Aumentando Tamanho do Bloco



AMAT é a Única Medida Correta

- Exemplo: sistema de memória fornece 16 bytes em 82 ciclos, 32 bytes em 84 ciclos, …

  AMAT = HT + MR * MP

  AMAT (16) = 1 + (3.94% * 82) = 4.231

  AMAT (32) = 1 + (2.87% * 84) = 3.411

  AMAT (64) = 1 + (2.64% * 88) = 3.323

  AMAT(128) = 1 + (2.77% * 96) = 3.659

2. Reduzindo Misses
Caches Maiores

- Aumentando caches reduz miss rate
- Mas lembre-se…

  AMAT = HT + MR * MP

  HT aumenta com caches maiores…

3. Reduzindo Misses Aumentando Associatividade

- Associatividade 8 é tão boa quanto uma cache completamente associativa
- Regra 2:1 para caches (<128KBytes):
  - Miss Rate DM p/ tamanho de cache N = Miss Rate de cache 2-way Associative com tamanho N/2
    Cuidado: **Tempo de execução** é a unidade final de medição!

Exemplo: Avg. Memory Access Time (AMAT) x Miss Rate

$AMAT_{8\text{-}w} = 1.52 + MR_{8\text{-}w} * 25$

$AMAT_{4\text{-}w} = 1.44 + MR_{4\text{-}w} * 25$

$AMAT_{2\text{-}w} = 1.36 + MR_{2\text{-}w} * 25$

$AMAT_{1\text{-}w} = 1.00 + MR_{1\text{-}w} * 25$

Cache size(KB)	Associatividade			
	1-way	2-way	4-way	8-way
4	3.44	3.25	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

4. Reduzir Penalidade
Caches multiníveis

- Equações para L2

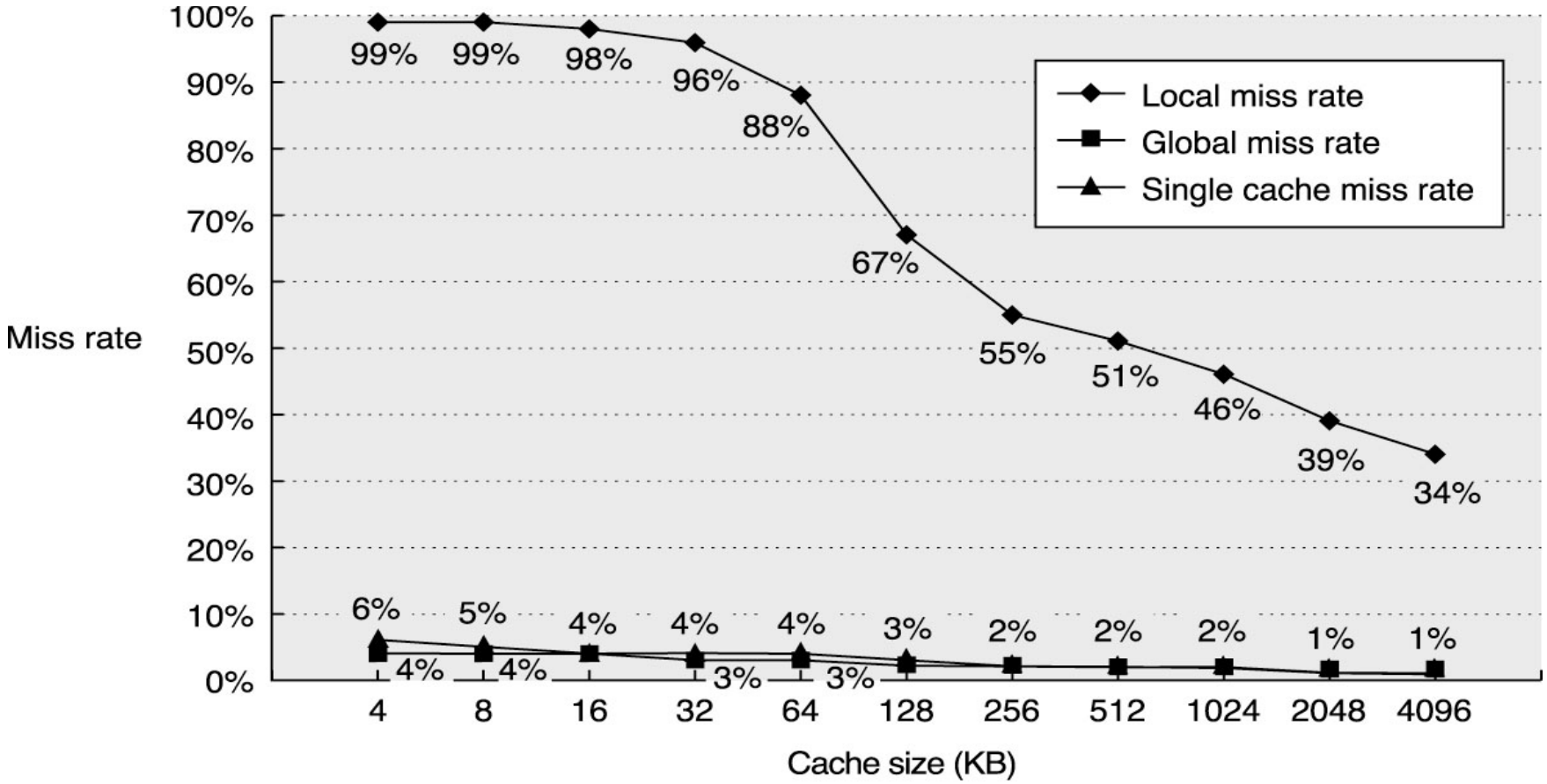
AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$

Miss Penalty$_{L1}$ = Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$

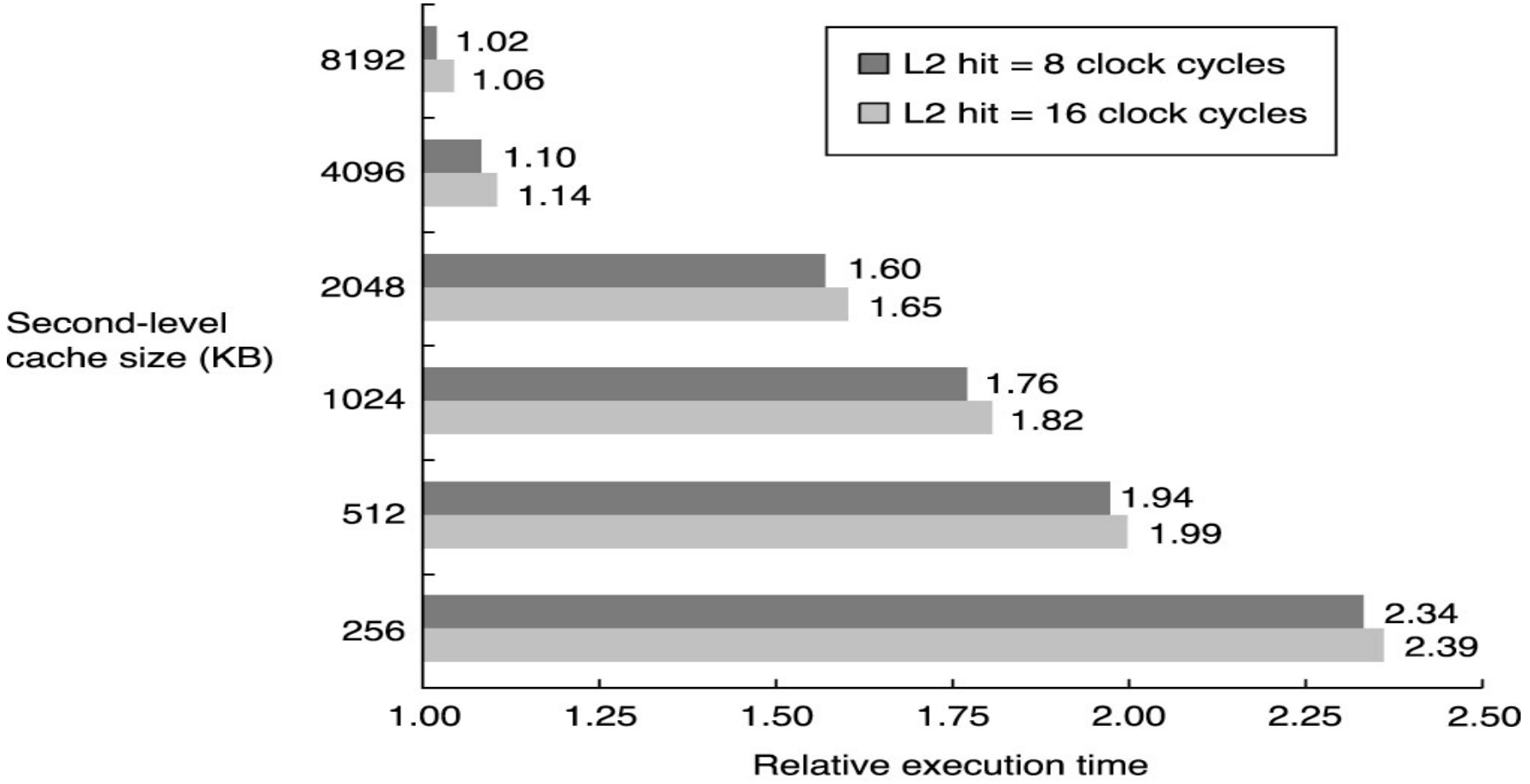
AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x (Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$)

- Definições:
  - *Local miss rate*— misses da cache divididas pelo número total de acessos para *esta cache* (Miss rate$_{L2}$)
  - *Global miss rate*— misses da cache divididos pelo número total de acessos de memória *gerados pela CPU*
    (Miss Rate$_{L1}$ x Miss Rate$_{L2}$)

MR vs. Tamanho das Caches



Variação de ET pelo Tamanho de L2



ref: 8192 L2, 1 ciclo

5. Reduzir Penalidade

Aumentando Prioridade a Read Misses sobre Writes

```
SW R3, 512(R0)
LW R1, 1024(R0)
LW R2, 512(R0)
```

- Blocos 512 e 1024 estão na mesma linha da cache, organizada como mapeamento direto

Aumentando a Prioridade de Reads sobre Writes

- Write back com write buffers oferecem conflitos RAW com leituras da memória principal em um miss de cache
- Parando leitura até buffer esvaziar pode aumentar MP em 50%
- Verifique write buffer antes de iniciar leitura; se não houver conflitos, deixe leitura continuar
- Write Back?
  - Read miss troca bloco sujo
  - Normal: Escreva bloco sujo na memória, e leia novo bloco
  - Copie bloco sujo para write buffer, leia o novo bloco e então escreva bloco na memória
  - Stall é menor porque não precisa aguardar término da escrita

6. Reduzir Tempo de Acerto Evitando Tradução de Endereço

- Envio de endereço virtual p/ cache? Chamado Virtually Addressed Cache ou Virtual Cache vs. Physical Cache
  - Cada vez que processo é trocado, precisamos limpar (flush) a cache; caso contrário podemos ter hits falsos
    - Custo = tempo para "flush" + misses "compulsórios" causados pela limpeza
  - aliases (ou sinônimos);
    Dois endereços virtuais apontando para mesmo endereço físico
  - I/O deve interagir com cache
- Solução para aliases
  - HW garante aliases são mapeados para mesma linha (DM) se eles possuem campos de índices iguais (page coloring)
- Solução para flush
  - Adicione em campo de tag identificador do processo: hit não pode ocorrer se processo for diferente

Evitando Tradução: Índice com Parte Física de Endereço

- Se índice é parte física de endereço, podemos iniciar acesso à cache em paralelo com tradução de endereço

- Limita cache à tamanho da página: O que podemos fazer se quisermos caches maiores com a mesma característica?
  - Maior associatividade
  - Page coloring

11

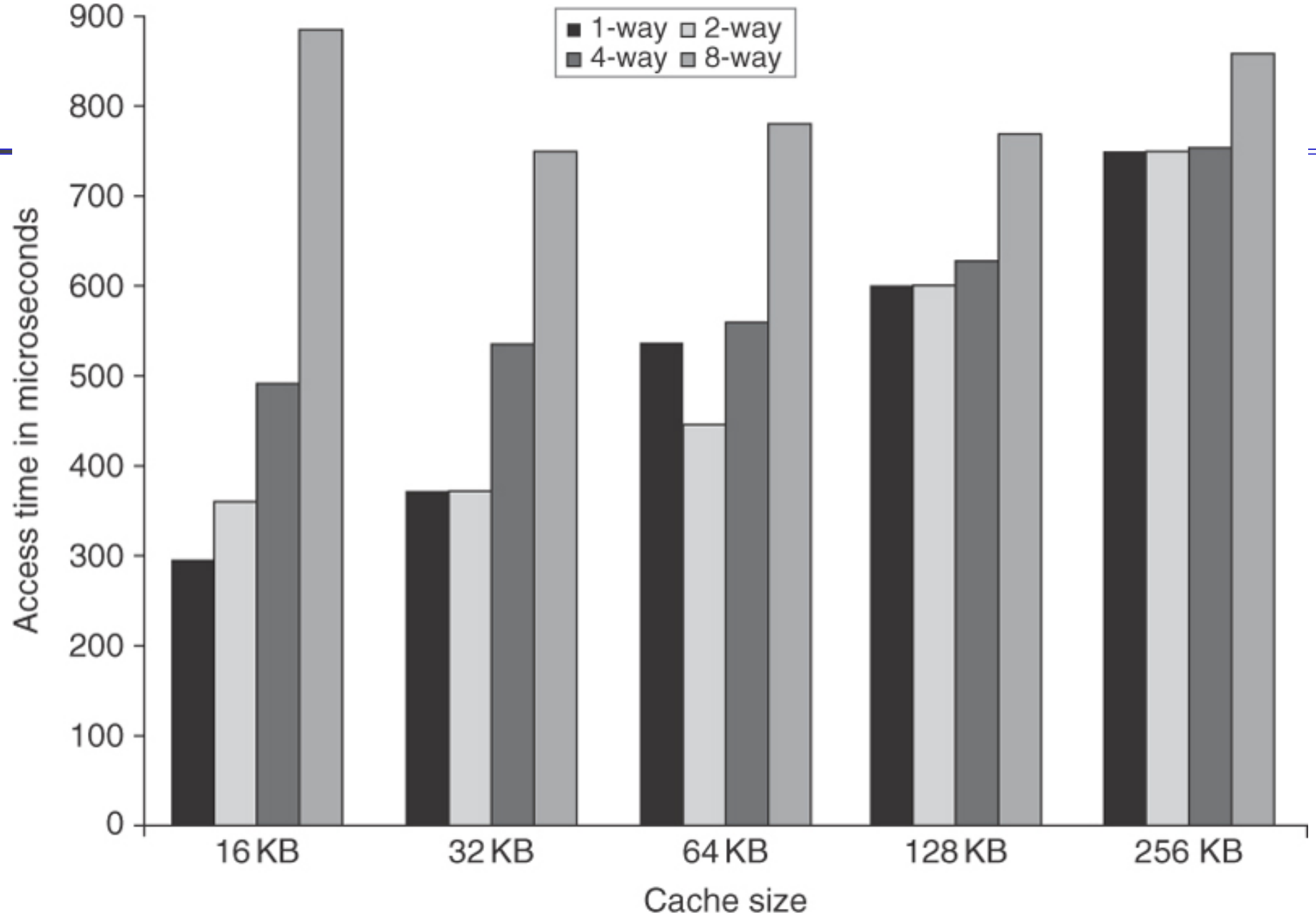
Otimizações Avançadas de Cache

1. Reduzir Tempo de Acerto

Caches Menores e Mais Simples

- Por que Alpha 21164 possui 8KB I-cache e 8KB D-cache + 96KB L2.

- Mapeamento direto no chip


1-way
2-way
4-way
8-way
Access time in microseconds
900
800
700
600
500
400
300
200
100
0
16 KB
32 KB
64 KB
128 KB
256 KB
Cache size

2. Reduzir Tempo de Acerto

Previsão de via

- Um bit marca qual o bloco mais provável de conter o dado em uma cache n-way
  - Faz decisão antecipada da direção do multiplexador para uma cache n-way – compara apenas um bloco.
  - Se previsão for errada, gasta mais tempo – compara outros blocos

2. Reduzir Tempo de Acerto

Previsão de via

- Como combinar hit rápido de mapeamento direto e ter a mesma taxa de miss de conflito de uma cache 2-way SA?
- Dividir a cache: em um miss, cheque outra metade da cache para ver se bloco está lá (pseudo-hit ou slow hit)

3. Aumentar largura de banda

Acesso à cache em Pipeline

- Acesso à cache em múltiplos ciclos
  - Pentium: 1 ciclo de clock
  - Pentium Pro e III: 2 ciclos de clock
  - Pentium 4 e I7: 4 ciclos de clock

4. Aumentar largura de banda

Caches sem bloqueios

- Non-blocking cache ou lockup-free cache permite cache de dados continuar a suprir dados que sejam hits enquanto ela processa um miss
- "hit under miss" reduzem a penalidade de misses ao fazer alguma coisa útil durante tratamento de miss
- "hit under multiple miss" ou "miss under miss" pode melhorar ainda mais a penalidade efetiva de misses
  - Mas podem aumentar de maneira significativa complexidade do controlador da cache pois a cache deverá manter lista de acessos não terminados

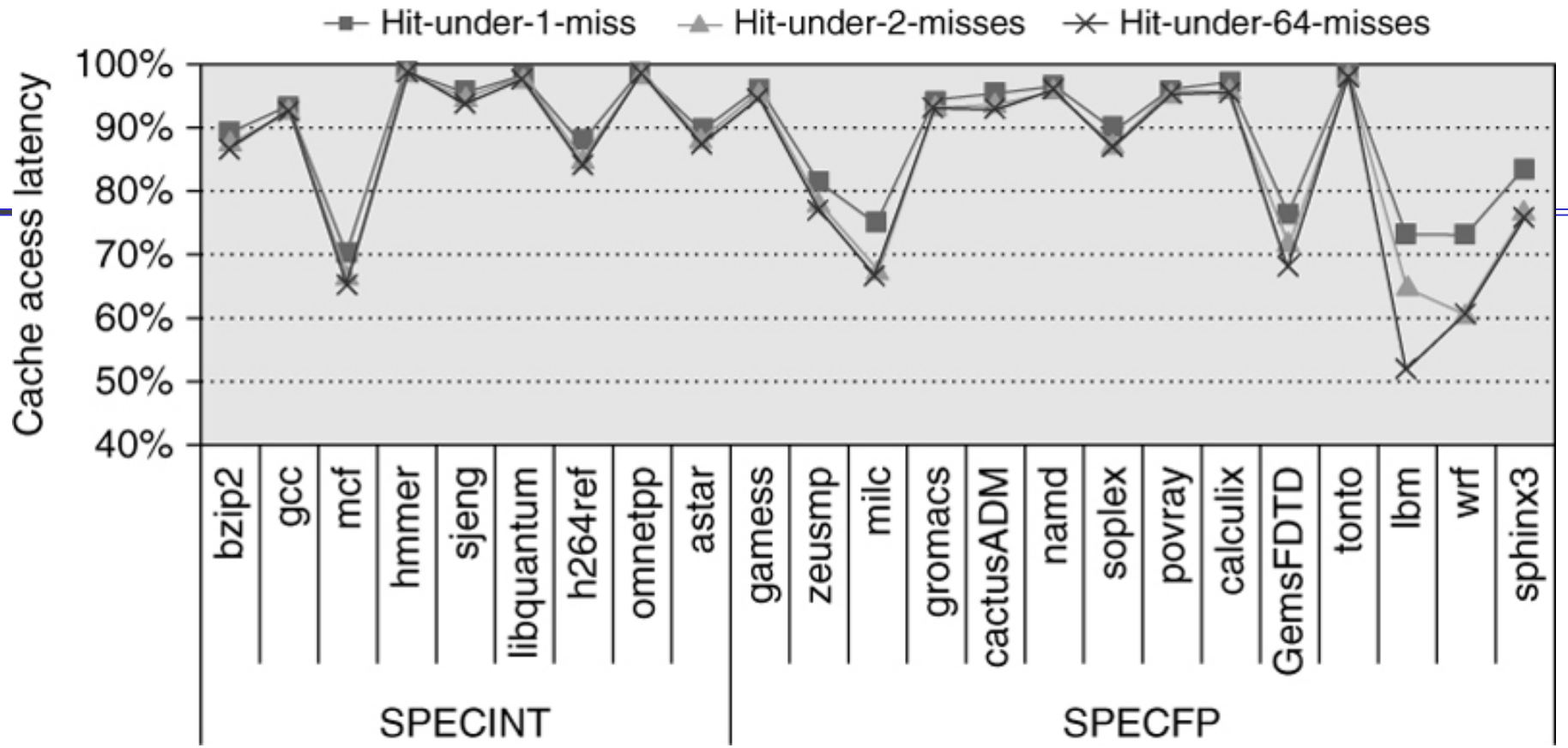


Figure 2.5 The effectiveness of a nonblocking cache is evaluated by allowing 1, 2, or 64 hits under a cache miss with 9 SPECINT (on the left) and 9 SPECFP (on the right) benchmarks. The data memory system modeled after the Intel i7 consists of a 32KB L1 cache with a four cycle access latency. The L2 cache (shared with instructions) is 256 KB with a 10 clock cycle access latency. The L3 is 2 MB and a 36-cycle access latency. All the caches are eight-way set associative and have a 64-byte block size. Allowing one hit under miss reduces the miss penalty by 9% for the integer benchmarks and 12.5% for the floating point. Allowing a second hit improves these results to 10% and 16%, and allowing 64 results in little additional improvement.

5. Aumentar largura de banda
Caches multibanco

- Dividir a cache em bancos independentes
  - Suporte à acessos simultâneos
  - ARM Cortex-A8:
    - 1 a 4 bancos em L2
  - Intel i7
    - 4 bancos em L1 (2 acessos por clock)
    - 8 bancos em L2

5. Aumentar largura de banda

Caches multibanco

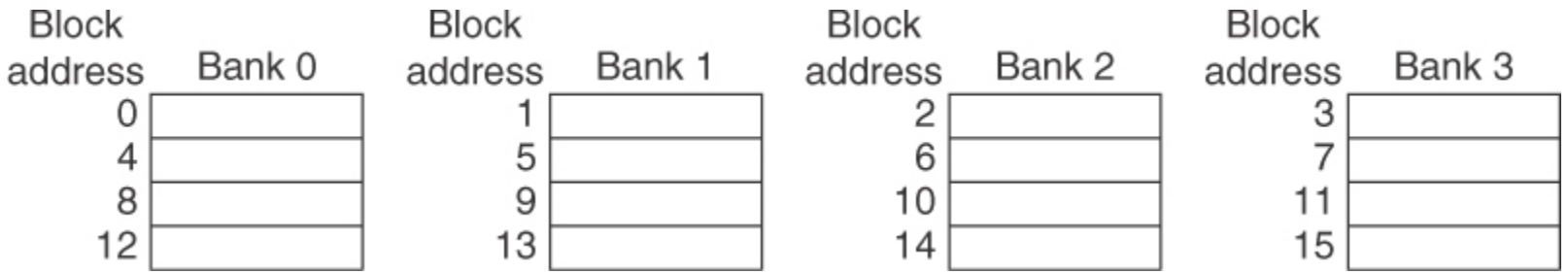


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

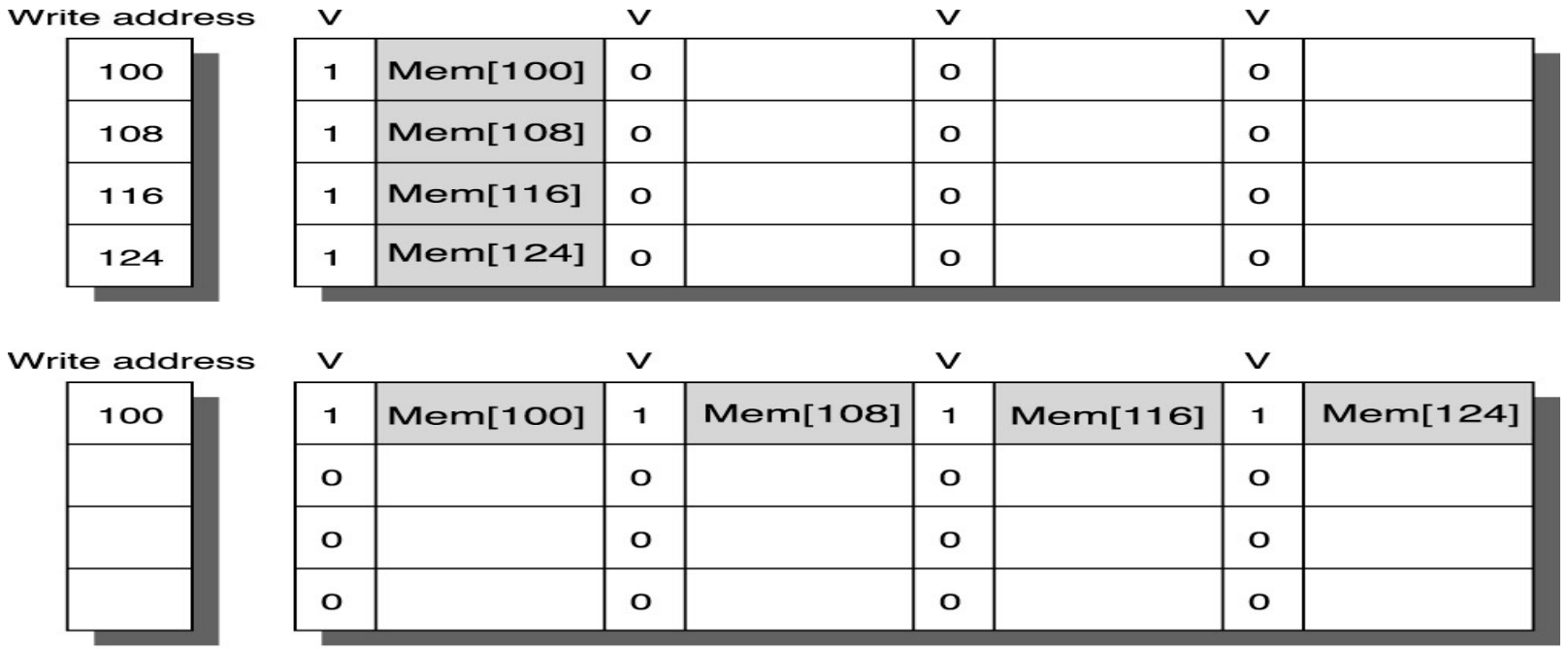
6. Reduzir Penalidade
Palavra crítica primeiro

- Não aguarde até bloco estar completamente carregado para liberar CPU
  - *Early restart*—Tão logo quanto palavra buscada estiver disponível, envie-a para CPU e deixe CPU continuar execução
  - *Critical Word First*—Busque palavra desejada primeiramente da memória e envie-a para CPU tão logo ela chegue; deixe CPU continuar execução enquanto restante do bloco é buscado.

    Também chamado de *wrapped fetch* e *requested word first*
- Bom para blocos grandes
- Localidade espacial é um problema; há uma tendência de se precisar buscar próxima palavra sequencial

7. Reduzir Penalidade
Merging Write Buffer

- Acesso sequencial de blocos grandes é mais eficiente em memória
- Write buffer pode ficar sobrecarregado com muitos acessos de pequenos blocos sequenciais



8. Reduzir Falha
Otimizações de Compiladores

- Instruções
  - Reorganizar procedimentos em memória para reduzir misses
  - Profiling para checar conflitos
  - McFarling [1989] reduziu misses de cache em 75% em 8Kb mapeamento direto com blocos de 4 bytes
- Dados
  - Merging Arrays: melhora localidade espacial colapsando 2 vetores em um único vetor
  - Loop Interchange: muda aninhamento de loops para acessar dados na ordem de armazenamento em memória
  - Loop Fusion: Combina 2 loops independentes que possuem mesmo controle de loop e alguma sobreposição de variáveis
  - Blocking: Melhora localidade temporal acessando blocos que dependem dos mesmos dados repetidamente vs. varrendo todas as colunas e linhas das matrizes.

Exemplo: *Merging Arrays*

```
/* Before */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
   int val;
   int key;
};
struct merge merged_array[SIZE];
```

Reduzindo conflitos entre val & key

Exemplo: *Loop Interchange*

```
/* Before */
for (j = 0; j < 100; j = j+1)
  for (i = 0; i < 5000; i = i+1)
     x[i][j] = 2 * x[i][j];

/* After */
for (i = 0; i < 5000; i = i+1)
  for (j = 0; j < 100; j = j+1)
     x[i][j] = 2 * x[i][j];
```

Acessos sequenciais ao invés de acessá-los sequencialmente a cada 100 palavras

Exemplo: *Loop Fusion*

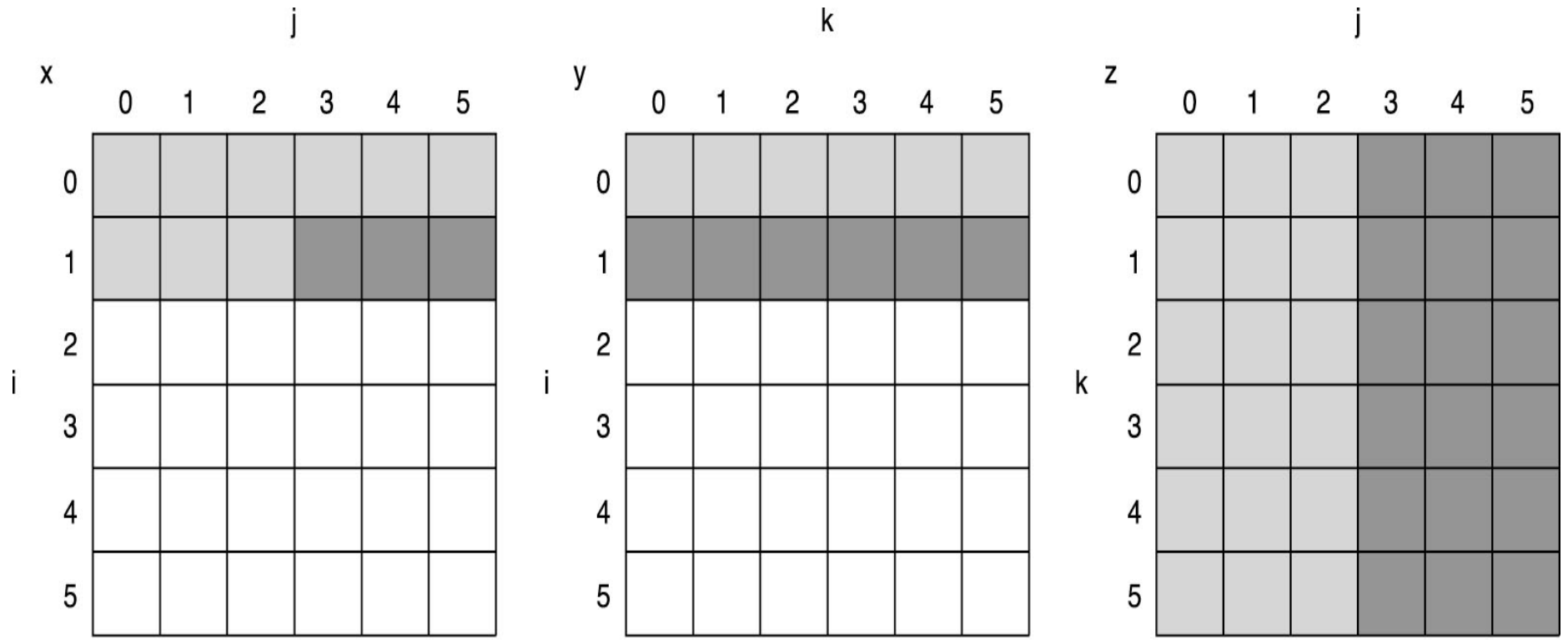
```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
     a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
     d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {  a[i][j] = 1/b[i][j] * c[i][j];
     d[i][j] = a[i][j] + c[i][j];}
```

2 misses por acesso de a e c vs. um miss por acesso

Exemplo: *Blocking*

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
     {r = 0;
      for (k = 0; k < N; k = k+1){
       r = r + y[i][k]*z[k][j];};
      x[i][j] = r;
     };
```

Blocking



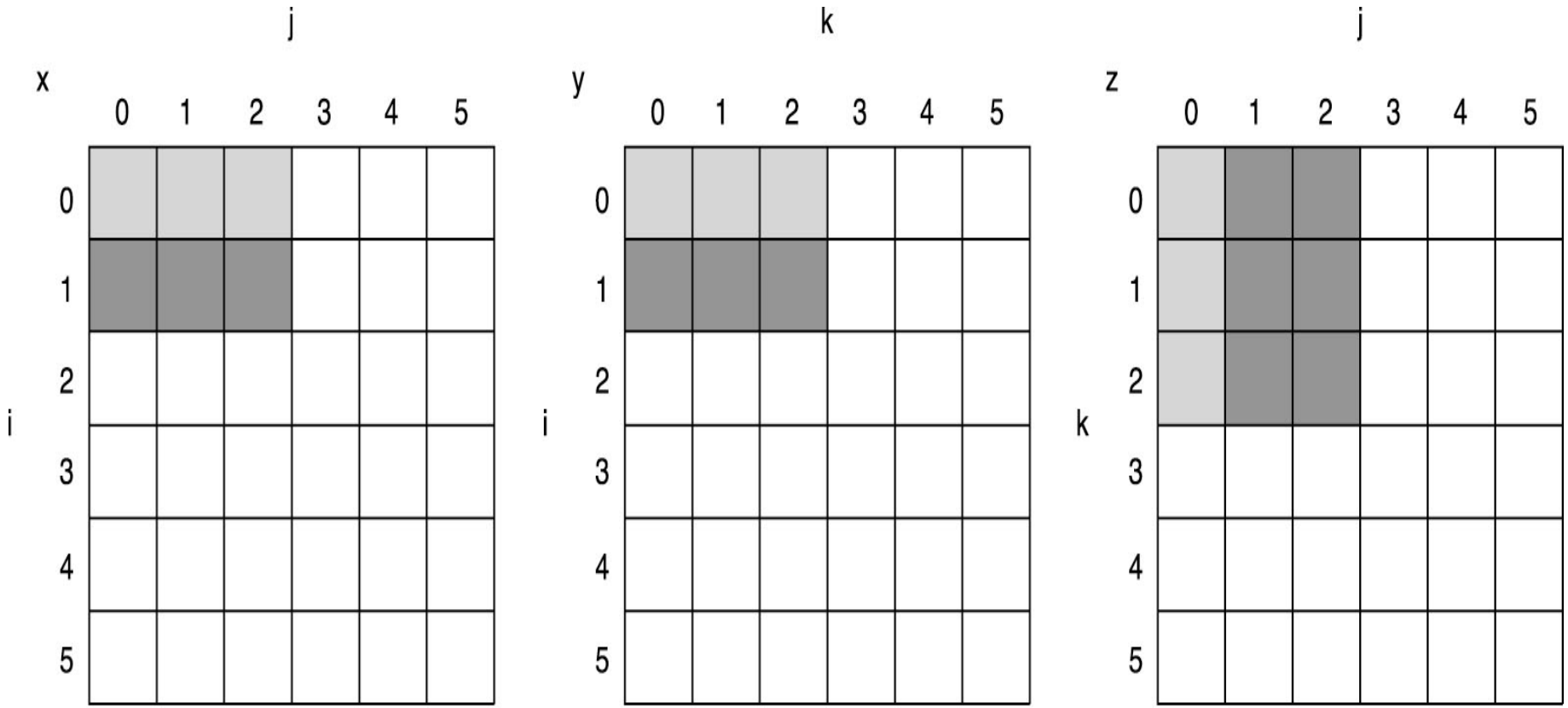
Blocking

- Dois loops mais internos:
  - Lêem todos NxN elementos de z[]
  - Lêem N elementos de 1 linha de y[] repetidamente
  - Escrevem N elementos de 1 linha de x[]
- Misses de Capacidade são uma função de N e tamanho de cache (3 NxN)
- Idéia: calcular submatriz BxB que cabe na cache

Blocking

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B,N); j = j+1)
     {r = 0;
      for (k = kk; k < min(kk+B,N); k = k+1) {
        r = r + y[i][k]*z[k][j];};
      x[i][j] = x[i][j] + r;
     };
```

Blocking



Exemplo: Blocking

- Misses de capacidade de $2N^3 + N^2$ a $2N^3/B + N^2$
- B é chamado de Blocking Factor

9. Reduzir Falha ou Penalidade
HW Prefetching de Instrução & Dados

- E.g., Prefetching de Instrução
  - Alpha 21064 busca 2 blocos em um miss
  - Bloco extra é colocado em stream buffer
  - Caso ocorra um miss, cheque stream buffer antes de gerar acesso à memória
- Também funciona com dados:
  - Jouppi [1990] 1 stream buffer de dados pegou 25% misses em cache de 4KB; 4 streams pegou 43%
  - Palacharla & Kessler [1994] para programas científicos com 8 streams pegou 50% a 70% dos misses de 2 caches 64KB, 4-way set associative
- Prefetching assume bandwidth da memória maior e pode ser usado sem penalidade

10. Reduzir Falha ou Penalidade
SW Prefetching de Dados

- Prefetch de Dados
  - Carrega dados em registrador (HP PA-RISC loads)
  - Cache Prefetch: carrega em cache (MIPS IV, PowerPC, SPARC v. 9)
  - Instruções especiais de prefetching não podem causar faltas; uma forma de execução especulativa
- Executando instruções de prefetch leva tempo
  - Custo de prefetch < economia em número menor de misses?

SW Prefetch de Dados

```
/* Before */
for (i=0; i < 3; i = i+1)
  for (j=0; j < 100; j = j+1)
    a[i][j] = b[j][0] * b[j+1][0];
```

Considere:

- Cache com 8KB com blocos de 16 bytes;
- Elementos de a e b: 8 bytes (PF de precisão dupla)
- a: 3 linhas e 100 colunas;
- b: 101 linhas e 3 colunas

Compilador determina quais acessos são passíveis de causar miss na cache.

SW Prefetch de Dados

```
/* Before */
for (i=0; i < 3; i = i+1)
  for (j=0; j < 100; j = j+1)
    a[i][j] = b[j][0] * b[j+1][0];
```

Determinar probabilidade de erros na cache:

- Array a
  - Elementos gravados na ordem que aparecem na memória;
  - Apenas os elementos pares de j são erros;
  - Erros = 3 x 100/2 = 150 erros
- Array b
  - Elementos não são gravados na ordem que aparecem;
  - Possui localidade temporal;
  - 100 erro em b[j+1][0] e 1 erro em b[0][0] = 101 erro

SW Prefetch de Dados

```
/* Before */
for (i=0; i < 3; i = i+1)
  for (j=0; j < 100; j = j+1)
    a[i][j] = b[j][0] * b[j+1][0];
```

Considerações do Prefetch (simplificações):

- Não nos preocupamos com a pré-busca dos primeiros acessos;
- Não nos preocupamos em suprir a pré-busca no fim do loop;
- Penalidade de erro muito grande
  - Não há ganho antes da oitava interação

SW Prefetch de Dados

```
/* After */
for (j=0; j < 100; j = j+1) {
  prefetch (b[j+7][0]);
  prefetch (a[0][j+7]);
  a[0][j] = b[j][0] * b[j+1][0];
}
for (i=1; i < 3; i = i+1)
  for (j=0; j < 100; j = j+1){
    prefetch (a[i][j+7]);
    a[i][j] = b[j][0] * b[j+1][0];
  }
```

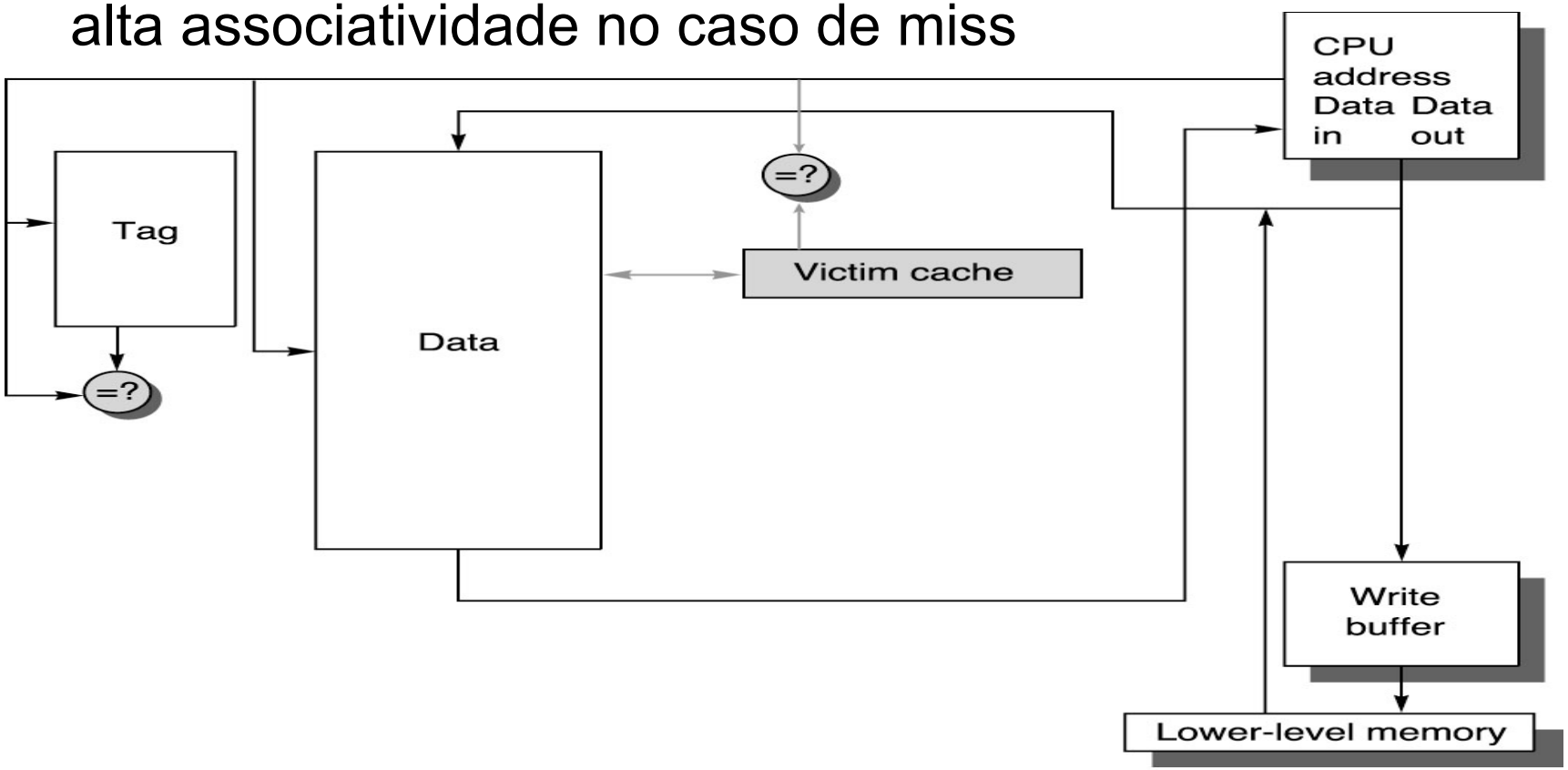
SW Prefetch de Dados

- Resultado…
  - 7 misses para b[0][0],…b[6][0] no primeiro loop
  - 4 misses para a[0][0], a[0][2], … , a[0][6] no primeiro loop
  - 4 misses para a[1][0], …, a[1][6] no segundo loop
  - 4 misses para a[2][0], …, a[2][6] no segundo loops

- Totalizando 19 misses sem prefetching, evitando 232 misses na cache, mas executando 400 instruções de prefetch

11. Reduzir Penalidade Caches de Vítimas

- Reduz Miss Penalty ao inserir uma cache pequena de alta associatividade no caso de miss



Tecnologias de Memória

Production year	Chip size	DRAM Type	Row access strobe (RAS)		Column access strobe (CAS)/ data transfer time (ns)	Cycle time (ns)
			Slowest DRAM (ns)	Fastest DRAM (ns)		
1980	64K bit	DRAM	180	150	75	250
1983	256K bit	DRAM	150	120	50	220
1986	1M bit	DRAM	120	100	25	190
1989	4M bit	DRAM	100	80	20	165
1992	16M bit	DRAM	80	60	15	120
1996	64M bit	SDRAM	70	50	12	110
1998	128M bit	SDRAM	70	50	10	100
2000	256M bit	DDR1	65	45	7	90
2002	512M bit	DDR1	60	40	5	80
2004	1G bit	DDR2	55	35	5	70
2006	2G bit	DDR2	50	30	2.5	60
2010	4G bit	DDR3	36	28	1	37
2012	8G bit	DDR3	30	24	0.5	31

Figure 2.13 Times of fast and slow DRAMs vary with each generation. (Cycle time is defined on page 95.) Performance improvement of row access time is about 5% per year. The improvement by a factor of 2 in column access in 1986 accompanied the switch from NMOS DRAMs to CMOS DRAMs. The introduction of various burst transfer modes in the mid-1990s and SDRAMs in the late 1990s has significantly complicated the calculation of access time for blocks of data; we discuss this later in this section when we talk about SDRAM access time and power. The DDR4 designs are due for introduction in mid- to late 2012. We discuss these various forms of DRAMs in the next few pages.

Arquiteturas Intel

Figure 4. CPU Internals of the Intel® Core™ 2 Duo Processor

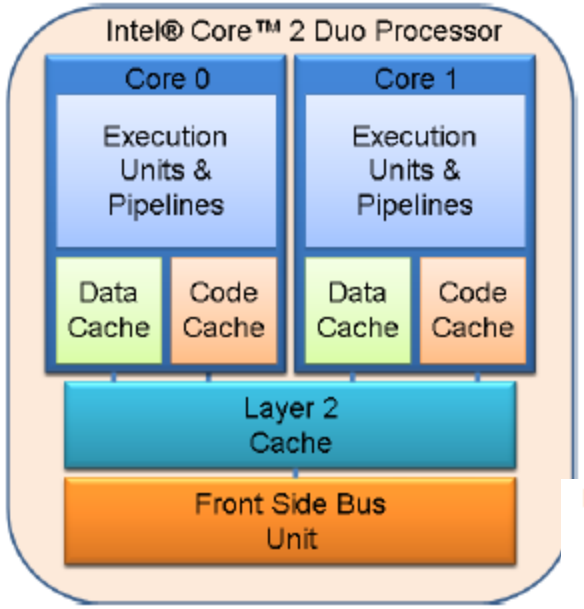
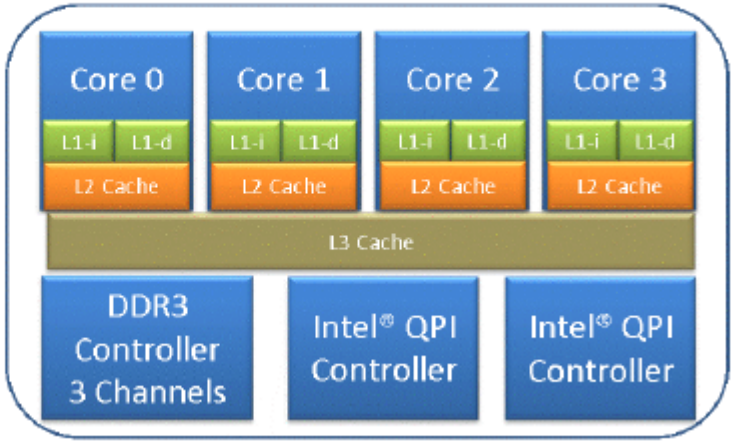


Figure 8. Intel® Core™ i7 Internals



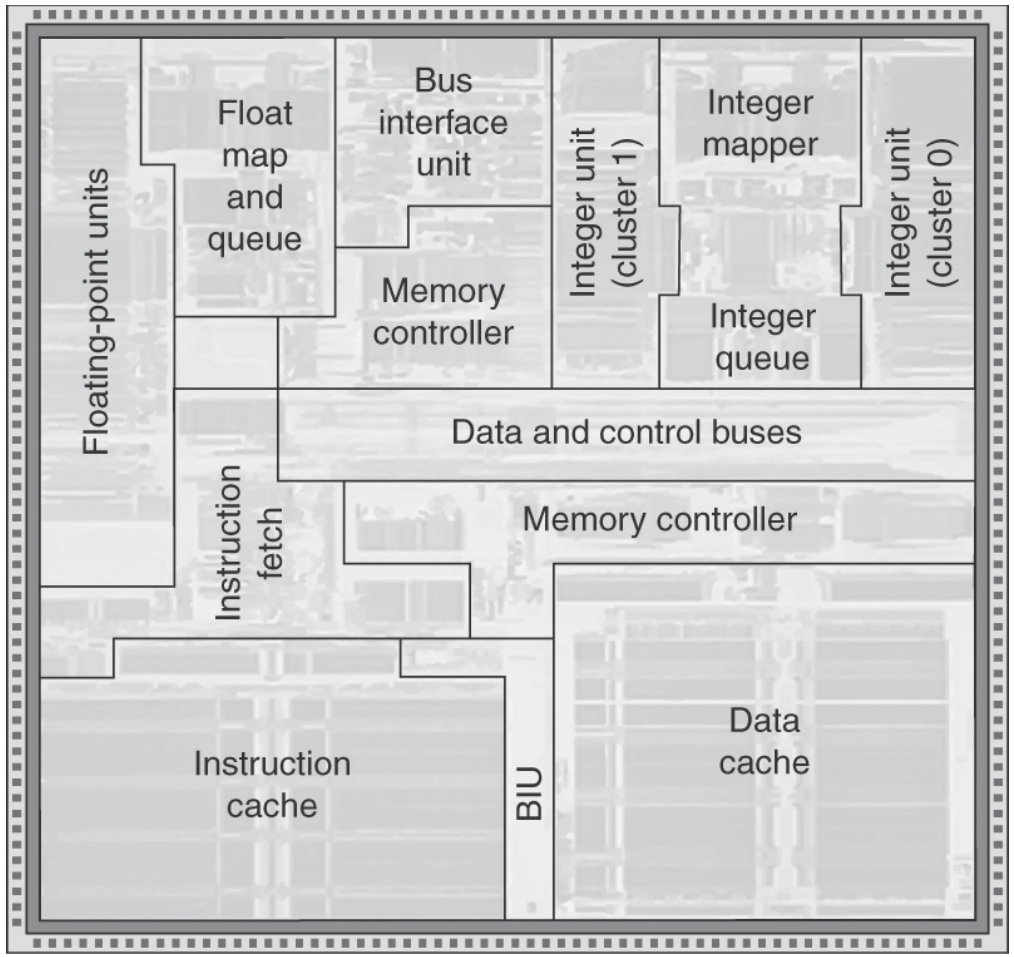


Figure 2.33 Floorplan of the Alpha 21264 [Kessler 1999].