Arquitecturas de

Desenho da Hierarquia da Memória

António Rui Borges

Resumo

- *Porque é que a gestão da memória é tão importante*
  - *Princípio da localidade*
  - *Hierarquia da memória*
- *Cache*
  - *Princípios de Cache*
  - *Desempenho da cache*
  - *Optimização da cache*
- *Memória principal*
  - *RAMs dinâmicas assíncronas*
  - *RAMs dinâmicas síncronas*
- *Leitura sugerida*

Porque é que a gestão da memória é tão importante - 3

Desde os primórdios da era informática, os programadores sonham com quantidades *ilimitadas de* memória *rápida* para armazenar e executar os seus programas, ou seja, sonham em ter tanta memória quanto acharem necessário para armazenar o código e os dados que podem então ser acedidos à taxa máxima que o processador pode operar.

Embora a capacidade de *memória principal* tenha aumentado constantemente ao longo dos anos, o facto é que os programas tendem a expandir-se em tamanho e complexidade preenchendo todo o espaço disponível - *Lei de Parkinson*.

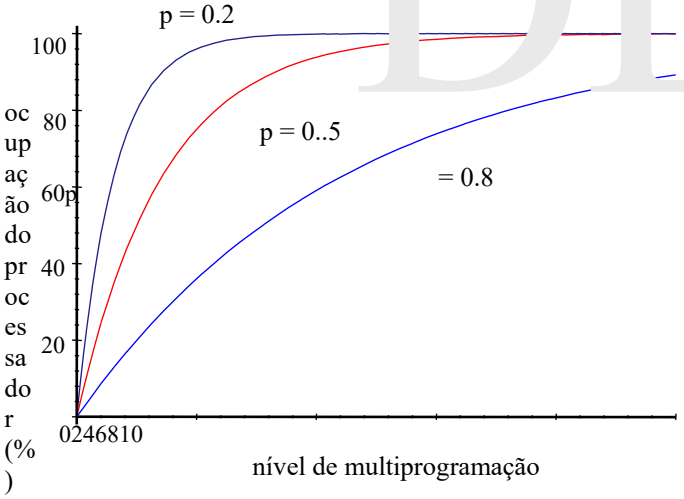
Isto é verdade não só porque os programas se tornaram mais complexos à medida que o desempenho dos sistemas informáticos melhora, lidando com problemas que não poderiam ter sido enfrentados anteriormente, mas também porque é fundamental num ambiente multitarefa armazenar em conjunto na memória principal os espaços de endereçamento de muitos processos para que o processador seja mantido totalmente ocupado e os tempos de *resposta* e/ou de *retorno* associados sejam minimizados.

Porque é que a gestão da memória é tão importante - 4

fração da ocupação do processador = 1 - p ^{n} (modelo simplificado)

p - fração do tempo que um processo espera bloqueado para a integridade das operações de E/S, sincronização ou qualquer outra causa

n - número de processos que coexistem actualmente na memória principal



N. de processos no MP	*% de ocupação (P)*
4	59
8	83
12	93
16	97

p = 0,8

Princípio da localidade - 5

Não é possível, contudo, satisfazer literalmente o sonho dos programadores.

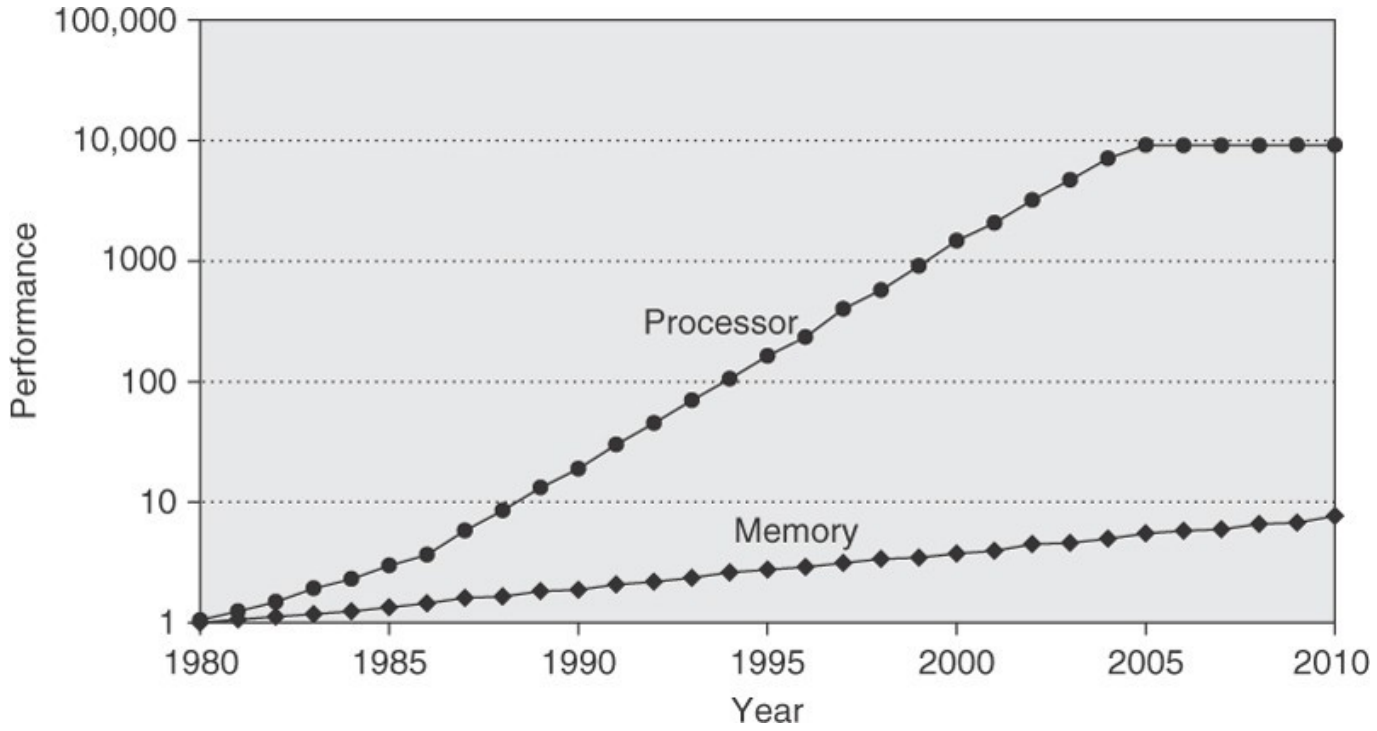
Pode-se demonstrar que, para uma dada tecnologia de implementação e um determinado orçamento de energia, é possível fazer hardware mais simples e mais rápido. À medida que a complexidade aumenta, os atrasos na propagação do sinal tornam-se mais longos devido ao aumento da quantidade necessária de circuitos de interconexão e à diminuição da intensidade das correntes eléctricas de condução.

Além disso, o problema tornou-se mais grave à medida que o tempo foi passando. Em vez de ser reduzido, o défice de desempenho do processador-DRAM aumentou gradualmente ao longo das últimas décadas, tendo mesmo piorado desde a introdução dos processadores multicondutores, onde a largura de banda máxima agregada é proporcional ao número de processadores no núcleo.

Princípio da localidade - 6

Ao longo do tempo variação de desempenho de um único processador vs. memória

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



curva do processador - número médio de pedidos de memória por segundo
curva de memória - inversa da latência de acesso DRAM

Princípio da localidade - 7

Para lidar com este problema, a abordagem seguida pelos designers de sistemas de memória baseia-se num facto observacional derivado do rastreio da execução do programa e conhecido como o *princípio da localidade*. Afirma simplesmente que, durante períodos de tempo relativamente grandes, um programa tende a referir uma fracção muito definida do seu espaço de endereçamento. Assim, pode-se falar de

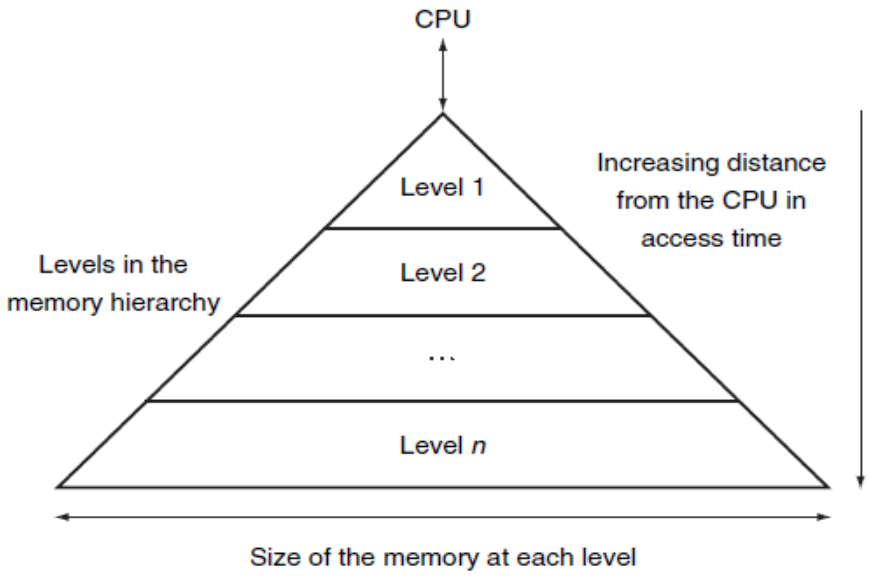
- *localidade espacial* - quando uma palavra de memória é referenciada uma vez, outra, que é armazenada nas proximidades, pode ser referenciada em breve
- *localidade temporal* - quando uma palavra de memória é referenciada uma vez, poderá ser novamente referenciada em breve.

Não é muito difícil perceber porque é que os programas têm este tipo de comportamento. Tente afirmar as razões por que o é!

Hierarquia da memória - 8

Para tirar partido do princípio da localidade, a memória do computador é implementada como uma *hierarquia de* memória, ou seja, consiste em múltiplos níveis de memória com diferentes tamanhos e velocidades de acesso. Como regra, os módulos de memória mais rápidos têm um preço por bits mais elevado do que os mais lentos e uma menor capacidade de armazenamento.

O objectivo é apresentar ao programador tanta memória como a que está disponível na tecnologia mais barata (nível mais baixo), ao mesmo tempo que proporciona acesso à velocidade oferecida pela mais cara (nível mais alto).



Hierarquia da memória - 9

Fonte: Organização e Projeto de Computadores: A interface Hardware/Software

Hierarquia da memória - 10

A hierarquia da memória pode ser dividida em níveis distintos

- *banco de registo* - memória interna ao processador, o acesso é controlado pelas instruções
- *cache* - memória externa ao processador, o acesso é controlado por *instruções de busca* e *transferência de dados*; consiste actualmente em vários níveis de RAM estática, todos eles colocados dentro do circuito integrado do processador; o primeiro nível está mesmo localizado dentro do chip do processador e é dividido em instruções e unidades de dados
- *memória principal* - memória externa para o processador; implementa o *conceito de programa armazenado* que define o dispositivo onde as instruções e os dados de um programa de execução são maioritariamente armazenados; consiste em RAM dinâmica
- *área de troca* - memória não volátil localizada em armazenamento de massa; funciona como uma extensão da memória principal para implementar uma organização de memória controlada do sistema operativo, normalmente uma *arquitectura de memória virtual paged-*;

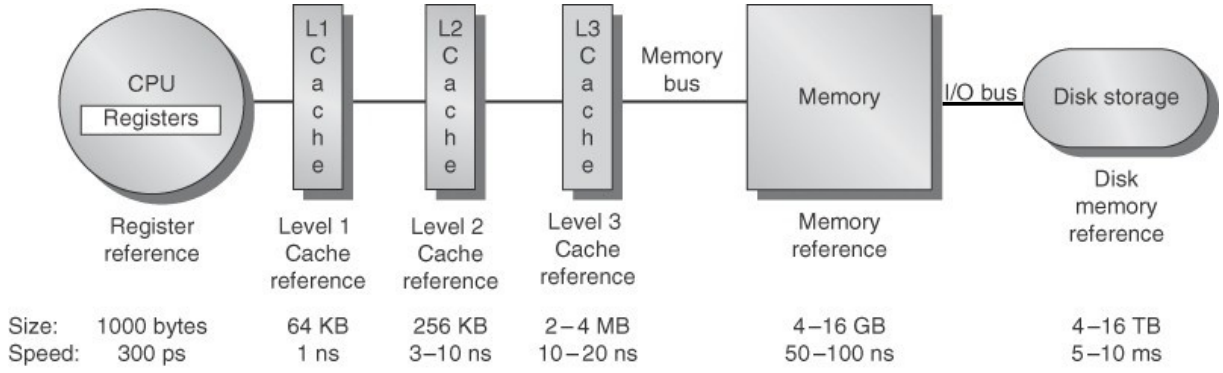
Hierarquia da memória - 11

consiste principalmente em dispositivos de memória HDD ou flash.

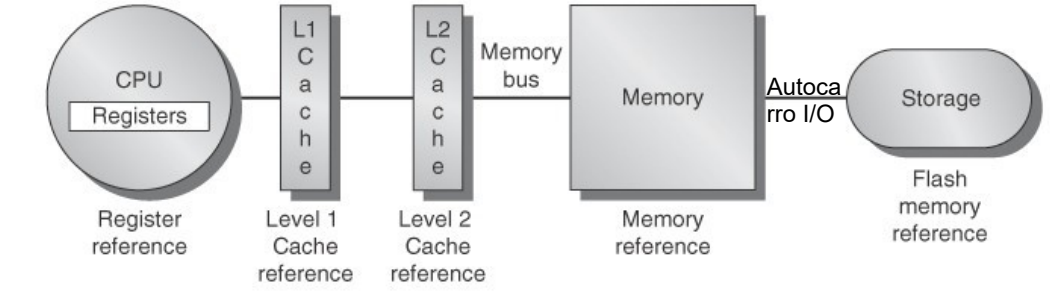
Hierarquia da memória - 12

Níveis de uma hierarquia de memória típica

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



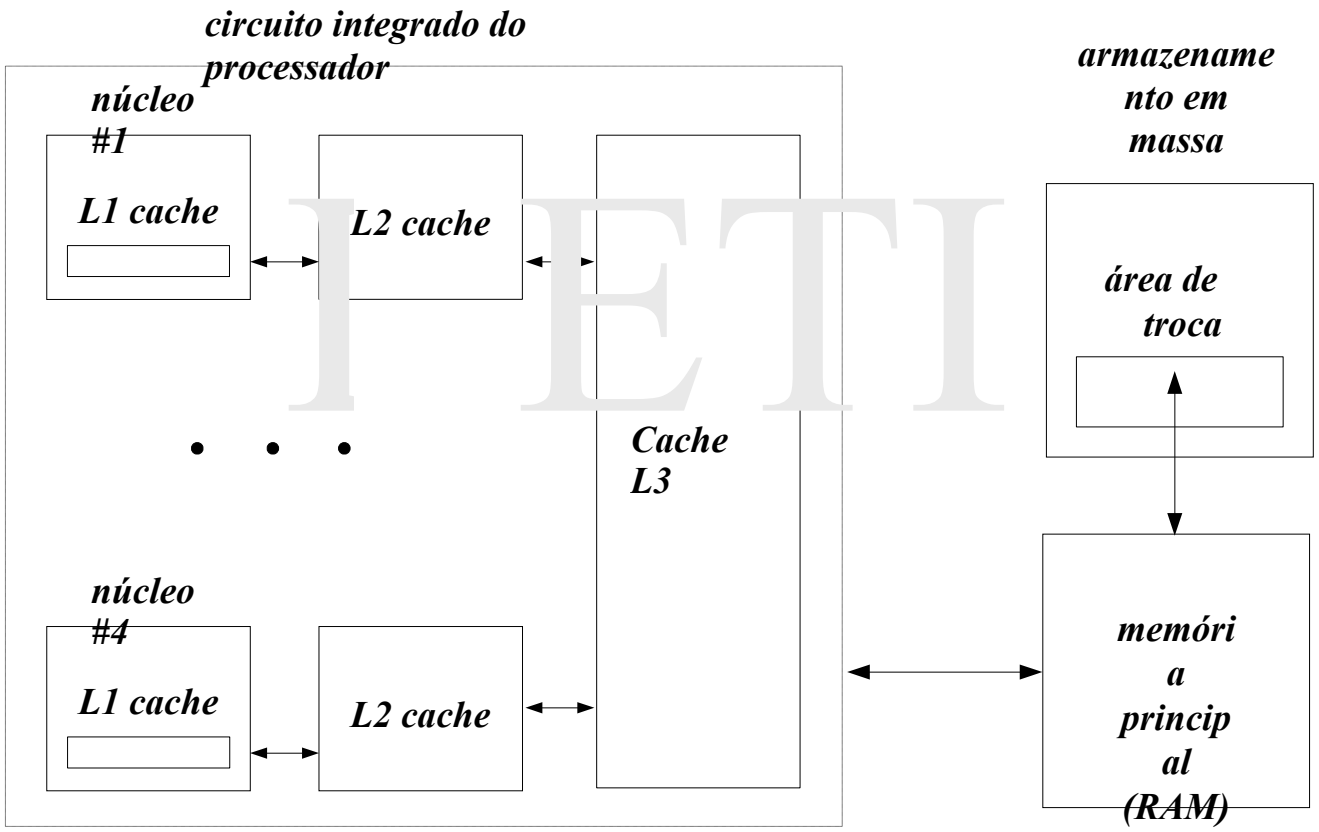
(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Hierarquia da memória - 13

Hierarquia típica de memória para um sistema processador de 4 núcleos



Hierarquia da memória - 14

Especificações técnicas de uma hierarquia de memória típica

Fonte: adaptado de Computer Architecture: Uma Abordagem Quantitativa

Nome	*Tamanho típico*	*Tecnologia de implementação*	*Tempo de acesso (ns)*	*Largura de banda (MB/s)*	*Gerido por*	*Com o apoio de*
banco de registo	menos 1KB	design personalizado multiportos - CMOS	0,15 – 0,30	10^5 – 10^6	compilador	cache
cache	32 KB - 8 MB	CMOS SRAM on/off-chip	0,5 – 15	10^4 – 4 x 10^4	hardware	memória principal
memória principal	menos 512 KB	CMOS DRAM	30 – 200	5 x 10^3 - 2 x 10^4	sistema operativo	área de troca

Hierarquia da

área de troca	maior 1 TB	semicondutor / magnético	3×10^4 - 5×10^6	50 – 500	sistema operativo	armazenamento a longo prazo

Hierarquia da memória - 16

Numa hierarquia de memória, quanto mais alto for um nível, mais próximo do processador que se encontra. As hierarquias de memória tiram partido da *localização temporal*, mantendo as instruções e dados acedidos mais recentemente mais perto do processador, e da *localização espacial*, movendo blocos constituídos por múltiplas palavras de memória contíguas para níveis superiores da hierarquia.

Na maioria dos sistemas, a memória constitui uma verdadeira hierarquia, ou seja, para que os dados estejam presentes no nível *i*, tem de estar presente primeiro no nível *i+1*, e todos os dados estão presentes no nível mais baixo.

Os conceitos subjacentes à construção de sistemas de memória afectam muitos outros aspectos para além da arquitectura informática. Estes incluem como o sistema operativo gere a memória e E/S, como os compiladores geram código e mesmo como as aplicações utilizam os recursos computacionais.

Como todos os programas passam grande parte do seu tempo a aceder à memória, o subsistema de memória é um factor determinante para o desempenho. Por conseguinte, os programadores precisam hoje em dia de compreender que a memória é hierárquica para melhorar o desempenho dos seus programas.

Hierarquia da memória - 17

Embora uma hierarquia de memória possa consistir em vários níveis, a transferência de dados ocorre geralmente apenas entre dois níveis adjacentes de cada vez, pelo que se pode concentrar no que acontece entre qualquer par de níveis para obter uma visão geral das operações.

A quantidade mínima de dados que pode estar presente ou ausente numa hierarquia de dois níveis é chamada *bloco*. Dizemos que ocorre um *acerto* quando os dados solicitados pelo processador aparecem em algum bloco no nível superior; caso contrário, o pedido é chamado de *falta* e o nível inferior é então acedido para recuperar o bloco que contém os dados solicitados.

A *taxa de acerto* ou o *rácio de acerto* é a fracção de referências de memória aos dados encontrados no nível superior sobre todas as referências de memória. Por outro lado, a taxa de *acerto* ou o *rácio de acerto* é o seu complemento a 1.

Uma vez que o desempenho é a questão principal, o tempo necessário para a manutenção de hits e misses é relevante. O *tempo de acerto*, sendo o tempo de acesso ao nível superior, compreende também o tempo para afirmar se o acesso é um acerto ou um erro. A *penalidade de erro*, portanto, é o tempo para substituir um bloco no nível superior pelo bloco que contém os dados solicitados pelo

Hierarquia da memória - 18

processador.

Princípios de Cache - 19

Cache é o nome tradicionalmente dado ao(s) nível(is) de hierarquia de memória localizado(s) entre o processador e a memória principal. Hoje em dia, porém, o termo tem um significado mais amplo: refere-se a qualquer dispositivo de armazenamento que seja gerido de uma forma que tire partido do princípio da localidade.

Ao lidar com a cache, o termo *linha* é utilizado especificamente para se referir à quantidade mínima de informação que é transferida entre qualquer par de níveis de cache ou armazenada ao nível de cache mais baixo. O *bloco* é reservado para se referir aos próprios dados armazenados numa linha de cache ou na memória principal.

Assumindo um único nível de cache, a resposta às seguintes perguntas ajudará a iluminar a forma como um cache funciona

- onde está uma linha localizada na cache? (*colocação da linha*)
- como é encontrado se estivesse presente? (*identificação de linha*)
- que linha deve ser alterada em caso de falha? (*substituição de linha*)
- o que acontece numa operação de escrita? (*estratégia de escrita*).

Princípios de Cache - 20

Uma vez que a capacidade de memória principal é muito maior do que o tamanho da cache, muitos blocos de memória irão sobrepor-se no mesmo local dentro da cache ao longo do tempo. Há várias maneiras de o fazer, mas é preciso ter em mente que os principais objectivos são manter o hardware simples e todo o procedimento de armazenamento/acesso eficiente.

Neste sentido, o tamanho do bloco não deve ser completamente arbitrário. É importante que o número de bytes armazenados seja uma potência de 2 para que um *endereço de memória* possa ser trivialmente dividido em um *endereço de bloco* e um *offset*.

Em geral, a cache pode ser organizada como

- *mapeado directamente* - quando existe um único local onde um *bloco* pode ser colocado
- *totalmente associativo* - quando um *bloco* pode ser colocado em qualquer lugar
- *conjunto associativo* - quando há um agregado de lugares, chamado *conjunto*, onde um

  pode ser colocado *um bloco.*

Princípios de Cache

21

Mapeado directamente a organizações totalmente associativas formam um continuum de níveis de associatividade definidos: o tamanho definido para o *mapeamento directo* é um e para a *associatividade total* é igual ao número de linhas na cache.

Princípios de Cache - 22

endereço de memória

endereço de bloco (*s* bits)	offset (*w* bits)

Caracterização da memória principal / cache

comprimento do endereço = $s + w$ bits

número de unidades endereçáveis na memória principal = 2 bytes^{s+w}

número de blocos na memória principal = 2^{s}

número de linhas na cache = $m = 2^{r}$

tamanho da cache = 2 bytes^{r+w}

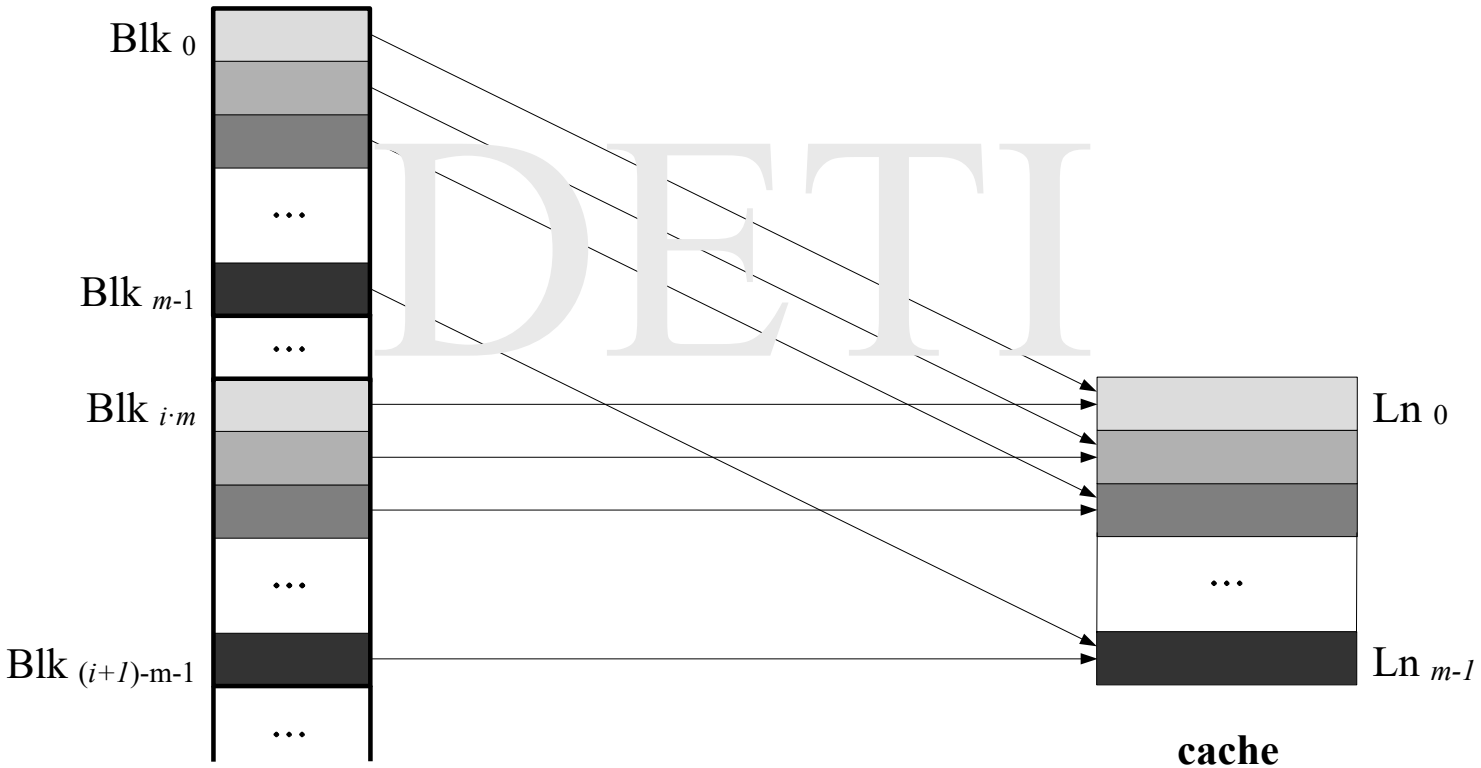
número de conjuntos na cache = $v = 2^{u}$

número de linhas por conjunto = $k = m \,/\, v = 2^{r-u}$

Princípios de Cache - 23

Cartografia directa

endereço da linha da cache = número de linhas do bloco de endereços na cache



Princípios de Cache - 24

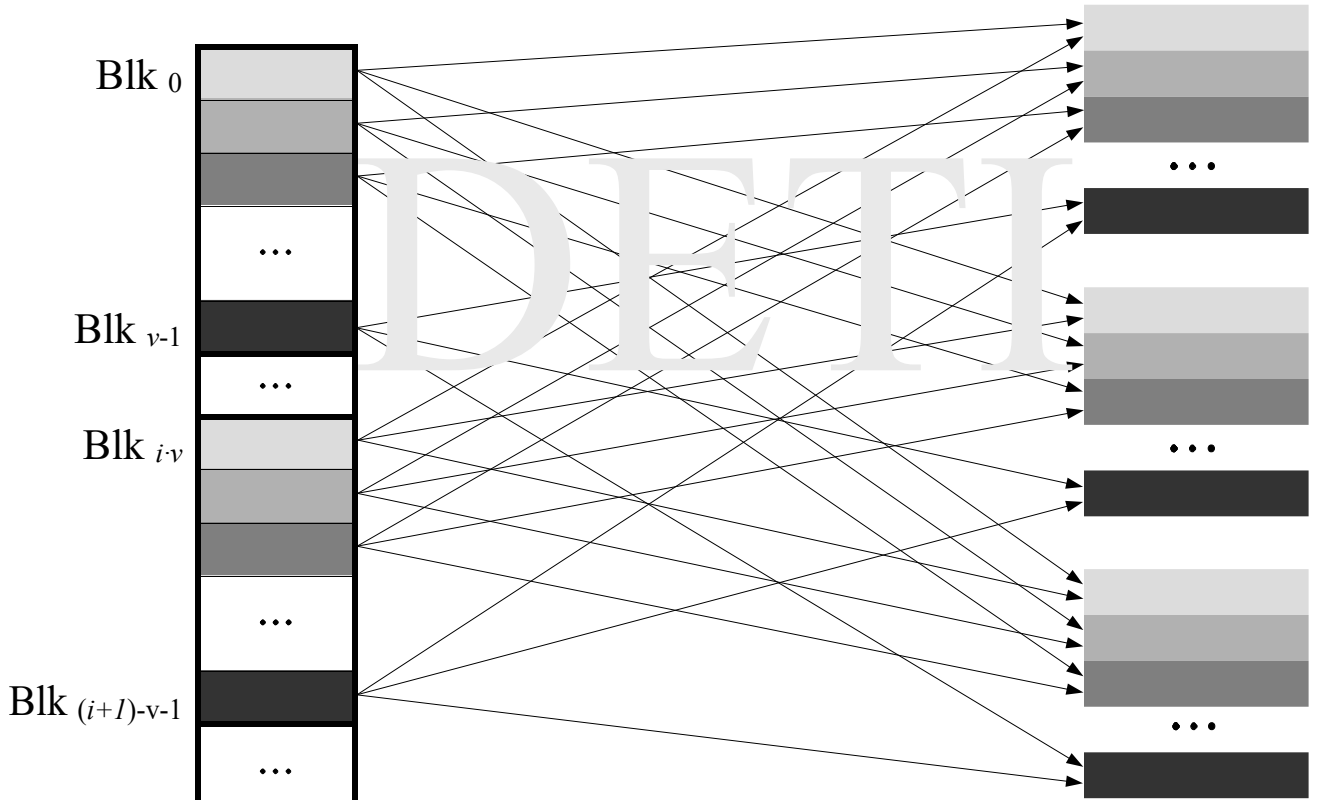
Associatividade total

endereço da linha de cache = qualquer

Princípios de Cache - 25

Associatividade de conjuntos

endereço da linha de cache = número de endereços de bloco de conjuntos na cache



Princípios de Cache - 26

Ln 0
Ln *1*
Ln *2*

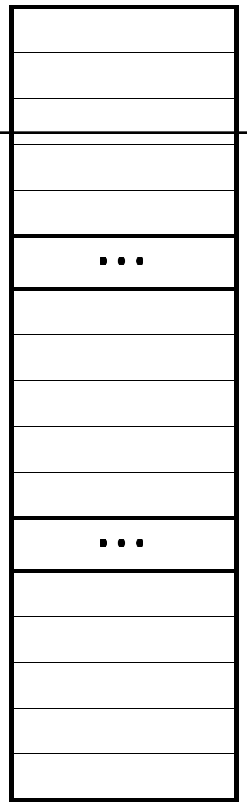
Ln v-1

Ln *j-v* Ln *j-v+1* Ln *j-v+2*

Ln (j+1)-v-1

Ln (*k-1*)-v Ln (*k-1*)-v+1 Ln (*k-1*)-v+2

Ln *k-v-1*



Princípios de Cache - 27

Para além do conteúdo de um bloco de memória principal, uma linha de cache também deve conter pelo menos parte do seu endereço, normalmente chamado *campo de etiqueta de* endereço do bloco. O *campo tag* é armazenado em cada linha da cache para que possa ser verificado em relação à parte correspondente do endereço de memória gerado pelo processador para determinar se existe uma correspondência quando ocorre um acesso. Como regra, todas as *etiquetas* possíveis são verificadas em paralelo para tornar a pesquisa rápida. Além disso, uma vez que um registo de dados nunca está vazio, é necessário um bit extra (chamado *bit de validação*) para afirmar se os dados actualmente armazenados na linha são significativos ou não.

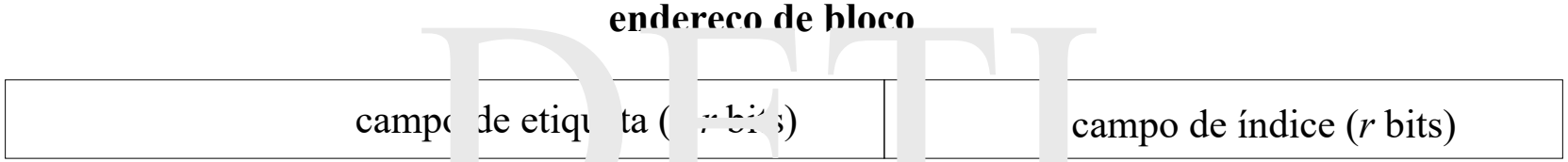
Sempre que o *campo tag* de um endereço de bloco não for o endereço de bloco completo, os bits restantes do endereço formam um segundo campo, o *campo índice*, cujo objectivo é seleccionar um conjunto específico dentro da cache.

endereço de bloco

campo de etiqueta (*s*-*u* bits)	campo de índice (*u* bits)

Princípios de Cache - 28

Cartografia directa



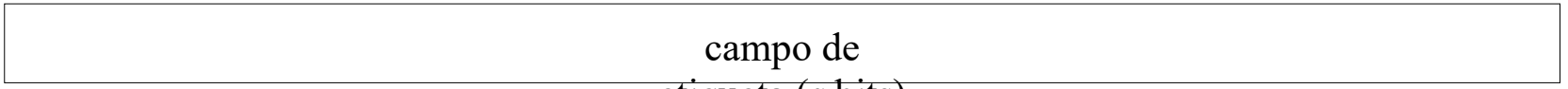
Quando a cache é organizada através de *mapeamento directo*, há uma única linha dentro da cache onde um determinado bloco de memória pode ser armazenado. Isto significa que o número de linhas por conjunto é igual a uma, o número de conjuntos é igual ao número de linhas e o campo de etiqueta contido em cada linha tem um comprimento mínimo.

Esta organização leva a implementações muito simples, rápidas e eficientes. A principal desvantagem é o risco de *surra:* um fenómeno que surge quando a fracção do espaço de endereçamento referenciado pelo processador num período de tempo prolongado contém grupos de dois ou mais endereços que mapeiam para as mesmas linhas de cache. Quando isto acontece, a taxa de acerto degrada-se bastante e a execução do programa torna-se bastante lenta porque nenhum benefício é retirado da localidade de referência.

Princípios de Cache - 29

Associatividade total

endereço de bloco



Quando a cache é organizada através de uma *associatividade total*, todas as linhas dentro da cache estão disponíveis para o armazenamento de um determinado bloco de memória. Isto significa que o número de linhas por conjunto é igual ao número de linhas na cache, o número de conjuntos é igual a um e o campo de etiqueta contido em cada linha tem o comprimento máximo (o campo de índice não existe).

Esta organização leva à minimização da taxa de falhas, uma vez que, em princípio, um bloco de memória específico pode ser armazenado em qualquer uma das linhas da cache. A principal desvantagem é a complexidade de concepção que implica e, devido a isso, para uma dada tecnologia de implementação e um determinado orçamento de energia, limita a velocidade de

tomar uma decisão por um acerto ou um erro.

Princípios de Cache - 30

Princípios de Cache - 31

Associatividade de conjuntos

endereço de bloco

Quando a cache é organizada através da *associatividade do conjunto* (*k-way* cache), existe uma linha actly *k* dentro da cache onde um determinado bloco de memória pode armazenar os seus meios de que o número de linhas por conjunto é igual a *k* e o número de se

Esta organização tenta alcançar *o melhor do mundo,* tal como retratado pelas duas organizações anteriores. Conduz a implementações não demasiado complexas que ainda são rápidas e eficientes e evita o risco de *destruição*, fornecendo alguma redundância para onde um bloco de memória possa ser armazenado.

Princípios de Cache - 32

Quando ocorre uma falha, o controlador de cache deve seleccionar uma linha cujo bloco será substituído com os dados desejados. Com a organização *directamente mapeada*, o problema é trivial, uma vez que apenas uma linha é verificada para um acerto e apenas o conteúdo desta linha pode ser modificado. Com organizações associativas totalmente associativas e definidas, existem em princípio muitas, ou pelo menos algumas, linhas a serem consideradas. Se o *bit de validação* para qualquer destas linhas for reiniciado, uma delas pode ser seleccionada, mas após algum tempo todas elas conterão dados válidos e, por isso, deve ser tomada uma decisão.

A estratégia óbvia, a que minimiza a taxa de erro, é escolher a linha dentro do grupo cujos dados já não serão referenciados ou, se o forem, a referência acontecerá à distância mais distante do presente - o *princípio da optimização*. Infelizmente, esta regra não é causal, exigiria adivinhar o futuro e, portanto, não pode ser implementada na prática.

Princípios de Cache - 33

As principais estratégias utilizadas para a selecção de linhas são

- *aleatório* - um gerador pseudo-aleatório é utilizado para distribuir a substituição uniformemente entre as linhas candidatas; isto potencia um comportamento reprodutível
- *menos recentemente utilizada* (LRU) - a fim de aproximar o princípio da optimização, é preciso contar com o passado para prever o futuro; assim, a linha de candidatos é a que não foi referenciada durante o período de tempo mais longo
- *primeiro a entrar, primeiro a sair* (FIFO) - porque a LRU leva a uma implementação complexa, uma aproximação a ela que é mais simples, mas que ainda depende do passado para prever o futuro, é considerar a linha cujo conteúdo permaneceu durante o mais longo período de tempo na cache.

Princípios de Cache - 34

Falhas no cache de dados por 1000 instruções para a arquitectura Alfa (DEC) usando 10 referências SPEC2000 (5 SPECint2000 e 5 SPECfp2000)

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa

Associatividade do conjunto (tamanho do bloco = 64 bytes)									
	2 vias			**4 vias**			**8 vias**		
Tamanho da cache	**Aleatório**	**LRU**	**FIFO**	**Aleatório**	**LRU**	**FIFO**	**Aleatório**	**LRU**	**FIFO**
16 KB	117,3	114,1	115,5	115,1	111,7	113,3	111,8	109,0	110,4
64 KB	104,3	103,4	103,9	102,3	102,4	103,1	100,5	99,7	100,3
256 KB	92,1	92,2	92,5	92,1	92,1	92,5	92,1	92,1	92,5

Princípios de Cache - 35

As operações de *leitura* a partir da memória dominam os acessos à cache do processador. Isto é assim porque todas as *instruções de busca* são operações de leitura e a maioria das instruções não escrevem na memória. De facto, 10 referências SPEC2000 (5 SPECint2000 / 5 SPECfp2000), executadas num processador MIPS, sugerem uma média de 26% / 30% de instruções de *carga* e de 10% / 9% de instruções de *loja* sobre todas as instruções executadas [Computer Architecture: A Quantitative Approach], o que leva a uma proporção de operações de *leitura* sobre *escrita* de 93:7 / 94:6.

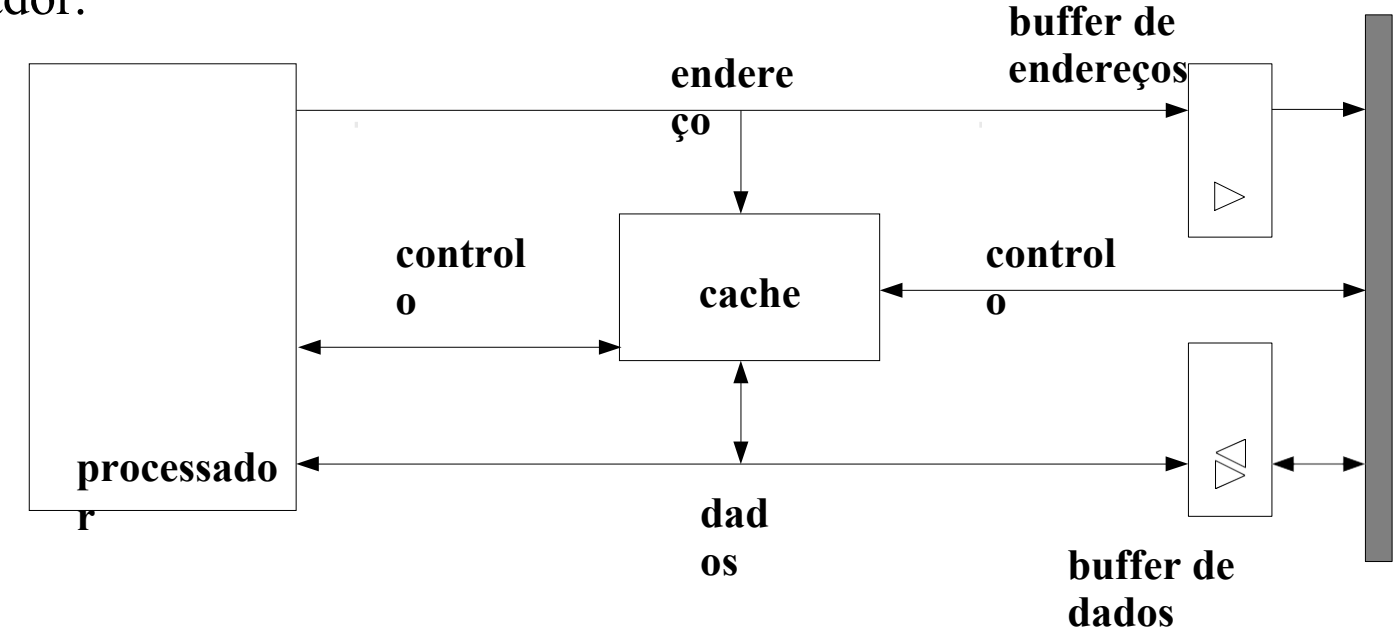
Fazer o *caso comum rapidamente* significa optimizar as caches para leituras, especialmente porque tradicionalmente os processadores esperam pela conclusão das leituras, mas não precisam de esperar pela conclusão das escritas. Uma operação *de leitura* pode começar assim que o endereço do bloco contendo os dados referenciados for disponibilizado. Ao mesmo tempo, as etiquetas das linhas candidatas são comparadas com o *campo da etiqueta* do endereço do bloco para determinar se há um acerto. Se assim for, a parte solicitada do bloco é imediatamente transmitida ao processador; caso contrário, é iniciada a transferência do bloco do nível inferior e a operação de leitura é interrompida até

Princípios de Cache - 36

que a transferência seja concluída.

Princípios de Cache - 37

A fim de optimizar ainda mais as caches para operações de leitura, as organizações contemporâneas ligam o par processador-cache via endereço, dados e linhas de controlo. As linhas de endereço e de dados também se ligam aos buffers de endereço e de dados que medeiam o acesso ao bus do sistema a partir do qual a memória principal é alcançada. Quando ocorre um hit, os buffers de endereços e de dados são desactivados e toda a comunicação do sistema é ignorada. Quando ocorre um erro, o bloco é carregado no cache e aos dois computador e o processador.



do sis te m a

Fonte: Organização de Computadores e Desenho de Arquitectura para o Desempenho

Princípios de Cache - 38

Princípios de Cache - 39

As operações de *escrita* são um pouco diferentes. A modificação do conteúdo do bloco só pode começar depois de a etiqueta ser verificada para determinar se há um acerto. Assim, as operações de escrita demoram normalmente mais tempo do que as operações de leitura. Além disso, embora o processador especifique sempre o tamanho e localização dos dados, apenas para operações de escrita isto é crítico porque uma parte definida do conteúdo do bloco é alterada; para operações de leitura, o acesso a mais bytes de dados do que é necessário, é irrelevante.

Existem duas *políticas* básicas *de* escrita

- *escrita* - os dados são escritos tanto para a linha de cache como para o bloco no nível inferior
- *write-back* - os dados são escritos na linha da cache; o conteúdo do bloco de linhas só é escrito no nível inferior quando o bloco é substituído.

Quando a política de write-back é implementada, uma característica chave, conhecida como a *parte suja do* bloco, é normalmente utilizada para garantir que apenas os blocos modificados são transferidos de volta na substituição. Quando um bloco é transferido pela primeira vez para a cache, o seu bit de estado assume o valor *limpo* para assinalar que o seu conteúdo não foi alterado; quando ocorre uma escrita, o bit de estado assume então o valor *sujo* e o bloco deve ser escrito de volta

Princípios de Cache - 40

aquando da substituição.

Princípios de Cache - 41

Vantagens de *escrita*

- é mais simples de implementar; uma vez que todas as operações de escrita resultam numa escrita para o nível inferior, as linhas de cache são sempre limpas
- o conteúdo actualizado do bloco está sempre presente no nível inferior, o que simplifica a garantia de coerência dos dados
- desempenha um papel importante na concepção de caches de vários níveis; para os níveis superiores, os escritos só precisam de se propagar para o nível inferior seguinte em vez de se propagarem até à memória principal.

Vantagens de *retorno escrito*

- as operações de escrita para hits ocorrem à velocidade da cache e múltiplas escritas no mesmo bloco requerem apenas uma escrita de volta ao nível inferior quando o bloco é substituído
- é utilizada menos largura de banda de memória, tornando-a atractiva para multiprocessadores
- também se poupa energia, tornando-a atractiva para aplicações incorporadas.

Princípios de Cache - 42

O processador bloqueia a conclusão das operações de escrita durante um procedimento de *escrita.* Uma optimização comum para reduzir as paragens de escrita é implementar um *buffer de escrita*, que permite ao processador continuar assim que os dados são escritos no buffer, sobrepondo de facto a execução do processador com a actualização da memória.

Um *erro de escrita* pode ser tratado das seguintes formas

- *write allocate* - uma linha é atribuída sempre que ocorre uma falha, então a operação de escrita tem lugar; contudo, se o tamanho do bloco for maior do que os dados a serem escritos, o bloco deve primeiro ser recuperado do nível inferior
- *alocação não escrita* - a cache não é afectada por falhas, a operação de escrita ocorre apenas no nível inferior, o que significa que os blocos ficam fora da cache até que o processador leia os dados dos mesmos.

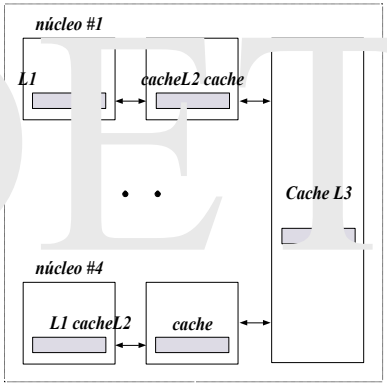
Qualquer uma das *políticas de escrita* pode ser utilizada com qualquer uma das *políticas de escrita*. No entanto, *as caches de reintegração* implementam geralmente a política *de atribuição de escrita* na esperança de que as escritas subsequentes a esse bloco sejam apanhadas pela cache. Da mesma forma, os *caches de write-back*

Princípios de Cache - 43

implementam a política de *atribuição de não escrita* porque todas as escritas devem ser escritas no nível inferior, pelo que não se obtém qualquer ganho com a atribuição do bloco.

Princípios de Cache - 44

Para um processador de múltiplos núcleos, onde múltiplos fios simultâneos podem estar a competir pelo acesso a dados partilhados, protegidos ou não por regiões críticas, surge um outro problema que tem a ver com a *coerência do cache*. Em tal situação, cópias do mesmo bloco de memória podem ser armazenadas em linhas de caches de nível 1 ou nível 2 associadas a diferentes processadores.



A fim de assegurar que todos os processadores vejam sempre os mesmos dados, deve ser implementada uma política de *gravação* para as caches de nível 1 e nível 2 e, quando uma *gravação* tem lugar, todas as cópias a estes níveis devem ser feitas *fora do prazo*, de modo a que uma transferência a partir da cache de nível 3 seja efectuada se surgir uma *leitura* subsequente.

Princípios de Cache - 45

Durante a execução de um programa, o processador não só vai buscar instruções, mas também acede à memória para carregar e armazenar operandos. Embora uma cache única, *unificada* ou *mista* possa fornecer ambas, pode gerar um estrangulamento de comunicação. Uma situação típica em que isto acontece é quando um processador pipelined executa uma instrução de *carga* ou de *armazenamento* e, ao mesmo tempo, vai buscar a seguinte. A fim de evitar este tipo de risco estrutural, é comum hoje em dia ter ao mais alto nível duas caches: uma dedicada a instruções e outra a dados. Assim, a largura de banda de comunicação com memória pode ser duplicada através da utilização de portas separadas.

As caches dedicadas também permitem a afinação individual de cada cache, especificando diferentes capacidades de armazenamento, tamanhos de blocos e associatividades para cada um.

O cache de dados de Opteron - 46

A cache de dados Opteron tem um tamanho de 64KB e está organizada como uma cache associativa de 2 vias, capaz de armazenar blocos de dados de 64 bytes. Inclui uma estratégia de substituição da LRU e implementa uma política de *write-back* com *write alocações de write* on write miss.

O Opteron apresenta um endereço de memória de 40 bits para a divisão da cache como se segue.

endereço de memória

endereço de bloco (34 bits)	offset (6 bits)

comprimento de endereço = 40 bits

número de unidades endereçáveis na memória principal = 2 bytes ^{40}/ 2

palavras ^{37}número de blocos na memória principal = 2 ^{34}

tamanho do bloco = 2 = ^{6}64 bytes

O cache de dados de Opteron - 47

Para calcular o número de linhas na cache, fazemos

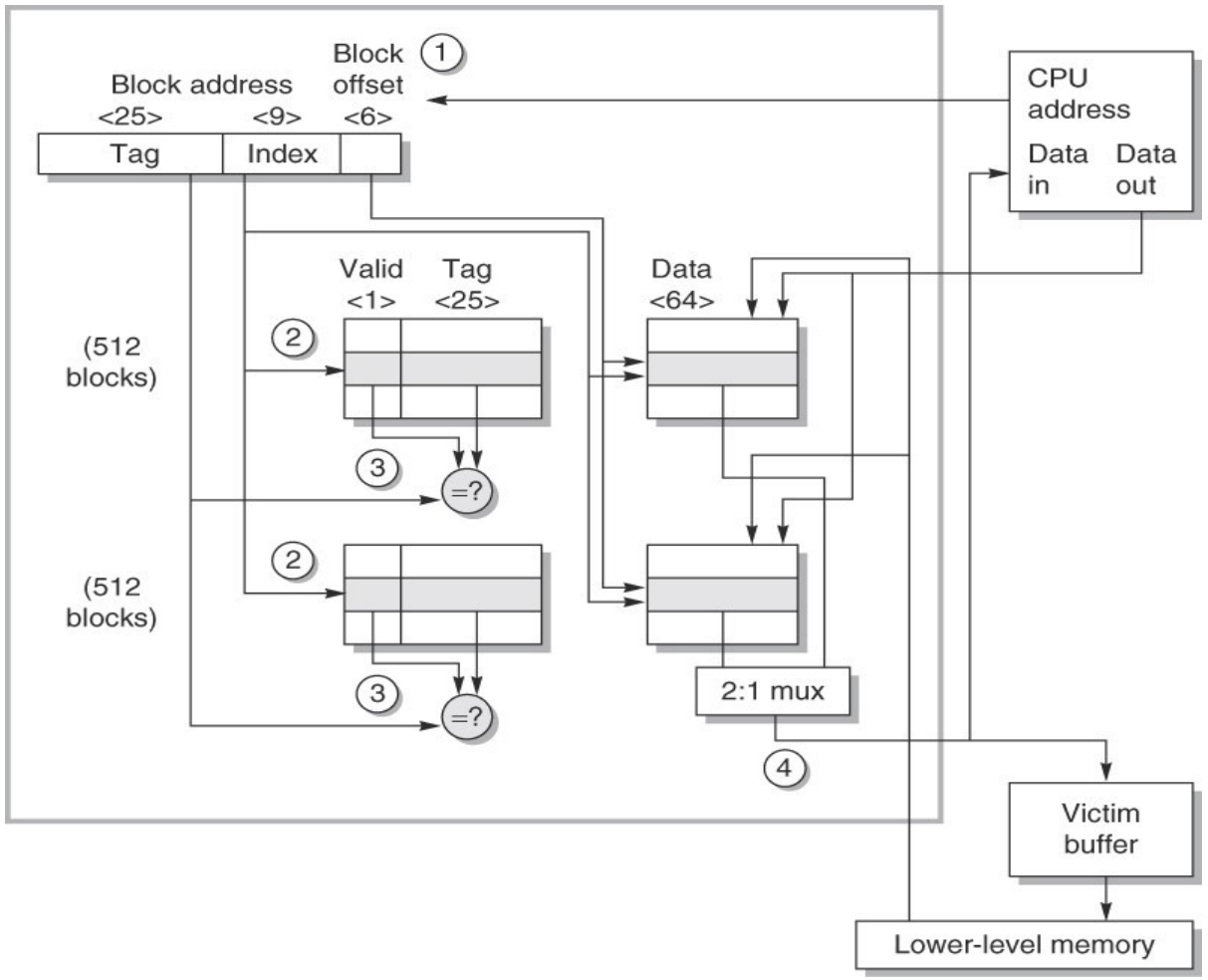
$$m = \frac{\text{tamanho da cache}=}{\text{associatividade do conjunto} \cdot \text{tamanho do bloco}} \qquad \frac{2^{169}}{2 \cdot 2^{6}} = 2 \quad = 512$$

endereço de bloco

campo de etiqueta (25 bits)	campo de índice (9 bits)

Um ponto digno de nota é que, como Opteron é um processador de 64 bits, cada acesso à memória tem no máximo 8 bytes de largura. Portanto, para além dos 9 bits do *campo índice* para seleccionar o bloco de memória adequado, os 3 bits mais significativos do *offset do bloco* são também utilizados para a selecção da palavra adequada dentro do bloco.

O cache de dados de Opteron - 48



Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa

O cache de dados de Opteron - 49

Várias etapas podem ser destacadas num acesso à cache

- *passo* 1 - o endereço de memória gerado pelo processador é dividido em duas partes: o *endereço do bloco*, os 34 bits mais significativos, que faz referência a um determinado bloco de memória, e o *offset do bloco*, os 6 bits menos significativos, que faz referência a um byte específico dentro do bloco
- *passo* 2 - o endereço do bloco, por sua vez, é também dividido em duas partes: o *campo tag*, os 25 bits mais significativos, que serve para determinar se o bloco está presente na cache, e o *campo index*, os 9 bits menos significativos, que faz referência ao conjunto dentro da cache onde o bloco pode ser armazenado
- *passo* 3 - se os *bits de validação* estiverem definidos, é feita uma comparação entre as *etiquetas* das duas linhas que pertencem ao conjunto referenciado, e o *campo da etiqueta* do endereço do bloco
- *passo* 4 - quando há um acerto, a cache assinala ao processador para prosseguir com a leitura ou escrita dos dados; no entanto, como Opteron executa fora de ordem, a escrita só ocorre depois de o processador assinalar que a instrução foi cometida.

O cache de dados de Opteron - 50

O Opteron permite 2 ciclos de relógio para estes quatro passos.

O cache de dados de Opteron - 51

Quando há um erro, a cache envia um sinal ao processador informando que os dados ainda não estão disponíveis. Em seguida, o bloco requerido é lido a partir do nível inferior da hierarquia, uma vez que a cache implementa uma política de *write-back* com *alocação write* on write miss. A latência é de 7 ciclos de relógio para os primeiros 8 bytes do bloco e, em seguida, 2 ciclos de relógio por 8 bytes para o resto do bloco.

Devido à associatividade de 2 vias que a cache apresenta, há que começar com duas linhas onde o bloco pode ser armazenado. Se alguma delas não contiver dados válidos, o seu *bit de validação* é reinicializado, a linha é imediatamente seleccionada; caso contrário, é aplicada a regra *menos recentemente utilizada*, ou seja, a linha seleccionada é aquela cujo *bit da LRU* é reinicializado. Note-se que a implementação da regra é neste caso quase trivial: em caso de acerto, o *bit LRU da* linha é definido e o *bit LRU* da linha da contraparte no conjunto é reinicializado.

Contudo, antes de o fazer, devido à política de *write-back*, é verificada a *parte suja* da linha cujo bloco deve ser substituído. Se estiver suja, a sua etiqueta e dados são primeiro removidos para o *buffer da vítima*. A cache fornece espaço para armazenar até oito blocos de *vítimas* que são posteriormente transferidos para o nível inferior, em paralelo com outras actividades de cache. Se, no entanto, em algum momento, o

O cache de dados de Opteron - 52

buffer de vítimas estiver cheio, o processador tem de parar até que seja disponibilizado espaço.

O cache de dados de Opteron - 53

Formato de uma linha de cache

bit de validação	**etiqueta** (25 bits)	**bit sujo**	**LRU bit**	**palavra de dados (**8 bytes)
				Idata palavra (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)
				palavra de dados (8 bytes)

Desempenho da cache - 54

Um método para avaliar o *desempenho da cache* é expandir a equação do *tempo de execução do processador*. A forma de o realizar é extrair da equação o número de ciclos de relógio durante os quais o processador está parado à espera de um acesso à memória, um parâmetro normalmente chamado *ciclos de relógio de paragem de memória*.

Depois obtém-se

$$\text{Tempo de execução da CPU} = (\text{ciclos de relógio da CPU} + \text{ciclos de relógio de perda de memória}) \cdot \text{tempo de ciclo de clock}.$$

Note-se que a equação acima assume que a variável *ciclos de relógio da CPU* inclui o número de ciclos de relógio para lidar com uma falha de cache e que o processador está completamente parado durante uma falha de cache, o caso de um processador de *execução por encomenda.*

Desempenho da cache - 55

O número de *ciclos do relógio de perda de memória* depende tanto do *número de falhas* e sobre o *custo por falta* ou *penalidade por falta*

ciclos de relógios de perda de memória = número de faltas ·miss ciclos de relógios de penalização =

= contagem das instruções · ——misses· ciclos de relógio de penalidades falhadas =

= contagem de instruções · $\frac{\text{acessos à memória}}{\text{instrução}}$ ·

· miss rate· miss ciclos de relógio de penalização .

Obter o valor da *contagem das instruções* é simples se se tiver uma listagem da compilação do programa na língua de montagem. Para processadores especulativos, no entanto, só devem ser contadas as instruções comprometidas. O valor dos *acessos à memória por instrução* pode ser estimado da mesma forma: cada instrução requer

um acesso à memória para a obtenção de instruções e, de acordo com a sua semântica, pode ou não requerer um acesso aos dados.

Desempenho da cache - 57

A *taxa de faltas* pode ser medida com simuladores de cache que tomam um traço de endereço das instruções e referências de dados, simulam o comportamento do cache para afirmar quais as referências que acertam ou falham e reportam no final os seus totais. Muitos microprocessadores fornecem hoje hardware para contar tanto o número de erros como o número de referências de memória em tempo real, o que torna a tarefa mais fácil.

A *pena* variável *de falta* é mais difícil de estimar. A memória por detrás da cache pode estar ocupada no momento da falha devido a pedidos de memória anteriores, de uma actualização da memória, ou mesmo de transferências de E/S que estão a ter lugar. O número de ciclos de relógio também varia nas interfaces entre os diferentes relógios do processador, do autocarro ou da memória. Assim, a utilização de um único valor para a *penalização da falta* é uma simplificação.

Finalmente, uma vez que em muitos casos a *taxa de faltas* e a *penalidade de falta* têm valores diferentes para operações de leitura e escrita, os ciclos de paragem de memória poderiam ser definidos tendo este facto em consideração

$$\text{ciclos do relógio de perda de memória} = \text{contagem de instruções} \cdot \text{operações } (i)$$

$$\text{—} \cdot \sum_{i=r,\,w} \text{instruction} \cdot \text{miss rate}(i) \cdot \text{miss penalty clock cycles}(i) .$$

Desempenho da cache - 58

Desempenho da cache - 59

Assumir um sistema informático onde o valor dos *ciclos do relógio por instrução* (CPI) é 1,0 quando todos os acessos à memória atingem a cache. Os únicos acessos aos dados são instruções de carregamento e armazenamento que representam 50% de todas as instruções executadas na referência a ser executada. Os *ciclos de relógio de penalização de erro* é de 25 e a *taxa de erro* é de 2%.

Quanto mais lento está a correr em comparação com um sistema de computador que tem o mesmo processador e executa o mesmo programa sem falhas de cache, ou, em alternativa, quanto mais rápido está a correr em comparação com um sistema de computador que tem o mesmo processador e executa o mesmo programa com 100% de falhas de cache?

0% taxa de faltas

$$\text{CPU0}_{mr}\,\text{exec time} = (\text{CPU clock cycles} + \text{mem stall clock cycles})\cdot$$
$$\cdot\text{clock tempo de ciclo} =$$
$$= (\text{contagem de instruções} \cdot \text{CPI} + 0)\cdot\text{clock tempo de ciclo} =$$
$$= 1.00\cdot\text{instruction contar}\cdot\text{clock tempo de ciclo}$$

Desempenho da cache - 60

2% taxa de faltas

ciclos de relógios mem stall $_{2\ mr}$ = contagem de instruções · $\frac{\text{acessos à memória}}{\text{instrução}}$ · · miss rate · miss ciclos de relógio de penalização =

= contagem das instruções ·(1.0 + 0.5)·0.02· 25 =

= 0.75· contagem das instruções

CPU2 $_{mr}$ exec time = (CPU clock cycles + mem stall clock cycles)· ·clock tempo de ciclo =

= contagem das instruções ·(1.00+0.75)·clock tempo de ciclo =

= 1.75·instruction contar ·clock tempo de ciclo

Rácio de desempenho

CPU0 $_{mr}$ exec time CPU2 $_{mr}$ exec time

$= \frac{1{,}00 \cdot \text{instruction}}{\text{contar} \cdot \text{clock tempo}}$

de ciclo

1,75 ·instruction contar ·clock tempo de ciclo = 0,5

Desempenho da cache - 61

Desempenho da cache - 62

Taxa de faltas a 100%

ciclos de relógios mem stall $_{100\ mr}$ = contagem de instruções · $\frac{\text{acessos à memória}}{\text{instrução}}$ ·

· miss rate · miss ciclos de relógio de penalização =

= contagem das instruções ·(1.0 + 0.5)·1.0 ·25 =

= 37.5·instruction contar

CPU100 $_{mr}$ exec time = (CPU clock cycles + mem stall clock cycles)·

·clock tempo de ciclo =

= contagem das instruções ·(1.00+37.5)·clock tempo de ciclo =

= 38.5 · contagem das instruções ·clock tempo de ciclo

Rácio de desempenho

$\frac{\text{CPU100 }_{mr}\text{ tempo de execução}}{\text{CPU2 }_{mr}\text{ tempo de execução}} = \frac{38.5 \cdot \text{contagem das instruções} \cdot \text{clock tempo de ciclo}}{}$

1.75·instruction contar ·clock tempo de ciclo

Desempenho da cache - 63

Desempenho da cache - 64

Como este exemplo ilustra, o comportamento da cache pode ter um enorme impacto no desempenho. As falhas de cache têm um impacto duplo sobre um processador com baixo CPI e um relógio rápido.

- quanto mais baixo for o IPC, maior é o impacto relativo da penalidade de falta independente do processador medida como um número fixo de ciclos de relógio
- mesmo que as hierarquias de memória para dois sistemas informáticos sejam idênticas, o processador com a maior taxa de relógio tem um maior número de ciclos de paragem de memória.

A importância da cache para processadores com baixo IPC e altas taxas de relógio é assim maior e, portanto, maior é também o perigo de negligenciar o comportamento da cache ao avaliar o desempenho destes sistemas informáticos.

Desempenho da cache - 65

Alguns designers preferem definir a *taxa de falhas* como o número de falhas por instrução, em vez do número de falhas por acesso à memória. Estes dois parâmetros estão relacionados por

$$\frac{\text{número total de faltas}}{\text{contagem de instruções}} = \text{taxa de miss rate} \cdot \frac{\text{acessos à memória}}{\text{contagem de instruções}}.$$

A vantagem de utilizar *misses por instrução* para especificar a *taxa de misses* é que converte este valor de mérito num valor que é independente da implementação. Os processadores especulativos, por exemplo, obtêm cerca de duas vezes mais instruções do que as que são realmente cometidas. Assim, reduzindo artificialmente a taxa de faltas, se medida como faltas por referência de memória.

O inconveniente é que *as falhas por instrução* são dependentes da arquitectura, o que significa que não faz qualquer sentido utilizá-la para comparar duas arquitecturas bastante diferentes, tais como MIPS e Intel 80x86.

Desempenho da cache - 66

Em vez de nos concentrarmos na *taxa de falhas* para avaliar o desempenho da hierarquia da memória, poderíamos usar como melhor figura de mérito a *memória média tempo de acesso*

tempo médio de acesso à memória = tempo de acerto + + falhar rate· ulhar ciclos de relógio de penalização ·clock tempo de ciclo .

O *tempo de acerto da* variável é o tempo de acesso à cache de um acerto de cache e as outras variáveis, *taxa de erro* e *penalidade de erro*, têm o mesmo significado que antes.

Desempenho da cache - 67

Deve ser tomada uma decisão sobre a implementação de uma cache de instruções de 16 KB e de uma cache de dados de 16 KB versus uma cache unificada de 32 KB para um determinado sistema informático.

Os resultados das simulações em pontos de referência devidamente escolhidos indicaram os seguintes valores para o número de faltas por 1000 instruções: 3,82 (16 KB cache de instruções),
40,9 (16 KB cache de dados) e 43,3 (32 KB cache unificado), onde 36% da totalidade das instruções que foram executadas são transferências de dados. Assumir que um acerto leva 1 ciclo de relógio e que o erro é de 200 ciclos de relógio de penalidade. Assumir também que um hit de carga ou de armazenamento leva 1 ciclo de relógio extra na cache unificada, uma vez que há uma única porta para atender a dois pedidos simultâneos e que a cache de dados e a cache unificada implementam uma política de escrita com um buffer de escrita (os bloqueios na escrita podem ser ignorados).

$$\text{taxa de falha} = \frac{\frac{\text{falhas por 1000 instruções}}{1000}}{\text{contagem de}}$$

Desempenho da cache à memória - 68

instruções de acesso

Desempenho da cache - 69

A *cache de instruções* tem exactamente um acesso à memória por instrução

$$\text{miss rate16}_{\text{KB instrução=}} \quad \frac{3.82 \cdot 10^{-3}}{1.00=} \qquad 0.004.$$

O *cache de dados* tem em média 0,36 acessos de memória por instrução

$$\text{miss rate16}_{\text{KB data=0}} \frac{40.90 \cdot 10^{-3}}{.36=} \quad 0.114 .$$

A *cache unificada* tem em média 1,36 acessos de memória por instrução

$$\text{miss rate32}_{\text{KB unified=}} \quad \frac{43.30 \cdot 10^{-3}}{1.36=} \qquad 0.032.$$

Desempenho da cache - 70

A *taxa* efectiva *de erros* da *instrução* combinada + *caches de dados* é dada por

$$\text{taxa de falha}_{\text{16 KB instrução + dados}} = \frac{\text{acessos mem}_{\text{instruction}}}{\text{mem}_{\text{total}}} \cdot \text{taxa de falha}_{\text{16 KB em construção}} + \frac{\text{acessos}_{\text{data}}\ \text{mem}}{\text{mem}_{\text{total}}} \cdot \text{taxa de falha}_{\text{16 KB dados}} =$$

$$= \frac{1.00}{1.36} \cdot 0.004 + \frac{0.36}{1.36} \cdot 0.114 = 0.033 \quad .$$

Note que a cache unificada de 32 KB tem uma taxa de falhas ligeiramente inferior à combinação de duas caches separadas de instruções e dados de 16 KB!

Desempenho da cache - 71

A *média dos ciclos do relógio de acesso à memória* é dada por

$$\text{média dos ciclos do relógio de acesso à memória} = \frac{\text{mem accessintruction}}{\text{acessos}} \cdot (\text{hit time} + \text{miss rate-instruction} \cdot \text{miss ciclos de relógio de penalização}) + \frac{\text{mem accessdata}}{\text{acessos mem}_{\text{total}}} \cdot (\text{hit time} + \text{miss}_{\text{ratedata}} \cdot \text{miss ciclos de relógio de penalização})$$

ceder em cada caso

$$\text{média de ciclos do relógio de acesso à memória}_{32\ \text{KB unificado}} = \frac{1.00}{1.36} \cdot (1.0 + 0.032 \cdot 200) + \frac{0.36}{1.36} \cdot (2 + 0.032 \cdot 200) = 7.66$$

Desempenho da cache - 72

média dos ciclos do relógio de acesso à memória16 $_{\text{Instrução KB + dados}}$ =

$$= \frac{1.00}{1.36} \cdot (1.0 + 0.004 \cdot 200) + \frac{0.36}{1.36} \cdot (1 + 0.114 \cdot 200) = 7.62 \quad .$$

Embora as duas caches separadas de instruções e dados de 16 KB tenham uma taxa efectiva de falhas ligeiramente superior à da cache unificada de 32 KB, o facto é que quando se considera o tempo médio de acesso à memória, o resultado é invertido devido à existência, neste caso, de duas portas de memória por ciclo de relógio, evitando assim o perigo estrutural presente na cache unificada de uma única porta.

Desempenho da cache - 73

Qual é o impacto de duas organizações de cache diferentes sobre o desempenho de um processador específico?

Assumir que o CPI com uma cache perfeita (sem falhas de cache) é 1,6, que o ciclo do relógio é 0,35 ns e que existem 1,4 referências de memória por instrução para a referência a ser executada. O tamanho de ambas as caches é 128 KB e ambas têm um tamanho de bloco de 64 bytes. A primeira está organizada como directamente mapeada e a segunda como associativa de 2 vias. Como a cache de dados Opteron ilustra, deve ser adicionado um multiplexor para seleccionar entre os dados do conjunto, dependendo da correspondência da etiqueta. Uma vez que a velocidade do processador pode ser ligada directamente à velocidade de um acerto de cache, suponha que o tempo de ciclo do processador é esticado 1,35 vezes para acomodar o multiplexador de selecção do conjunto associativo de cache. Em ambos os casos, o tempo de acerto é de 1 ciclo de relógio, a penalidade de falta é de 65 ns e a taxa de falta é de 2,1% e 1,9%, respectivamente, para o mapa directo e as caches associativas de 2 vias do conjunto.

Desempenho da cache - 74

tempo médio de acesso à memória128 KB directamente mapeada=

= relógio de acerto cycles·clock tempo de ciclo + miss rate· miss penalidade =

= 1,0 ·0,35 + 0,021 ·65 = 1,72 ns

tempo médio de acesso à memória128 KB 2-way set associative=

= relógio atingido cycles·effective tempo de ciclo do relógio + taxa de erro ·miss penalidade =

= 1,0 ·1,35 ·0,35 + 0,019 ·65 = 1,71 ns

Note que a cache associativa de 128 KB 2 vias tem um tempo médio de acesso à memória ligeiramente inferior ao da cache directamente mapeada de 128 KB!

Desempenho da cache - 75

Tempo de execução da CPU128 $_{\text{KB directamente mapeada}}$ =

$$= \text{contagem de instruções} \cdot \text{CPI} \cdot \text{clock tempo de ciclo} +$$

$$+ \text{contagem de instruções} \cdot \frac{\text{acessos à memória}}{\text{instrução}} \cdot \text{miss rate} \cdot \text{miss penalização} =$$

$$= (1{,}6 \cdot 0{,}35 + 1{,}4 \cdot 0{,}021 \cdot 65) \cdot \text{instruction contagem} = 2{,}47 \cdot \text{instruction contagem}$$

Tempo de execução da CPU128 $_{\text{KB 2-way set associative}}$ =

$$= \text{contagem de instruções} \cdot \text{CPI} \cdot \text{effective tempo de ciclo de relógio} +$$

$$+ \text{instrução count} \cdot \frac{\text{acesso à memória}}{\text{instrução}} \cdot \text{miss rate} \cdot \text{miss penalidade} =$$

$$= (1.6 \cdot 1.35 \cdot 0.35 + 1.4 \cdot 0.019 \cdot 65) \cdot \text{instruction contagem} = 2.49 \cdot \text{instruction contagem}$$

Embora a cache associativa de 128 KB 2 vias tenha um tempo médio de acesso à memória inferior ao da cache directamente mapeada de 128 KB, o facto é que quando se considera o tempo de execução da CPU, o resultado é invertido devido ao

alongamento do ciclo do relógio no caso da cache associativa.

Desempenho da cache - 77

Para um processador de *execução fora de ordem*, a *penalidade de falta* já não pode ser definida como a latência total da falta à memória. Esta questão é relevante uma vez que os processadores *fora de ordem* são conhecidos por tolerar alguma latência devido a falhas na memória sem afectar o seu desempenho.

O número de *ciclos de paragem de memória* pode ser redefinido para levar a uma nova definição de
pena de falta como uma latência não sobreposta

$$\text{ciclos de paragem de memória} = \text{contagem de instruções} \cdot \frac{\text{acessos à memória}}{\text{instrução}} \cdot$$
$$\cdot \text{ taxa de faltas } \cdot (\text{total de faltas} - \text{ latência de faltas sobrepostas}) .$$

Diz-se que um *processador é bloqueado num ciclo de relógio* se não retirar o máximo número possível de instruções nesse ciclo. A *barraca* é atribuída à primeira instrução que não pôde ser reformada. *A latência, por* outro lado, pode ser medida desde o momento em que a instrução de memória é enfileirada na janela de instruções, ou desde o momento em que o endereço é gerado, até ao momento em que a transferência de dados tem lugar. Qualquer alternativa é válida desde que seja

Desempenho da cache - 78

utilizada de forma consistente.

Desempenho da cache - 79

Considere que o processador do último exemplo tem um tempo de ciclo de relógio 1,35 vezes mais longo para suportar a execução fora de ordem e tem uma cache directamente mapeada. Assumir também que 30% dos 65 ns de falta de penalização podem ser sobrepostos, reduzindo assim o ciclo médio de perda de memória para 45,5 ns.

acesso médio à memória com tempo de acesso directo $_{\text{mapeado, OOO}}$ =

= relógio de acerto cycles·clock tempo de ciclo + miss rate·effective miss penalidade =

= 1,0 ·1,35 ·0,35 + 0,021 ·45,5 = 1,43 ns

CPU execução timedirect $_{\text{mapeada, OOO}}$ =

= contagem de instruções ·CPI ·clock tempo de ciclo =

+ contagem de instruções · $\frac{\text{acesso à memória}}{\text{instrução}}$ · miss rate·effective miss penalidade = miss

= (1.6 ·1.35 ·0.35 + 1.4 ·0.021·45.5)·instruc contagem = 2.09· contagem de instruções

Desempenho da cache - 80

Contudo, devido à sua complexidade, os designers tendem a utilizar simuladores do processador fora de ordem e da memória quando avaliam os trade-offs na hierarquia da memória para avaliar que uma melhoria que ajuda a latência média da memória, também ajuda o desempenho do programa.

Optimização da cache - 81

A abordagem tradicional para melhorar o comportamento de uma cache é a de minimizar a taxa de falhas. As faltas podem ser modeladas em três categorias básicas

- *erros obrigatórios* - erros que ocorrem mesmo que a cache tenha um tamanho infinito; o primeiro acesso a qualquer byte ou palavra traduzir-se-á sempre num erro porque o bloco de memória onde o byte ou palavra reside deve ser trazido primeiro para a cache; são também conhecidos como *erros de arranque a frio* ou *erros de primeira referência*
- *falhas de capacidade* - falhas que ocorrem numa cache totalmente associativa; estão directamente relacionadas com o tamanho da cache, se a cache não for suficientemente grande, os blocos de memória serão descartados durante a execução do programa e mais tarde serão recuperados porque são novamente necessários
- *falhas de conflito* - falhas que ocorrem especificamente devido à organização interna da cache; pondo de lado o caso de uma cache totalmente associativa e considerando a cache directamente mapeada como uma instância de uma cache associativa de um conjunto, os blocos de memória podem ser descartados e mais tarde recuperados simplesmente porque demasiados

blocos são mapeados em alguns dos conjuntos; são também conhecidos como *falhas de colisão*.

Taxa de falta para a arquitectura Alfa (DEC) usando 10 referências SPEC2000 (5 SPECint2000 e 5 SPECfp2000) - substituição da LRU - tamanho do bloco = 64 bytes

Fonte: adaptado de Computer Architecture: Uma Abordagem Quantitativa

			Componentes da taxa em falta (valor / % relativa do total)					
Tamanho da cache (KB)	***Grau de associatividade***	***Taxa de faltas totais***	***Obrigatório***		***Capacidade***		***Conflito***	
4	1 via	0.098	0.0001	0.1%	0.070	72%	0.027	28%
4	2 vias	0.076	0.0001	0.1%	0.070	93%	0.005	7%
4	4 vias	0.071	0.0001	0.1%	0.070	99%	0.001	1%
4	8 vias	0.071	0.0001	0.1%	0.070	100%	0.000	0%
16	1 via	0.049	0.0001	0.1%	0.040	82%	0.009	17%
16	2 vias	0.041	0.0001	0.2%	0.040	98%	0.001	2%
16	4 vias	0.041	0.0001	0.2%	0.040	99%	0.000	0%
16	8 vias	0.041	0.0001	0.2%	0.040	100%	0.000	0%
64	1 via	0.037	0.0001	0.2%	0.028	77%	0.008	23%
64	2 vias	0.031	0.0001	0.2%	0.028	91%	0.003	9%
64	4 vias	0.030	0.0001	0.2%	0.028	95%	0.001	4%
64	8 vias	0.029	0.0001	0.2%	0.028	97%	0.001	2%
256	1 via	0.013	0.0001	0.5%	0.012	94%	0.001	6%
256	2 vias	0.012	0.0001	0.5%	0.012	99%	0.000	0%
256	4 vias	0.012	0.0001	0.5%	0.012	99%	0.000	0%
256	8 vias	0.012	0.0001	0.5%	0.012	99%	0.000	0%

Optimização da cache - 84

Como seria de esperar que *as falhas obrigatórias* fossem independentes da dimensão da cache, as *falhas de capacidade* diminuem à medida que a dimensão da cache aumenta e as *falhas de conflito* diminuem à medida que o grau de associatividade aumenta.

É importante aumentar o tamanho da cache. Se a memória do nível superior for demasiado pequena para conter a fracção do código do programa e os dados necessários pelo princípio da localidade, então uma parte significativa do tempo é gasto transferindo blocos de memória entre dois níveis adjacentes da hierarquia e ocorrerá *thrashing*. Assim, o sistema informático funcionará mais próximo da velocidade da memória de nível inferior, ou ainda mais lento devido à falta de sobrecarga. Por outro lado, a implementação da associatividade total para se livrar das falhas de conflito é expansiva em termos de hardware e pode levar a um ritmo de relógio lento, baixando assim o desempenho global.

O *modelo miss que* acaba de ser apresentado tem os seus próprios limites: dá uma visão do comportamento médio, mas não explica os erros individuais, nem incorpora a política de substituição. Muitas técnicas que reduzem as taxas de faltas, aumentam também o *tempo de acerto* e/ou a *pena de falta*. Por conseguinte, é necessária uma

abordagem equilibrada para tornar todo o sistema informático mais rápido.

Optimização da cache - 86

Uma forma directa de reduzir a taxa de falhas é ***aumentar o tamanho do bloco***. Os tamanhos de blocos maiores diminuirão os *erros obrigatórios devido* à localidade espacial. Contudo, se o tamanho do bloco for demasiado grande em relação ao tamanho da cache, a redução do número de linhas de cache tende a aumentar a *capacidade* e as *falhas de conflito* e a piorar a taxa total de falhas.

Taxa em falta vs. tamanho do bloco para o DECStation 5000 utilizando os padrões de referência SPEC92

Fonte: Gee, Hill, Pnevimatikatos, Smith, "Cache performance of the SPEC92 benchmark suite", IEEE Micro 13:4, Agosto 1993

	Tamanho da cache			
Tamanho do bloco (bytes)	**4 *KB***	**16 *KB***	**64 *KB***	**256 *KB***
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	**7.00%**	**2.64%**	1.06%	0.51%
128	7.78%	2.77%	**1.02%**	**0.49%**

Optimização da cache - 87

[illegible]	9.51%	3.39%	1.15%	**0.49%**

Optimização da cache - *88*

Ao mesmo tempo, o tamanho maior do bloco tende a aumentar a *penalidade de falta*, devido ao número de bytes a serem transferidos também aumenta

miss penalty = ciclos de relógio de latência de cache ·clock tempo de ciclo +
+ tamanho do bloco / largura de banda (ciclos de relógio) ·clock tempo de ciclo .

Média dos ciclos do relógio de acesso à memória vs. tamanho do bloco para DECStation 5000 utilizando os valores de referência SPEC92

Fonte:Computer Architecture: Uma Abordagem Quantitativa

ciclos de acerto do relógio = 1
ciclos de relógio de latência de cache = 80 largura de banda = 8 bytes / ciclo de relógio

		Tamanho da cache			
Tamanho do bloco	***Pena de falta (ciclos de***	**4 *KB***	**16 *KB***	**64 *KB***	**256 *KB***

Optimização da cache - 89

(bytes)	(cego)				
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

Optimização da cache - 90

Uma vez que a escolha do tamanho do bloco será afectada pela latência e pela largura de banda da memória de nível inferior, o desenhador da cache tem de considerar tanto a taxa de falhas como a penalização de falhas ao decidir sobre o tamanho do bloco de cache.

A alta latência e a alta largura de banda encorajam um bloco de grandes dimensões porque a cache recebe muito mais bytes por miss para um pequeno aumento da penalidade de miss. Por outro lado, a baixa latência e a baixa largura de banda encorajam tamanhos de blocos mais pequenos porque o tempo poupado de um bloco maior não é significativo. É preciso lembrar que ter um grande número de blocos pequenos pode reduzir as falhas de conflito.

As falhas de capacidade são obviamente reduzidas através do aumento do tamanho da cache. No entanto, existe um limite porque o tempo de acerto é potencialmente mais longo devido ao aumento da complexidade lógica, tornando-o também uma solução de maior custo e maior potência. Não obstante, a sua técnica tem sido especialmente popular nas caches localizadas fora do chip processador.

Optimização da cache - 91

Da mesma forma, embora uma maior associatividade reduza a taxa de faltas, o tempo médio de acesso à memória nem sempre é também diminuído.

Ciclos médios do relógio de acesso à memória vs. associatividade para o DECStation 5000 usando padrões de referência SPEC92 - substituição da LRU - tamanho do bloco = 64 bytes

Fonte:Computer Architecture: Uma Abordagem Quantitativa

ciclos de acerto do relógio = 1

tempo de ciclo do relógio $_{2-way}$ = 1,36 ·clock tempo de ciclo $_{1-way}$

tempo de ciclo de relógio4-vias= 1,44 ·clock tempo de ciclo1-vias tempo de ciclo8-vias= 1,52 ·clock tempo de ciclo1-vias

	Associatividade			
***Tamanho da cache* (KB)**	**1 via**	**2 vias**	**4 vias**	**8 vias**
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
128	1.52	1.84	1.92	2.00

Optimização da cache - 92

[illegible]	1.32	1.66	1.74	1.8[illegible]

Memória principal

A memória principal satisfaz principalmente as exigências das caches e serve de interface I/O tanto para a área de troca numa organização de memória virtual como para os diferentes controladores de dispositivos, actuando como destino para os dados de entrada e fonte para os dados de saída.

Tradicionalmente, a *latência de memória*, que afecta a penalidade de falha de memória, é a principal preocupação da cache, enquanto a *largura de banda de memória* é a principal preocupação dos controladores de E/S, mas também é importante para as caches de vários níveis com os seus blocos de maiores dimensões. Uma maior largura de banda pode ser conseguida utilizando múltiplos bancos de memória, alargando o bus de dados e introduzindo o *modo de transferência de explosão*, onde múltiplas palavras são transferidas no mesmo acesso.

A memória principal é construída em torno de chips DRAM. Actualmente, *a latência da memória* é expressa através de duas figuras de mérito

- *tempo de acesso* - intervalo de tempo entre a emissão de um pedido de leitura / escrita e o momento em que os dados associados ficam disponíveis / são armazenados
- *tempo de ciclo* - intervalo de tempo mínimo entre pedidos de memória não

relacionados.

DRAM

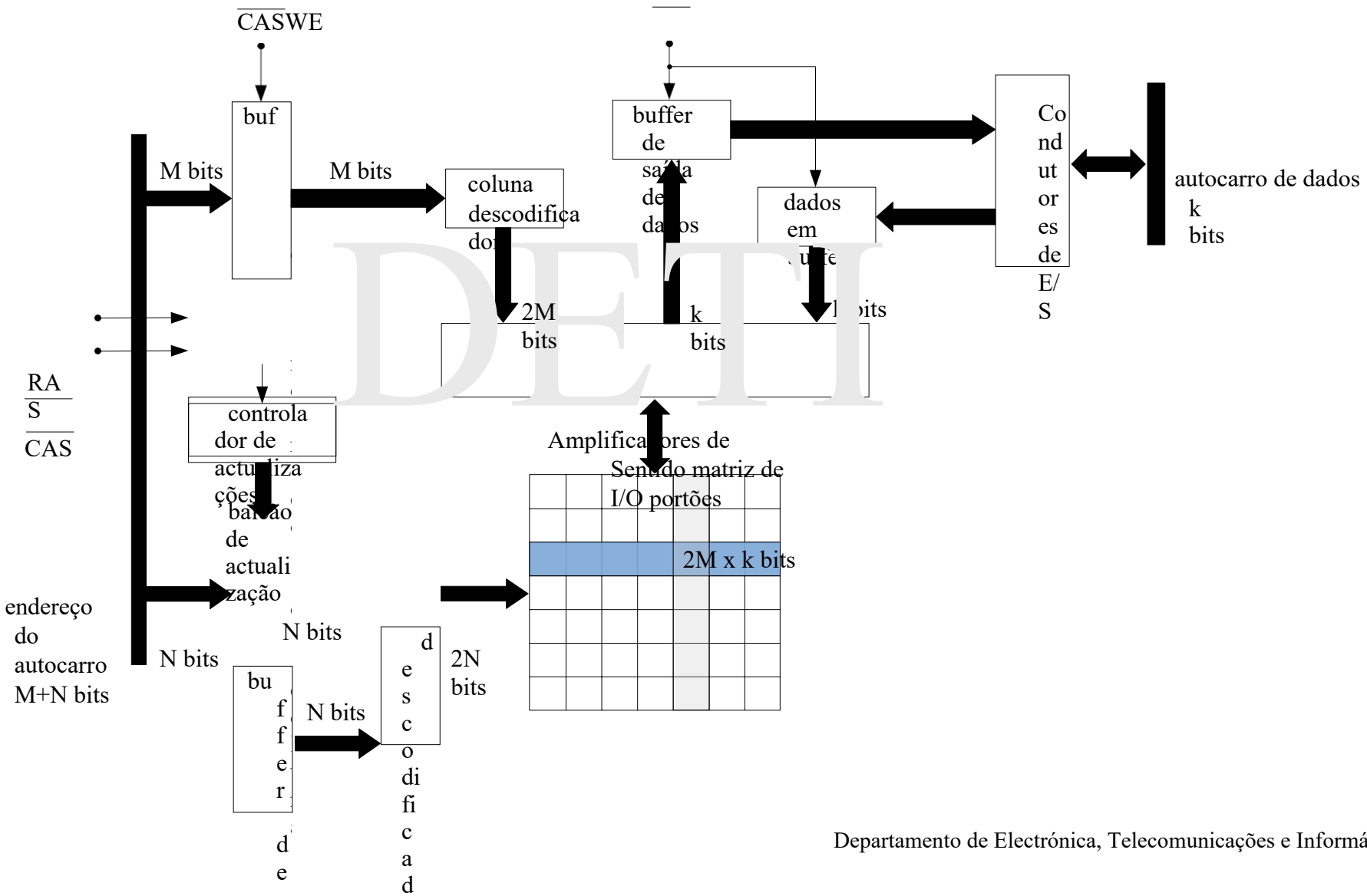
À medida que as RAM dinâmicas cresciam em capacidade, o custo do pacote com todos os pinos de endereço necessários tornou-se um problema. Para o resolver, as linhas de endereço foram multiplexadas: parte do endereço é bloqueado internamente durante a primeira fase de acesso à memória, com o sinal de controlo de *acesso à linha strobe*, e o endereço restante é bloqueado mais tarde, durante a segunda fase, com o sinal de controlo de *acesso à coluna strobe.*

A fim de embalar mais pedaços por chip, um único transistor é utilizado para armazenar um pedaço. A leitura deste bit destrói a informação armazenada, o que significa que tem de ser restaurada (escrita de volta) após cada leitura. Esta é uma das razões pelas quais o tempo de ciclo de memória é mais longo do que o tempo de acesso à memória. Com a introdução de múltiplas DRAM bancárias, que permitem ocultar a parte reescrita do ciclo, este problema foi atenuado.

Além disso, para evitar a perda de informação quando um bit armazenado não é lido durante um longo período de tempo, tem de se fazer um refresco de carga. Todos os bits pertencentes à mesma linha podem ser refrescados simultaneamente, bastando para isso aceder a uma coluna dessa linha. Assim, têm de ser disponibilizados meios para que cada DRAM aceda a cada uma das suas filas dentro de algum período de tempo, normalmente algumas dezenas de milissegundos. Os

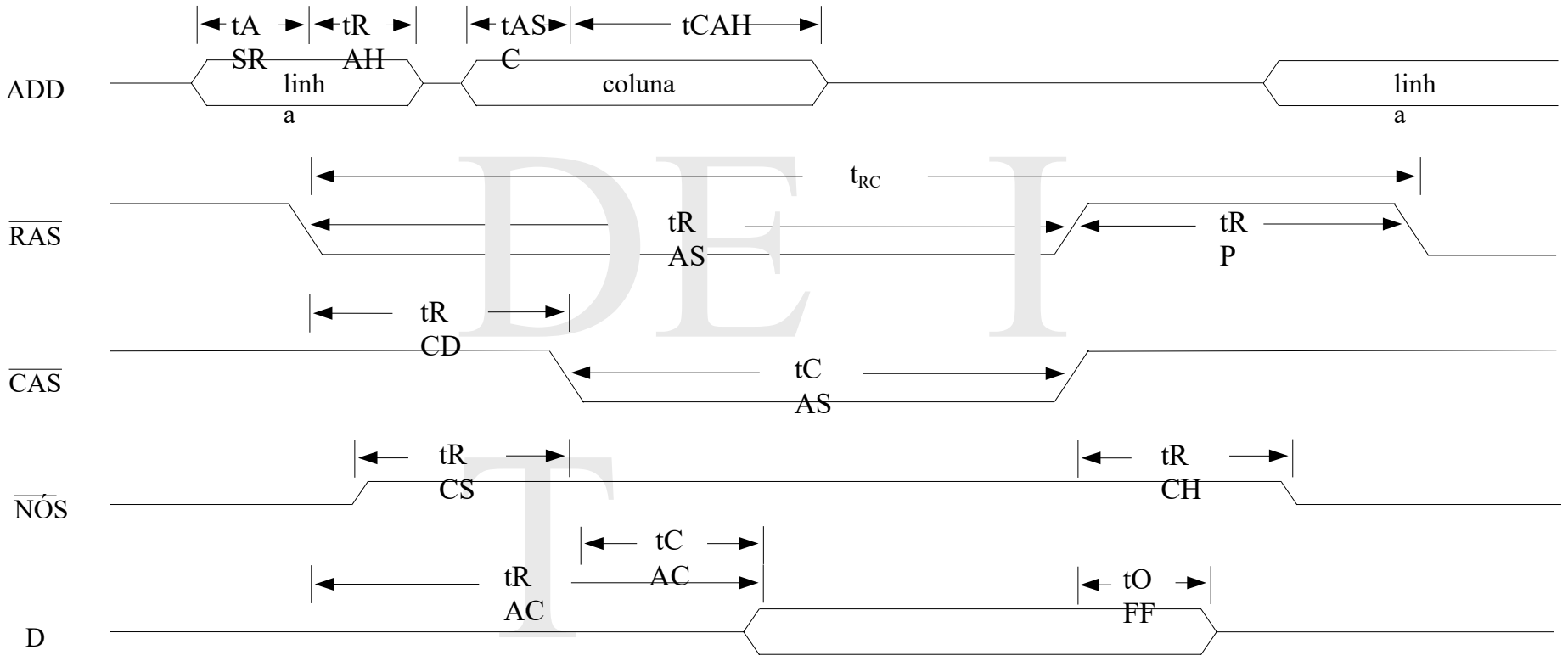
controladores de memória têm hardware para levar a cabo esta tarefa.

DRAM assíncrona - 97



DRAM assíncrona - 98

Operação de leitura



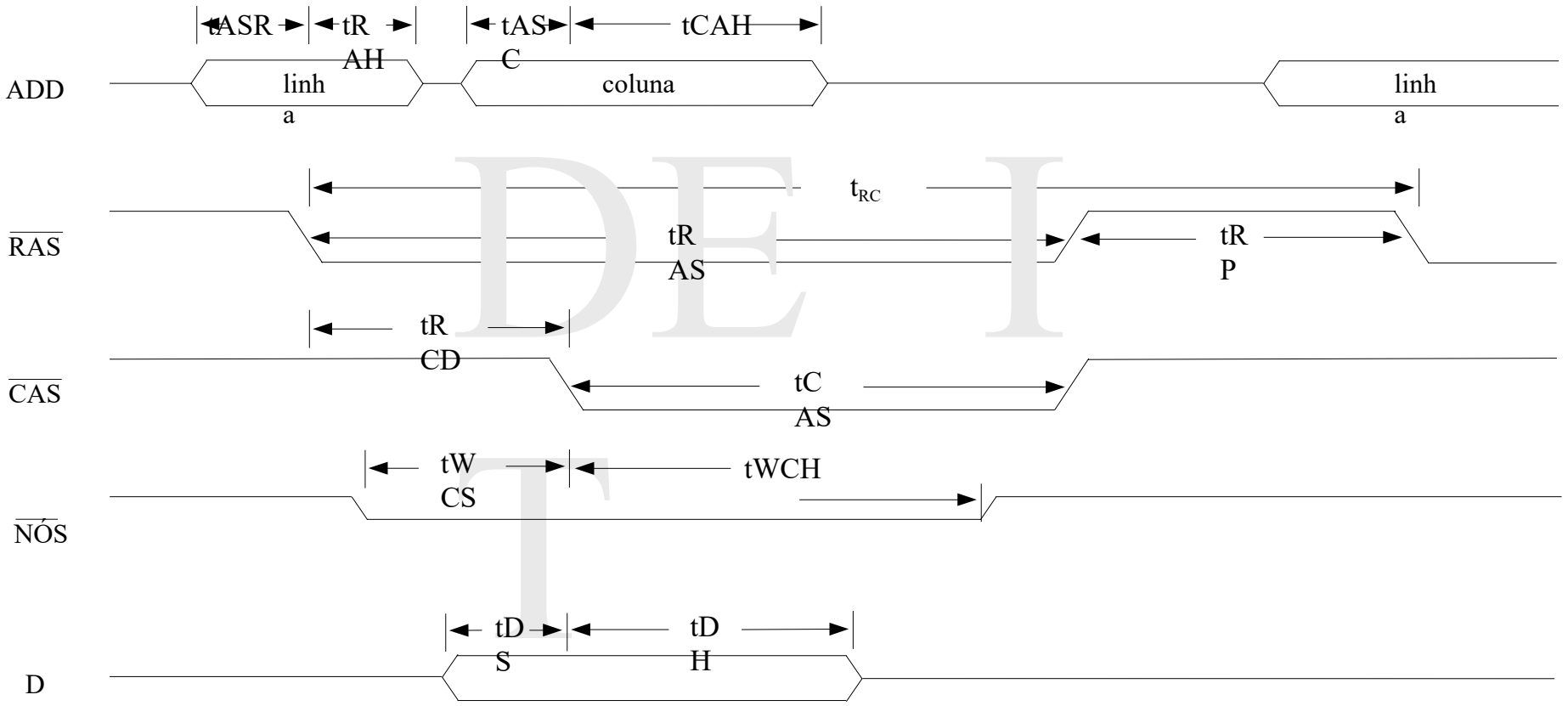
DRAM assíncrona - 99

Operação de leitura

- o sinal de *permissão de escrita* é afirmado alto pelo *controlador de memória* para significar que uma operação de leitura está a ter lugar
- um endereço N-bit é colocado no barramento de endereços e é fixado no *buffer de endereços da linha* na borda de descida do *strobe de acesso à linha* afirmado pelo *controlador de memória*
- os valores dos dados armazenados na linha seleccionada de células de memória são então detectados e mantidos na *matriz de amplificadores de sentido* para mais tarde serem escritos de volta para as células de memória
- um endereço M-bit é colocado a seguir no barramento de endereços e é fixado no *buffer de endereços da coluna* na borda descendente do *strobe de acesso à coluna* afirmado pelo *controlador de memória*
- os valores de dados mantidos na coluna seleccionada da *matriz de amplificadores de sentido* são fixados no *buffer de saída de dados* para conduzir o bus de dados

DRAM assíncrona - 100

Operação de escrita



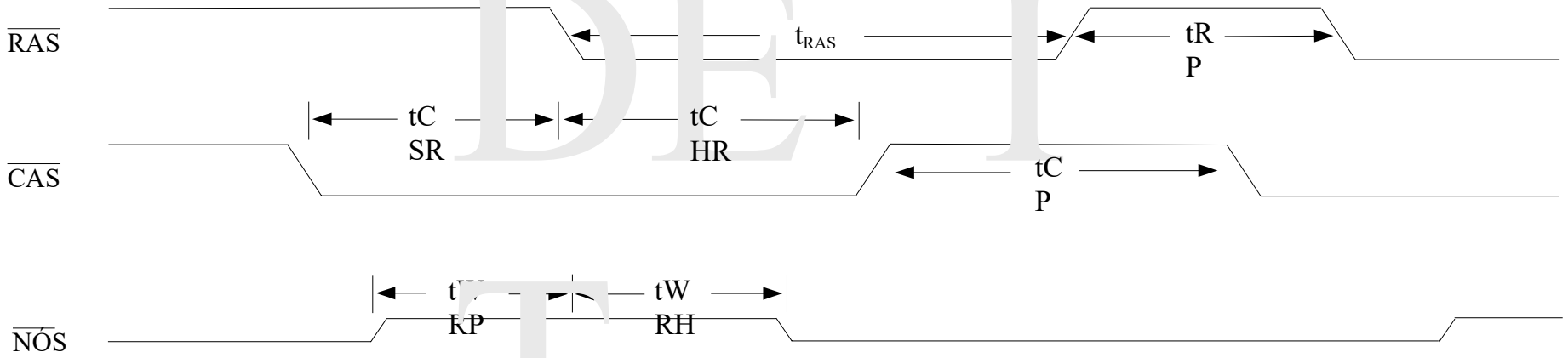
DRAM assíncrona - 101

Operação de escrita

- o sinal de *permissão de escrita* é afirmado baixo pelo *controlador de memória* para significar que uma operação de escrita está a ter lugar
- um endereço N-bit é colocado no barramento de endereços e é fixado no *buffer de endereços da linha* na borda descendente do *strobe de acesso à linha* afirmado pelo *controlador de memória*
- os valores dos dados armazenados na linha seleccionada de células de memória são então detectados e mantidos na *matriz de amplificadores de sentido* para mais tarde serem escritos de volta para as células de memória
- um endereço M-bit é colocado a seguir no barramento de endereços e é fixado no *buffer de endereços da coluna* na borda descendente do *strobe de acesso à coluna* afirmado pelo *controlador de memória*
- os valores dos dados presentes no barramento de dados são fixados nos *dados em buffer* na borda descendente do *strobe de acesso à coluna* e substituem os valores armazenados na coluna seleccionada da *matriz de amplificadores de sentido* antes da escrita de volta para as células de memória

DRAM assíncrona - 102

Operação de actualização (CAS antes do RAS)



DRAM assíncrona - 103

Operação de actualização (CAS antes do RAS)

- o sinal *strobe de acesso à coluna* é afirmado baixo pelo *controlador de memória* antes do *strobe de acesso à linha* e o sinal de *permissão de escrita* é afirmado alto, a lógica de controlo dentro do chip interpreta esta disposição como uma operação de actualização
- as entradas das linhas de endereço são ignoradas e o conteúdo do *contador de actualizações* é fixado no *buffer de endereços da linha* na borda de descida do *strobe de acesso à linha*
- os valores dos dados armazenados na linha seleccionada de células de memória são então detectados e mantidos na *matriz de amplificadores de sentido* para mais tarde serem escritos de volta para as células de memória

DRAM assíncrona - 104

A interface DRAM assíncrona foi modificada para melhorar o desempenho das operações de leitura e escrita na mesma linha de memória, evitando a exigência de pré-carga e abertura repetida da linha para acesso a uma coluna diferente.

No *modo de página* DRAM, depois de uma fila ser aberta mantendo a linha de *acesso strobe* baixa, a fila é mantida aberta e múltiplas operações de leitura ou escrita são realizadas para qualquer uma das colunas da fila. Cada acesso de coluna é iniciado afirmando o *strobe de acesso de coluna* baixo e apresentando um endereço de coluna. Para operações de leitura, o sinal *de permissão de escrita* também deve ser definido como alto. Para operações de escrita, o sinal de permissão de *escrita* deve ser definido como baixo e os valores dos dados a escrever devem ser apresentados juntamente com o endereço da coluna.

O *modo de página* DRAM foi posteriormente melhorado com uma pequena modificação que reduziu ainda mais a latência, dando origem à chamada DRAM *em modo de página rápida*, ou DRAM FPM. Num modo de *página* DRAM, o *strobe de acesso à coluna* é afirmado após o endereço da coluna ser fornecido. Numa DRAM FPM, o endereço da coluna pode ser fornecido enquanto o *estroboscópio de acesso à coluna* continua a ser desasserir. O endereço da coluna propaga-se através do

DRAM assíncrona - 105

caminho de dados de endereço da coluna, mas não produz dados nos pinos de dados até que o *strobe de acesso à coluna* seja afirmado.

DRAM síncrona - 106

A DRAM síncrona, ou SDRAM, muda de forma radical a forma como a interface de memória externa interage com o dispositivo. Os sinais de controlo já não têm um efeito directo nas funções internas, mas um sinal de relógio controlado externamente é utilizado para gerir uma máquina de estado finito incorporada que actua sobre os comandos de entrada. Assim, os sinais RAS e CAS já não actuam como estroboscópios, mas passam a fazer parte do comando de entrada.

As operações internas são canalizadas para melhorar o desempenho. As operações iniciadas anteriormente estão a ser concluídas, enquanto novos comandos são recebidos e começam a ser processados.

A matriz de células de memória bidimensional está dividida em várias secções de igual tamanho, mas independentes, chamadas *bancos*, permitindo ao dispositivo operar num comando de acesso à memória, ler ou escrever, em cada banco simultaneamente e acelerar o acesso de uma forma intercalada.

O modo de transferência de dados é também implementado para aceder a múltiplas palavras de dados com o mesmo comando.

DRAM síncrona - 107

As gerações posteriores de SDRAMs melhoraram a largura de banda de transferência, permitindo que a transferência de dados ocorresse tanto na borda ascendente como na descendente do sinal do relógio controlado externamente. Esta optimização é conhecida como *taxa de dados dupla*, ou DDR.

O DDR deu origem a uma sequência de normas

- DDR, ou DDR1 - tem uma tensão de alimentação de 2,5 V e funciona com uma frequência de 133, 150 e 200 MHz
- DDR2 - tem uma tensão de alimentação de 1,8 V e funciona a frequências de relógio de 266, 333 e 400 MHz
- DDR3 - tem uma tensão de alimentação de 1,5 V e funciona a frequências de relógio de 533, 666 e 800 MHz
- DDR4 - tem uma tensão de alimentação de 1,2 V e funciona a frequências de 1066, 1333 e 1600 MHz.

As DRAM são normalmente vendidas em placas pequenas, chamadas *módulos de memória dupla em linha*, ou DIMMs, que contêm 4 a 16 chips SDRAM e são normalmente organizadas para fornecer um comprimento de palavra de 64 bits de

DRAM síncrona - 108

dados + bits de *código de correcção de erros.*

DRAM síncrona - 109

SDRAM de 64 Mbit

(organizado em 4 bancos de 2048 filas x 1024 colunas de células de memória de 8 bits cada)

$\overline{CS}$	$\overline{RAS}$	$\overline{CAS}$	$\overline{NÓS}$	BAn	A10	Um	Comando
H	X	X	X	X	X	X	inibir o comando (o dispositivo não é seleccionado)
L	H	H	H	X	X	X	nenhuma operação
L	H	H	L	X	X	X	fim da explosão: parar uma leitura de explosão, ou uma escrita de explosão, em curso
L	H	L	H	banco	L	coluna	ler um rebentamento de dados da linha activa actual
L	H	L	H	banco	H	coluna	ler uma sequência de dados da linha activa actual com pré-carga (fechar a linha) quando terminado
L	H	L	L	banco	L	coluna	escrever uma explosão de dados a partir da linha activa actual
L	H	L	L	banco	H	coluna	escrever uma descarga de dados da linha activa actual com pré-carga (fechar a linha) quando estiver feito
L	L	H	H	banco	linha		activar (abrir) a linha para um comando de leitura ou escrita
L	L	H	L	banco	L	X	pré-carregar, ou desactivar (fechar), a linha actual do banco seleccionado
L	L	H	L	X	H	X	pré-carregar, ou desactivar (fechar), a fila actual de todos os bancos
L	L	L	H	X	X	X	actualizar uma fila de cada banco (actualização automática) utilizando um contador interno (todas as filas devem ser pré-carregadas)
L	L	L	L				registo em modo de carga para configuração do chip: a palavra

Gerações DRAM

Especificações de tempo

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa

Ano de produção	Tamanho do chip (bit)	DRAM tipo	DRAMs mais lentas (ns)	DRAMs (ns) mais rápidas	CAS / tempo de transferência de dados (ns)	Duração do ciclo (ns)
1980	64K	DRAM	180	150	75	250
1983	256K	DRAM	150	120	50	220
1986	1M	DRAM	120	100	25	190
1989	4M	DRAM	100	80	20	165
1992	16M	DRAM	80	60	15	120
1996	64M	SDRAM	70	50	12	110
1998	128M	SDRAM	70	50	10	100
2000	256M	DDR1	65	45	7	90
2002	512M	DDR1	60	40	5	80
2004	1G	DDR2	55	35	5	70
2006	2G	DDR2	50	30	2.5	60
2010	4G	DDR3	36	28	10	37
2012	8G	DDR3	30	24	0.5	31

Classificação DDR-SDRAM

Caracterização DIMM

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa

Norma	Velocidade do relógio (MHz)	M (transf/s)	DRAM nome	MB/s/DIMM	DIMM nome
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10664	PC10700
DDR3	800	1600	DDR3-1600	12800	PC12800
DDR4	1066-1600	2133-3200	DDR4-3200	17056-25600	PC25600

Leitura sugerida

- *Arquitectura informática: A Quantitative Approach*, Hennessy J.L., Patterson D.A., 6ª Edição, Morgan Kaufmann, 2017
  - Apêndice B: *Revisão da Hierarquia da Memória*
  - Capítulo 2: *Desenho da Hierarquia da Memória*
- *Organização e Arquitectura de Computadores: Designing for Performance*, Stallings W., 10ª Edição, Pearson Education, 2016
  - Capítulo 4: *Memória Cache*
  - Capítulo 5: *Memória interna*
  - Capítulo 6: *Memória externa*
  - *Capítulo* 8: *Apoio ao sistema operativo*