A Memória

Arquitetura de Computadores

Emilio Francesquini
e.francesquini@ufabc.edu.br

2021.Q1

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

Disclaimer

UFABC

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

Introdução

Princípio de localidade

UFABC

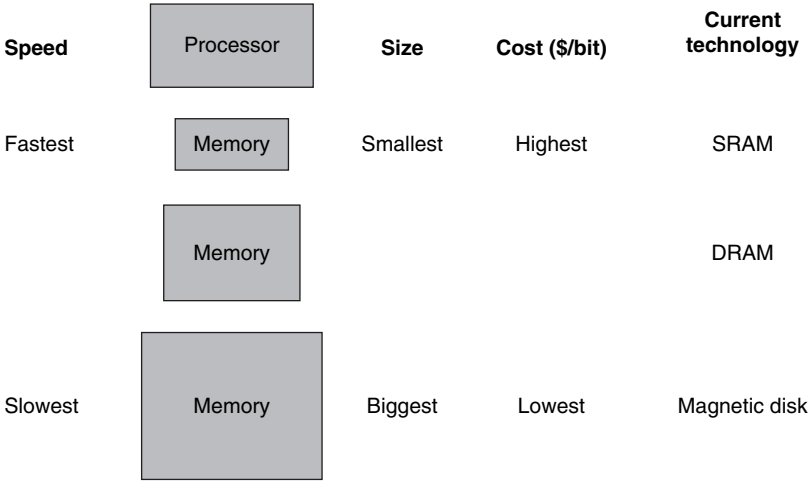
- Programas tendem a acessar uma pequena fatia do espaço de memória disponível.
- Localidade temporal (*en:* ***temporal locality***).
  - Posições da memória que foram acessadas recentemente têm alta probabilidade de serem acessadas em breve novamente.
    - Ex: instruções em um laço, variáveis de indução
- Localidade espacial (*en:* ***spacial locality***).
  - Posições de memória próximas às que já foram acessadas, tendem a ser acessadas em breve.
  - Ex: acesso sequencial de instruções, dados em uma array

Aproveitando o princípio da localidade

UFABC

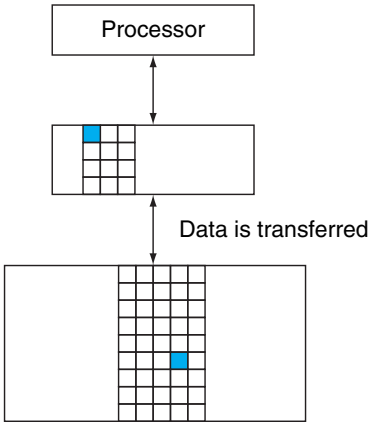
- Utilizando hierarquia de memória
- Guardar tudo no disco
- Copiar dados recentemente acessados (e aqueles próximos dos acessados) do disco para memórias menores, podem mais rápidas (como DRAM - também chamada de **memória principal**).
- Copiar os dados mais recentemente acessados ainda (e aqueles próximos) da DRAM para memória SRAM (também chamadas de caches)
  - Caches são ligadas diretamente à CPU e frequentemente estão no mesmo chip.

Hierarquia de memória



Níveis de hierarquia de memória

- **Blocos** (ou **linhas**): unidade mínima de cópia
  - Podem ser múltiplas palavras do processador
- Se o dado ao qual se deseja acesso estiver nos níveis mais altos, dizemos que houve um **acerto** (*hit*)
  - **taxa de acertos** (*hit ratio*): razão de hits pelo número de acessos
- Se o dado ao qual se deseja acesso não estiver nos níveis mais altos, dizemos que houve uma **falha** (*miss*)
  - **taxa de falhas** (*miss ratio*): razão de misses pelo número de acessos ou 1 - hit ratio
  - No caso de falhas, o dado é recuperado do nivel inferior e fornecido ao nível superior.



Outros termos importantes

UFABC

- Hit time
  - Tempo necessário para acessar um nível de memória da hierarquia incluindo o tempo necessário para determinar se o acesso é um hit ou miss.
- Miss penalty
  - Tempo necessário para trazer um bloco do nível de memória de um nível inferior para o superior, incluindo o tempo para acessar o bloco, transmiti-lo de um nível a outro, colocá-lo no nível onde o miss ocorreu e repassar o bloco para quem o requisitou.

Tecnologias de memória

Tecnologias de memória

UFABC

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $500 – $1000 / GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $10 – $20 / GB
- SSD (Flash)
  - 5.000ns - 50.000ns, $0.75 - $1 /GB
- Disco
  - 5.000.000ns – 20.000.000ns, $0.05 – $0.10 / GB
- Sonho de consumo:
  - Tempo de acesso de SRAM
  - Capacidade e custo / GB de um disco

UFABC

- Dados armazenados em um capacitor
- Um único transistor é usado para acessar o dado
- Precisa ser periodicamente atualizada (*refreshed*)
  - Leituras destrutivas
  - Feitas em uma **linha** (***row***)

Cara da DRAM

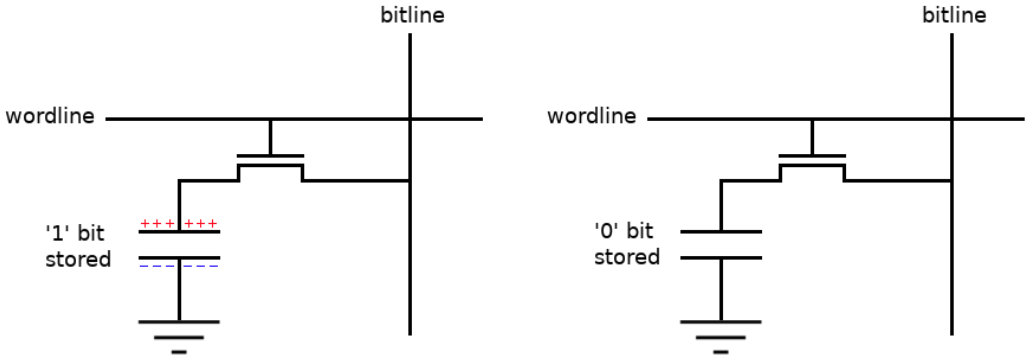


Figura: https://www.allaboutcircuits.com/technical-articles/introduction-to-dram-dynamic-random-access-memory/

Cara da DRAM

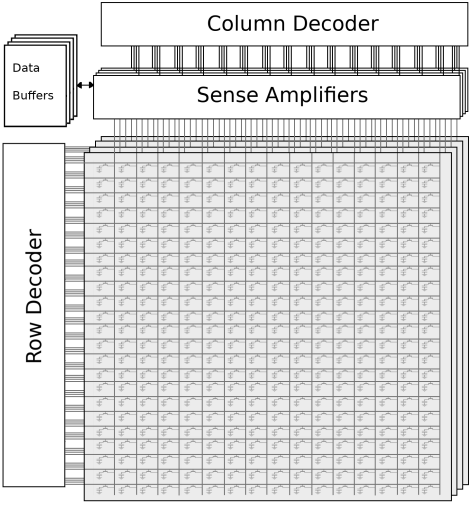


Figura: https://www.allaboutcircuits.com/technical-articles/introduction-to-dram-dynamic-random-access-memory/

Cara da DRAM

Figura: https://www.allaboutcircuits.com/technical-articles/introduction-to-dram-dynamic-random-access-memory/

Organização Avançada da DRAM

- Bits na DRAM são organizados em linhas em uma matriz retangular
  - Quando há um acesso, acessa-se uma linha inteira
  - *Burst mode* ocorre quando acessamos palavras sucessivas de uma mesma linha[1].
- DDR RAM - Double Data Rate
  - Transfere nas bordas ascendentes e descendentes do clock
- QDR RAM - Quad data rate
  - Canais DDR separados para entradas e saídas

[1] Em ingles usa-se as palavras row e line. Row invariavelmente diz respeito à linha da matriz da DRAM e line/block às unidades de transferência entre os níveis hierarquicos da memória

Gerações da DRAM

UFABC

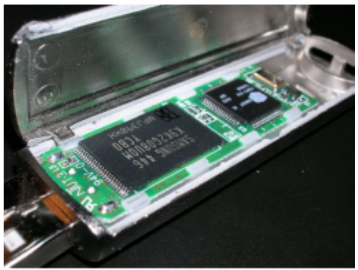
Year introduced	Chip size	$ per GiB	Total access time to a new row/column	Average column access time to existing row
1980	64 Kibibit	$1,500,000	250 ns	150 ns
1983	256 Kibibit	$500,000	185 ns	100 ns
1985	1 Mebibit	$200,000	135 ns	40 ns
1989	4 Mebibit	$50,000	110 ns	40 ns
1992	16 Mebibit	$15,000	90 ns	30 ns
1996	64 Mebibit	$10,000	60 ns	12 ns
1998	128 Mebibit	$4,000	60 ns	10 ns
2000	256 Mebibit	$1,000	55 ns	7 ns
2004	512 Mebibit	$250	50 ns	5 ns
2007	1 Gibibit	$50	45 ns	1.25 ns
2010	2 Gibibit	$30	40 ns	1 ns
2012	4 Gibibit	$1	35 ns	0.8 ns

Fatores de desempenho da DRAM

UFABC

- Row Buffer
  - Permite que várias palavras sejam lidas e atualizadas em paralelo
- Synchronous DRAM - SDRAM
  - Permite acessos consecutivos em bursts sem necessidade de mandar cada endereço
  - Aumenta a banda
- DRAM Banking
  - Permite acesso simultâneo a vários chips de DRAM
  - Aumenta a banda

- Armazenamento não volátil
- De 100 a 1000× mais rápida que discos
- Menor, consome menos energia, mais robusta
- Contudo, mais cara e se posiciona entre disco e DRAM



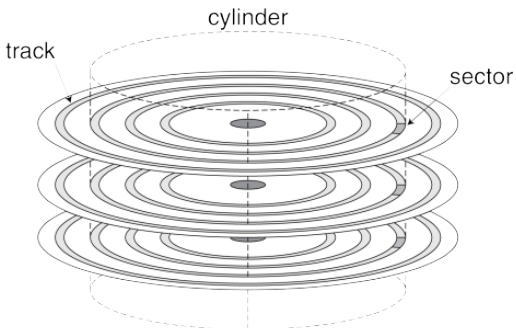
Tipos de memória Flash

UFABC

- NOR Flash: cada célula se assemelha a uma porta NOR
  - Acesso aleatório de leitura/escrita
  - Usada para memória de instruções em sistemas embarcados
- NAND Flash: cada célula se assemelha a uma porta NAND
  - Mais densa, porém só permite acesso em blocos
  - Mais barata por GB
  - Usada para pen drives, SSDs, ...
- Flash se degrada (*wear*) com o uso depois de alguns milhares de ciclos de escrita
  - Não é apropriada para substituir, diretamente, RAM ou disco
  - Mecanismos de *wear leveling* remapeiam dados para os blocos menos acessados

Discos rígidos

- Armazenamento de dados não volátil em discos giratórios magnéticos



Setores do disco e acesso

UFABC

- Cada setor armazena
  - Seu ID
  - 512 a 4096 bytes de dados
  - Códigos de correção de erros (ECC)
    - Usado para esconder defeitos e erros de gravação
  - Campos de sincronização e algumas lacunas
- Acessar um setor envolve
  - Aguardar caso outro acesso já esteja ocorrendo
  - Busca (***seek***): move a cabeça até a trilha desejada
  - Latência Rotacional (***rotacional latency***)
  - Transferência de dados
  - Eventuais overheads do controlador

Exemplo de acesso à disco

UFABC

- Dados
  - setores de 512 bytes, 15.000 RPM, 4ms de tempo médio de busca, 100 MB/s de taxa de transferência, overhead do controlador de 0.2 ms e que o disco não está ocupado
- Tempo médio para leitura
  - 4ms de busca +
  - 0.5 / (15000/60) = 2 ms de latência rotacional +
  - 512 B / 100 MB/s = 0.005ms de transferência +
  - 0.2 ms do controlador =
  - Total: 6.2 ms

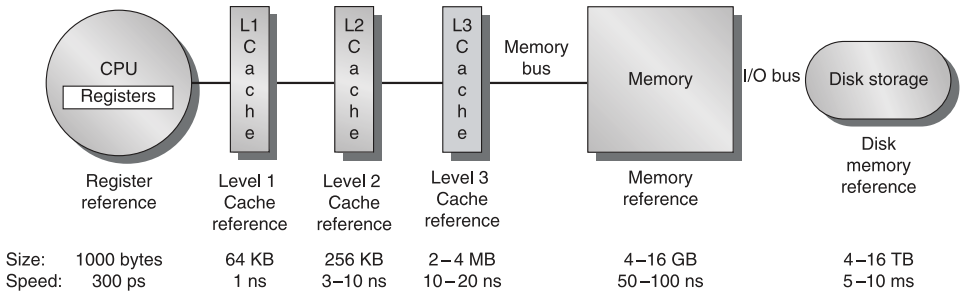
Considerações de desempenho em discos

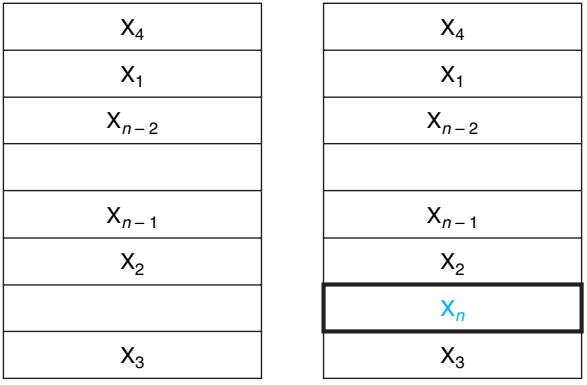
UFABC

- Fabricantes anunciam o tempo de busca médio
  - Baseando-se em todos as buscas possíveis
  - Localidade e o próprio SO acabam levando a um tempo de busca inferior
- Discos frequentemente incluem caches
  - Que buscam antecipadamente setores que podem ser necessários
  - Evitam tempo de busca e latência rotacional

Memória cache

- Memória cache (*cache memory*)
  - Nível de memória mais próximo à CPU





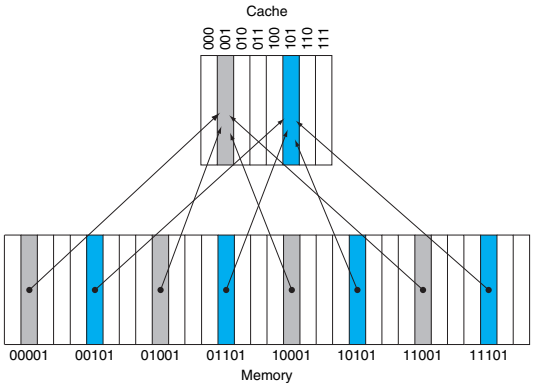
a. Before the reference to X_n b. After the reference to X_n

Dados acessos $X_1, X_2, \ldots X_{n-1}, X_n$

- Como saber se o dado está na cache?
- Como saber onde procurar?

Caches com mapeamento direto

- A localização dentro da cache é determinada pelo endereço
- Em caches com **mapeamento direto** (***direct mapped***) há apenas uma escolha de localização possível.
  - (Endereço da linha) módulo (Nº de blocos da cache)
  - Nº de blocos da cache é uma potência de 2
  - Utiliza-se o bits menos significativos do endereço



UFABC

- Como saber se um bloco/linha está localizada em uma determinada posição?
  - Precisamos guardar a qual posição da memória cada bloco se refere, ou seja, armazenamos tanto o endereço quanto o dado
  - Sendo justo, precisamos apenas os bits mais significativos
    - Esses bits mais significativos são chamados **tag**
- E se não houver nenhum dado guardado naquela posição?
  - Incluímos um bit que indica se a linha é válida
    - Se 1, linha presente. Se 0, linha não presente.
    - Inicalmente todos valem 0.

Exemplo de acesso à cache

UFABC

- Vamos assumir uma cache com 8 blocos/linhas, com mapeamento direto
- O estado inicial é:

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Exemplo de acesso à cache - Acesso 1

Endereço em Decimal	Endereço em Binário	Linha de Cache End. *mod* 8_{10}
22	10110	110

- Miss - Carrega linha

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Exemplo de acesso à cache - Acesso 2

Endereço em Decimal	Endereço em Binário	Linha de Cache End. *mod* 8_{10}
26	11010	010

- Miss - Carrega linha

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Exemplo de acesso à cache - Acesso 3

Endereço em Decimal	Endereço em Binário	Linha de Cache End. *mod* 8_{10}
22	10110	110

- Hit - não há mudança de estado

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Exemplo de acesso à cache - Acesso 4

Endereço em Decimal	Endereço em Binário	Linha de Cache End. *mod* 8_{10}
26	11010	010

- Hit - não há mudança de estado

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Exemplo de acesso à cache - Acesso 5

Endereço em Decimal	Endereço em Binário	Linha de Cache End. *mod* 8_{10}
16	10000	000

- Miss - Carrega linha

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Exemplo de acesso à cache - Acesso 6

Endereço em Decimal	Endereço em Binário	Linha de Cache End. $mod\ 8_{10}$
3	00011	011

- Miss - Carrega linha

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Exemplo de acesso à cache - Acesso 7

Endereço em Decimal	Endereço em Binário	Linha de Cache End. *mod* 8_{10}
16	10000	000

- Hit - não há mudança de estado

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Exemplo de acesso à cache - Acesso 8

Endereço em Decimal	Endereço em Binário	Linha de Cache End. *mod* 8_{10}
18	10010	010

- Miss - Carrega linha

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

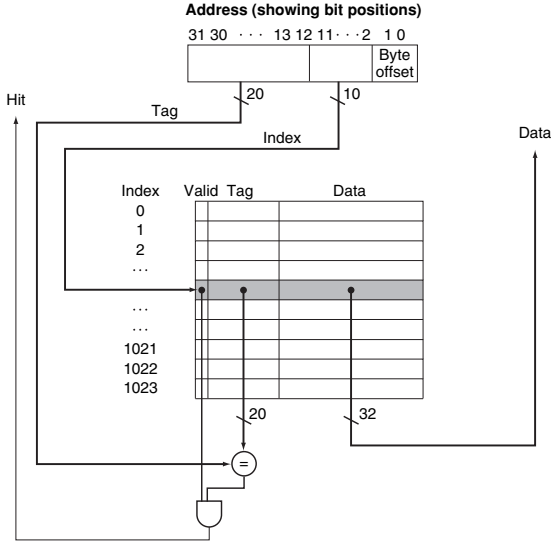
Exemplo de acesso à cache - Acesso 9

Endereço em Decimal	Endereço em Binário	Linha de Cache End. *mod* 8_{10}
16	10000	000

- Hit - Não há mudança de estado.

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Divisão dos endereços

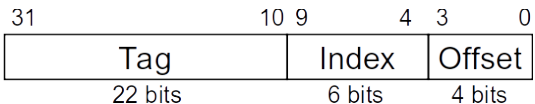


Exemplo com um bloco maior

- Considere uma cache com mapeamento direto com 64 blocos
- Cada bloco com 16 bytes (4 palavras)

Para qual bloco da cache o endereço 1200 deve ser mapeado?

- Bloco = $\left\lfloor \frac{1200}{16} \right\rfloor = 75$
- Nº do Bloco na Cache = $75 \mod 64 = 11$

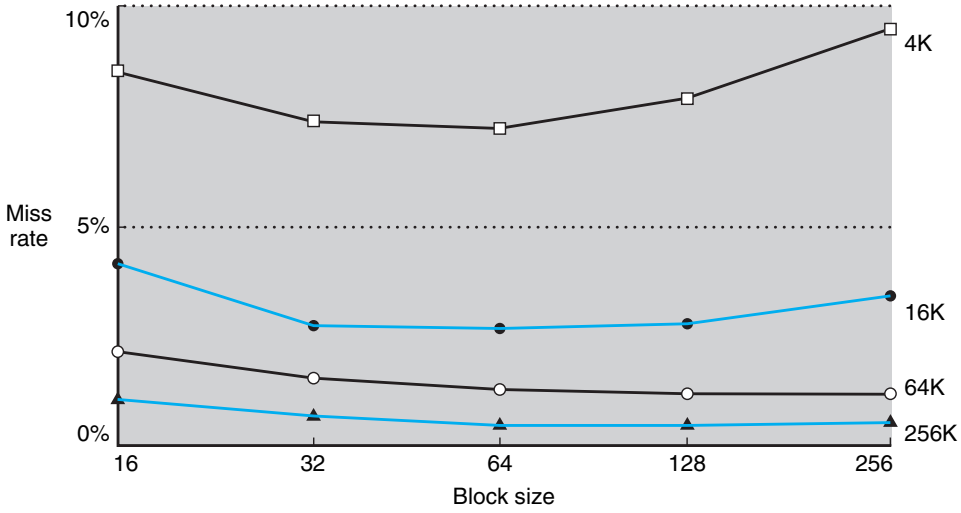


Algumas considerações sobre o tamanho do bloco

UFABC

- Blocos maiores tendem a diminuir a taxa de cache misses
  - Devido a localidade espacial
- Mas, dado um tamanho fixo de cache:
  - Quanto maior o bloco $\rightarrow$ Menos blocos
    - Mais competição por blocos $\rightarrow$ aumento na taxa de cache misses
  - Maior custo de acesso à memória
    - Pode desperdiçar os ganhos de uma taxa de misses mais baixa
    - *Early restart* e *critical-word-first* podem ajudar

Algumas considerações sobre o tamanho do bloco



Resultados do SPEC92

Falhas de cache

- Quando há um cache hit
  - CPU prossegue normalmente
- Quando há um cache miss
  - Stall no pipeline
  - Busca o bloco requerido do próximo nível da hierarquia
  - Miss de instrução
    - Reinicia o fetch da instrução
  - Miss de dados
    - Completa o acesso aos dados

Políticas de escrita na cache - *Write-through*

UFABC

- Quando houver um hit, durante uma escrita, poderia simplesmente atualizar o bloco na cache
  - Mas neste caso a cache e a memória ficariam inconsistentes
- Na política de ***write-through*** quando houver uma escrita também atualizamos a memória
- Isso, contudo, atrasa as escritas
  - Por exemplo: se a o CPI base for 1, 10% das instruções forem stores, e escrever na memória levar 100 ciclos
    - CPI = $1 + 0.1 * 100 = 11$
- Solução: **write buffer**
  - Mantém os dados esperando para ser escritos para a memória
  - CPU continua a execução sem pausas
    - Stalls ocorrem apenas quando o write buffer estiver cheio

- Alternativa à política de write-through: ***write-back***
  - Apenas atualiza a linha na cache
  - Mantém um controle se cada um dos blocos está **sujo** (***dirty***) ou **limpo** (***clean***).
- Quando um bloco sujo precisar sair da cache
  - Escreve o bloco de volta à memória
  - Pode ser usada com um write buffer para permitir que a substituição do bloco sujo por um limpo ocorra antes da escrita na memória

Alocacão de escritas

UFABC

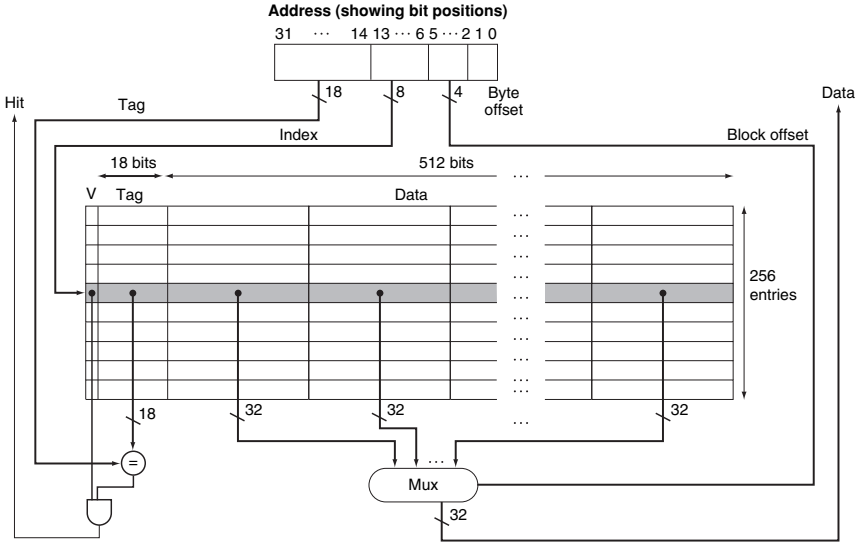
- O que devemos fazer quando queremos escrever em um bloco que não está na cache?
- Write-through
  - *Allocate on miss*: busca o bloco e escreve
  - *Write around*: escreve diretamente na memória
    - Útil pois programas frequentemente escrevem na memória e não leem logo em seguida (ex. inicialização com zeros).
- Write-back
  - Normalmente busca o bloco antes

Exemplo real: O processador Intrinsity FastMATH

UFABC

- É um processador MIPS embarcado
  - Pipeline com 12 estágios
  - Acesso às instruções e dados a cada ciclo
- *Split-cache*: Caches de dados e instruções separadas
  - Normalmente chamadas de I-cache e D-cache
  - No FastMath cada uma tem 16KB: 256 blocos $\times$ 16 palavras/bloco
  - D-cache pode ser write-through ou write-back
- Taxa de cache misses no SPEC2000
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Média ponderada: 3.2%

Exemplo real: O processador Intrinsity FastMATH



Caches são realmente necessárias?

UFABC

- Usamos DRAM para memória principal
  - Normalmente tem largura fixa (ex. 1 palavra)
  - Conectadas por um barramento com clock de largura fixa
    - O clock do bus é tipicamente mais lento do que o clock da CPU
  - Exemplo: leitura de um bloco
    - 1 clock do bus para a transferência do endereço
    - 15 clocks do bus por acesso à DRAM
    - 1 clock do bus por palavra transferida
- Para um bloco de 4 palavras, em uma RAM como acima teríamos
  - Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ ciclos do bus
  - Largura de banda = $16\text{bytes}/65\text{ciclos} = 0.25\text{B/ciclo}$

Vamos olhar com mais cuidado como medir o desempenho da cache

UFABC

- Componentes do tempo de CPU
  - Número de ciclos do programa
    - Já incluem os tempos de hits nas caches
  - Ciclos em stall
    - Em sua maioria por cache misses
- Então, de maneira simplificada

$$\text{Ciclos em stall} = \frac{\text{Acessos à memória}}{\text{Programa}} \times \text{Taxa de misses} \times \text{Miss penalty}$$

que é equivalente a

$$\text{Ciclos em stall} = \frac{\text{Instruções}}{\text{Programa}} \times \frac{\text{Misses}}{\text{Instrução}} \times \text{Miss penalty}$$

Exemplos de desempenho de uma cache

UFABC

- Dados
  - I-cache, taxa de misses = 2%
  - D-cache, taxa de misses = 4%
  - Miss penalty = 100 ciclos
  - CPI base (cache ideal) = 2
  - Loads e Stores compõem 36% de todas as instruções
- Ciclos gastos em misses por instrução
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- CPI real $= 2 + 2 + 1.44 = 5.44$
  - CPU ideal é $\frac{5.44}{2} = 2.72$ vezes mais rápida

Tempo médio de acesso à memória - AMAT

UFABC

- O tempo do hit é essencial a um bom desempenho
- O tempo médio de acesso à memória (**AMAT** na sigla em inglês)
  - AMAT = Tempo Hit + Taxa de Miss × Miss penalty
- Exemplo:
  - CPU com clock de 1ns, tempo hit = 1 ciclo, miss penalty = 20 ciclos, taxa de miss da I-cache 5%
  - AMAT $= 1 + 0.05 \times 20 = 2ns$
  - Ou, 2 ciclos por instrução

Resumo da análise de desempenho

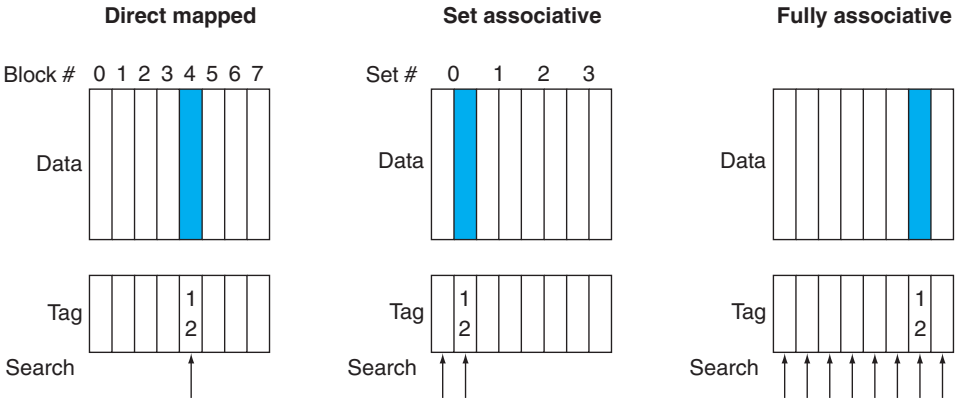
UFABC

- Quanto mais rápida a CPU $\rightarrow$ maior o impacto de cada cache miss
- Aumento do CPI indica uma proporção maior do tempo gasta em stalls causados por acesso à memória
- Conforme a frequência do processador aumenta $\rightarrow$ maior o número de ciclos gastos em stalls
- É essencial que prestemos atenção ao comportamento da cache para avaliar o desempenho do sistema.

Associatividade

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

Exemplo de caches associativas



Espectro da associatividade

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

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

Exemplo de associatividade

UFABC

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

Exemplo: Mapeamento direto

UFABC

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

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

Exemplo: Associativa com 2 vias

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

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

Exemplo: Completamente Associativa

UFABC

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

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

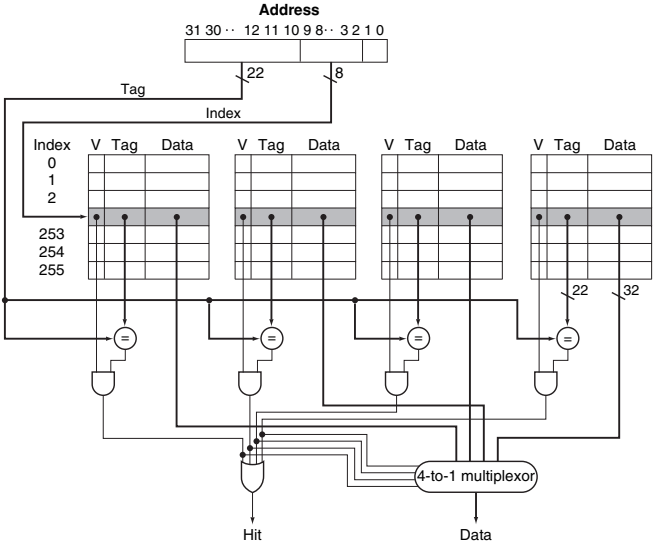
Até onde levar isso?

UFABC

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

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

Organização de Caches Associativas



Política de substituição

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

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

Caches com múltiplos níveis

UFABC

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

Exemplo de cache com múltiplos níveis

UFABC

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

Exemplo de cache com múltiplos níveis (Cont.)

UFABC

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

Algumas considerações sobre caches com múltiplos níveis

UFABC

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

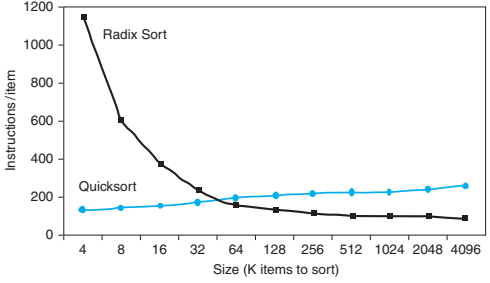
Interações da Cache com Hardware e Software

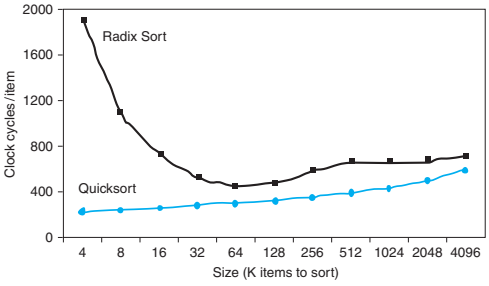
UFABC

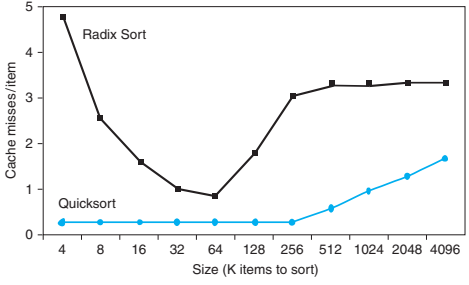
- CPUs OoO são capazes de executar instruções enquanto ocorre um cache miss
  - Stores pendentes ficam armazenados na unidade de load/store
  - Instruções com dependências ficam aguardando
  - Instruções independentes simplesmente continuam
- Efeito dos misses dependem do fluxo de dados do programa
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

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

Exemplo: QuickSort *vs.* Radix Sort







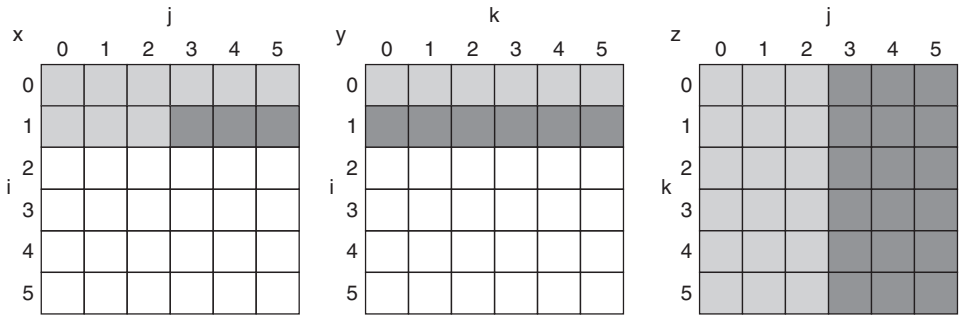
UFABC

- Objetivo principal é aproveitar o máximo os dados na cache antes que eles sejam substituídos
  - blocagem (*cache blocking*)
- Considere os laços internos da DGEMM[2]

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

[2] Double precision GEneral Matrix Multiplication

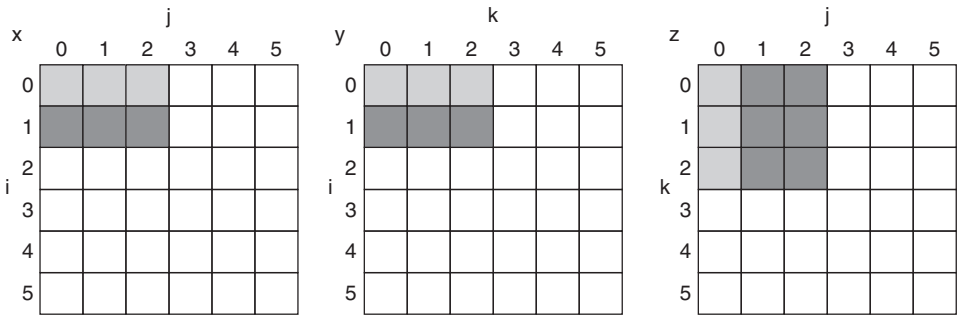
- C, A e = B com `N = 6` e `i = 1`



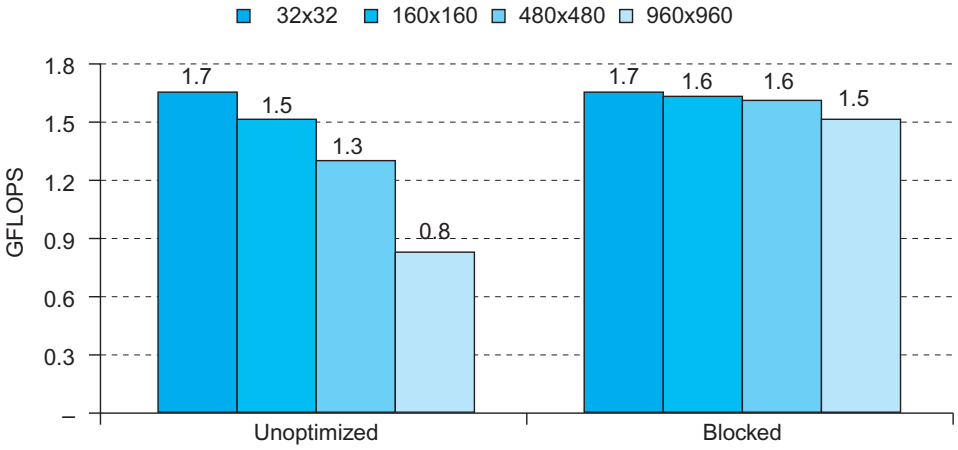
Versão blocada

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

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



Comparação DGEMM - Versão padrão *vs.* Versão blocada



Fundamentos sobre o uso da memória

- O código precisa estar na memória para ser executado, mas o programa completo raramente é usado
  - Código de erro, rotinas incomuns, grandes estruturas de dados
- Todo o código do programa não é necessário ao mesmo tempo

- Considere a capacidade de executar programas parcialmente carregados
  - Programa não seriam mais limitados pelos por limites físicos da memória RAM
  - Cada programa precisa de menos memória durante a execução: assim mais programas podem executar ao mesmo tempo
    - Maior utilização da CPU e taxa de transferência sem aumento no tempo de resposta ou no tempo de retorno
  - Menos E/S necessário para carregar ou trocar programas em memória: logo cada programa do usuário é executado mais rapidamente

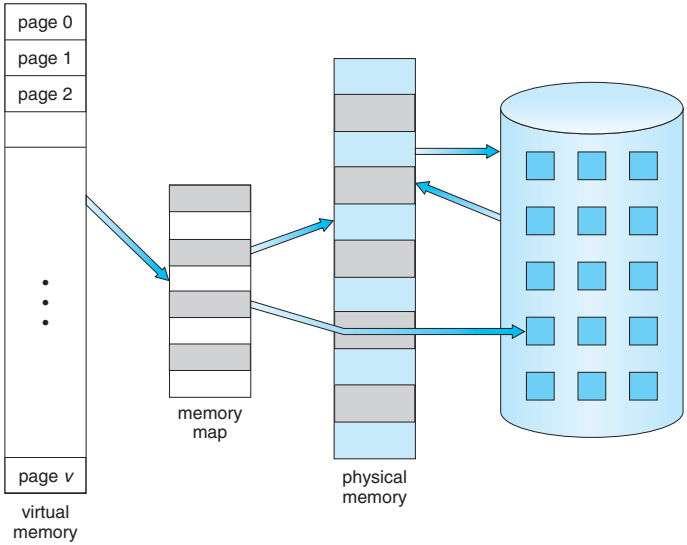
Memória virtual

UFABC

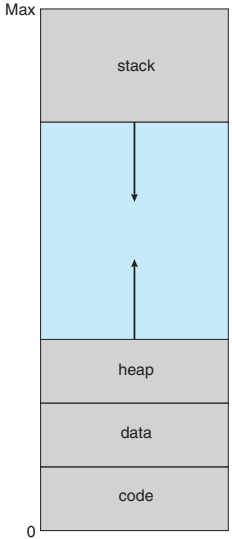
Memória virtual - separação da memória lógica do usuário da memória física

- Apenas parte do programa precisa estar na memória para execução
- O espaço de endereçamento lógico pode, portanto, ser muito maior do que espaço de endereçamento físico
- Permite que os espaços de endereço sejam compartilhados por vários processos
- Permite uma criação de processo mais eficiente
- Mais programas em execução simultaneamente
- Menos E/S necessário para carregar ou trocar processos
- Memória virtual pode ser implementada via:
  - Paginação por demanda
  - Segmentação por demanda

Memória virtual que é maior que a memória física



Espaço de endereço virtual



Espaço de endereço virtual

UFABC

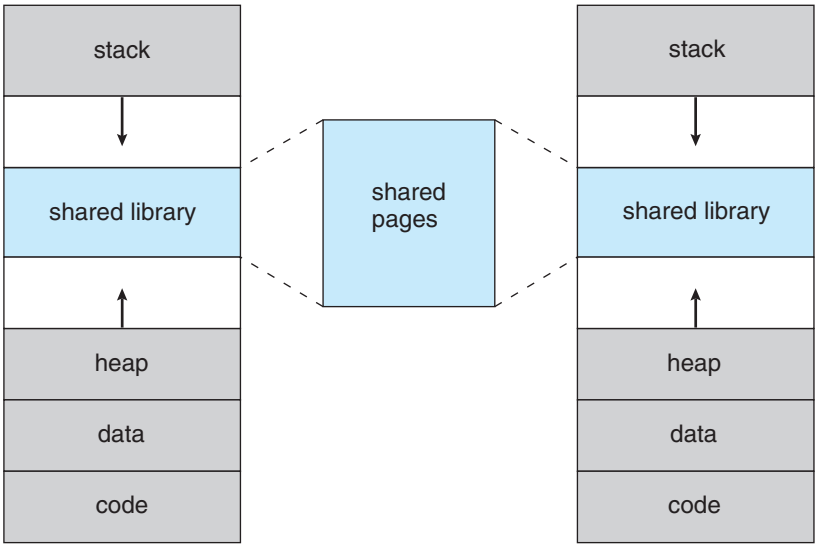
- Geralmente projeta-se o espaço de endereço lógico da pilha para começar no endereço lógico máximo e crescer ”para baixo”enquanto heap cresce ”para cima”
  - Maximiza o uso do espaço de endereçamento
  - Espaço de endereço não utilizado entre os dois é buraco
    - Nenhuma memória física necessária até a pilha ou heap crescerem até uma nova página

Espaço de endereço virtual

UFABC

- Habilita espaços de endereços **esparsos** com buracos deixados para crescimento, bibliotecas ligadas dinamicamente, etc.
- Bibliotecas compartilhadas via mapeamento no espaço de endereçamento virtual
- Memória compartilhada com páginas mapeadas em modo de leitura-escrita no espaço de endereço virtual
- As páginas podem ser compartilhadas durante o `fork()`, acelerando a criação de processos

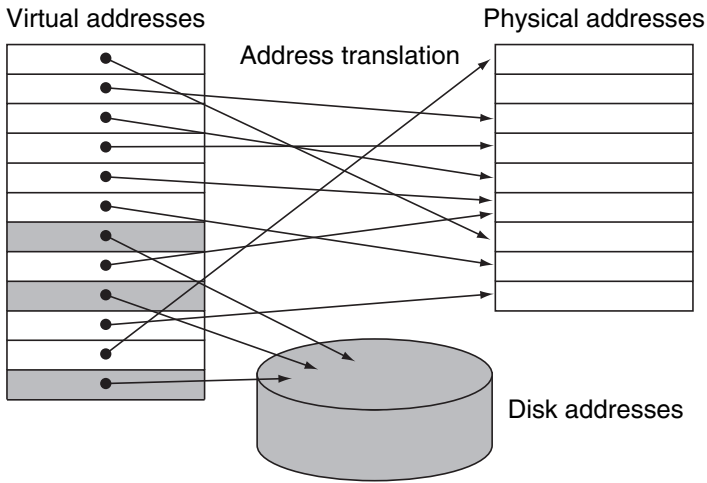
Biblioteca compartilhada usando memória virtual

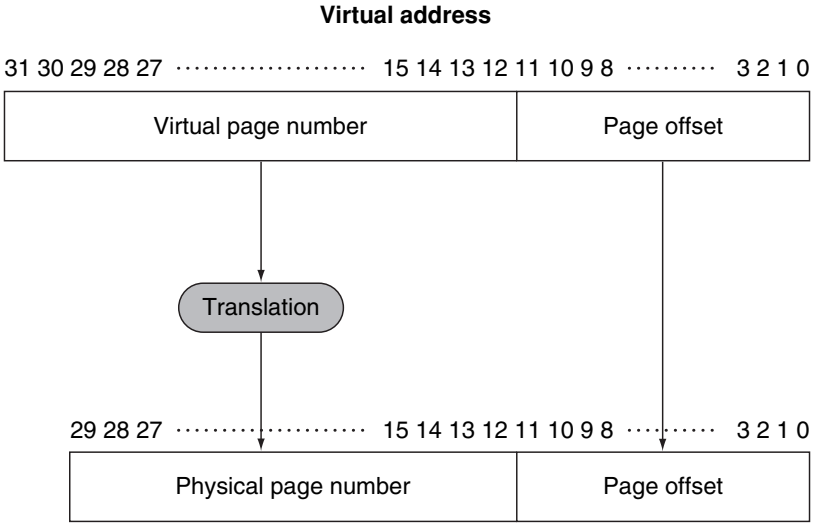


Memória-Virtual - Objetivos

- Usar a memória principal como ”cache”para o armazenamento secundário (disco, SSD,...)
  - Sua manutenção é um trabalho conjunto do SO e da CPU
- Programas compartilham a memória principal
  - Cada programa recebe um espaço de endereçamento virtual que armazena seus dados e código que são utilizados com mais frequência
  - Isola um programa dos demais
- A CPU em conjunto com o SO traduz o endereço virtual para os endereços físicos.
  - Cada ”bloco”da memória virtual é chamada de **página**
  - Quando ocorre um ”miss”de uma página, chamamos de **falha de página** (***page fault***)

- Para tamanhos de página fixos (ex. 4KB)




Virtual address
31 30 29 28 27 ····· 15 14 13 12 11 10 9 8 ····· 3 2 1 0
Virtual page number
Page offset
Translation
29 28 27 ····· 15 14 13 12 11 10 9 8 ····· 3 2 1 0
Physical page number
Page offset
Physical address

Exemplo - Intel Core i7-7500U

UFABC

```
$ lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
Address sizes:           39 bits physical, 48 bits virtual
...
Model name:              Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
...
Virtualization:          VT-x
L1d cache:               64 KiB
L1i cache:               64 KiB
L2 cache:                512 KiB
L3 cache:                4 MiB

$ getconf PAGESIZE
4096
```

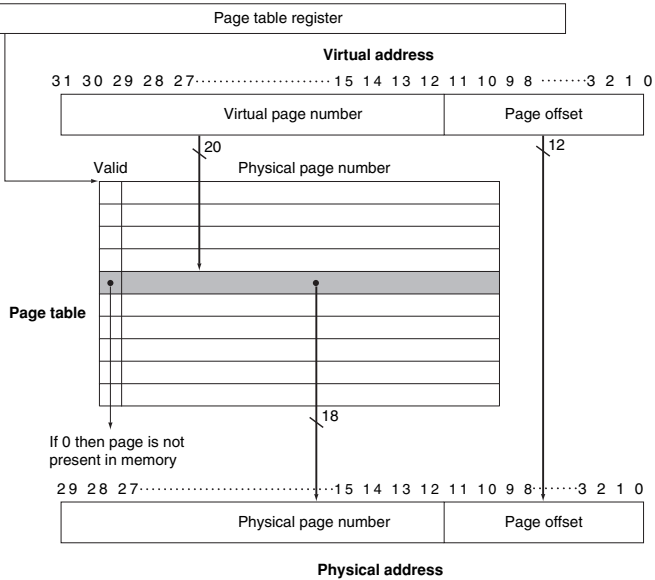
UFABC

- Quando ocorre uma falha de página a página deve ser carregada do disco
  - Isso leva **milhões** de ciclos do processador
  - Tratamento é feito pelo SO
- Então, a estratégia é minimizar ao máximo a **taxa de falhas de página** (***page fault rate***)
  - **Localização totalmente associativa** (***Fully associative placement***)
  - Algoritmos de substituição de páginas inteligentes

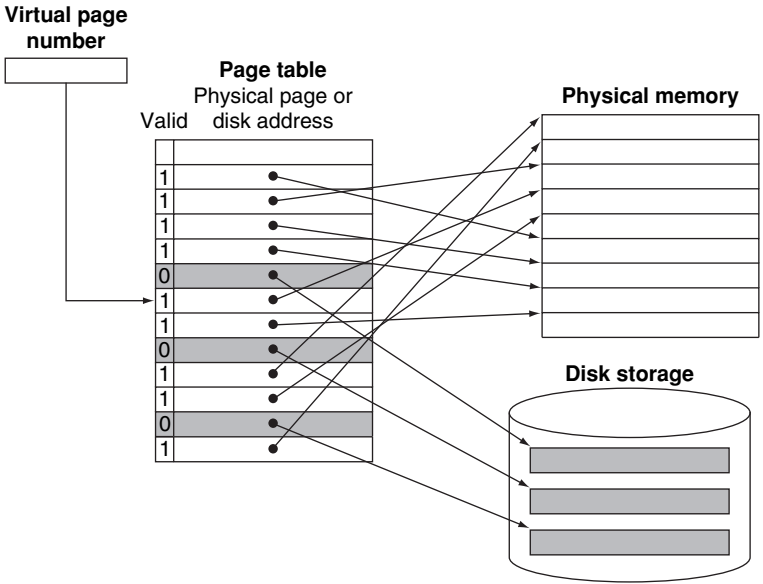
UFABC

- Tabelas de páginas (*page tables*) armazenam informações sobre a localização e estado das páginas
  - Pode ser vista como uma array de páginas indexadas pelo número virtual da página
  - O registrador da tabela de páginas na CPU aponta para a tabela de páginas armazenada na memória física
  - Se uma página está presente na memória
    - A PTE (page table entry) guarda a tradução do endereço virtual para o real
    - Além de outros bits de status como *referenced*, *dirty*, ...
  - Se não estiver presente
    - A PTE pode, por outro lado, apontar para uma localização no espaço de swap armazenado em disco.

Tradução de endereços usando uma tabela de páginas



Mapeando páginas para o dispositivo de armazenamento secundário



Substituição de páginas e escritas

UFABC

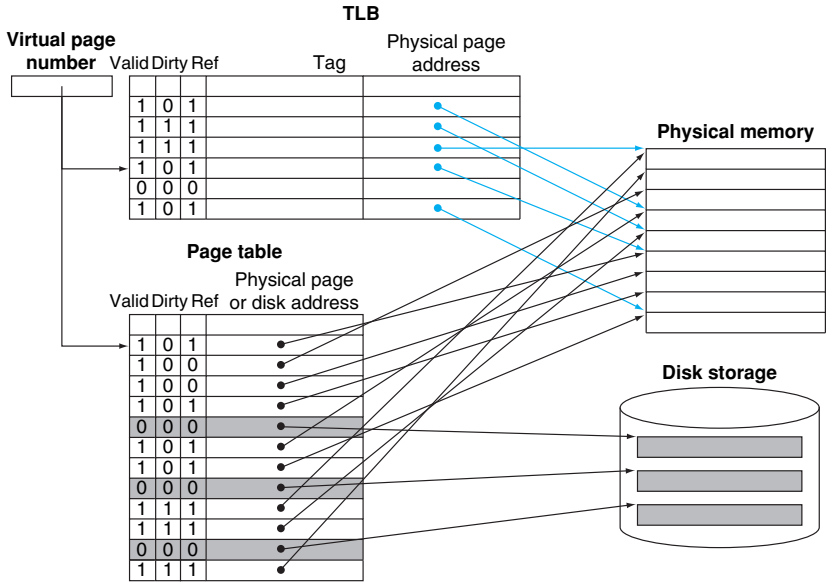
- Para reduzir a taxa de falhas de páginas, prefere-se em geral uma política LRU
  - **Bit de referência** (***reference bit***) é mantido na PTE e setado para 1 sempre que uma página é acessada
  - O SO periodicamente reseta o bits para 0
  - Uma página com o bit de referência 0, portanto, não foi usada recentemente
- Escritas no disco levam milhões de ciclos
  - Escritas ocorrem em blocos, não em posições individuais
  - Utilizar uma política de write-through é impraticável, logo, utiliza-se write-back
  - Bit **sujo** (***dirty***) na PTE é setado quando ocorrem escritas em uma página

Acelerando a tradução usando uma TLB

UFABC

- A tradução de endereços aparenta necessitar acessos adicionais à memória
  - Um acesso para ler a PTE
  - Um acesso para ler o dado propriamente dito
- Contudo, o acesso à tabela de páginas tem uma ótima localidade
  - Utilizamos, portanto, uma cache rápida apenas para as PTEs localizada dentro da CPU
  - Essa cache é chamada de **Translation Look-aside Buffer** (**TLB**)
  - Tamanho típico de 16-512 PTEs, 0.5-1 ciclo por hit, 10-100 ciclos por miss, 0.01%-1% miss rate
  - Misses podem ser tratadas tanto por software quanto por hardware

Acelerando a tradução usando uma TLB



UFABC

- Se a página estiver na memória
  - Carrega a PTE da memória e tenta novamente
  - Pode ser tratada em hardware
    - Pode ficar complexo se as estruturas da tabela de página forem complexas
  - Pode ser tratada em software
    - Levanta uma exceção para o SO que trata a miss
- Se a página não estiver na memória (*page fault*)
  - SO trata o problema carregando a página do disco e atualizando a tabela de páginas
  - Reinicia a execução da instrução que causou a page fault

Tratador de TLB Miss

UFABC

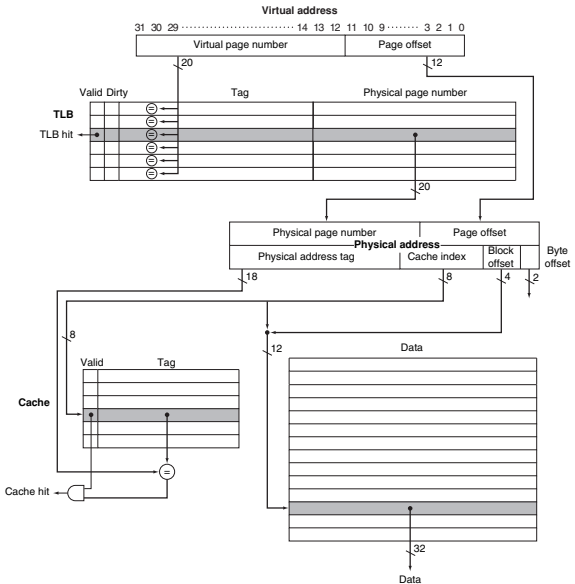
- TLB Miss indica
  - página presente, mas PTE não está na TLB
  - página não presente
- Deve reconhecer o TLB miss antes que o registrador de destino seja sobrescrito
  - Levanta Exceção
- Tratador copia a PTE da memória para a TLB
  - Então reinicia a instrução
  - Se a página não estiver presente, ocorre um page fault

Tratador de Page Fault

UFABC

- Utiliza o endereço virtual que causou a falha para encontrar a PTE
- Localiza a página no disco
- Escolhe a página a substituir
  - Se suja, escreve a página no disco antes
- Lê página para a memória e atualiza a tabela de páginas
- Reinicia a execução do processo
  - Recomeça a partir da instrução que causou a falha

Interação entre a TLB e a cache



- Se a tag da cache usa o endereço físico
  - É preciso traduzir o endereço antes de fazer a busca na cache
- Alternativa: usar o tag do endereço virtual
  - Pode complicar a implementação devido ao **aliasing**: múltiplos endereços virtuais para endereços físicos compartilhados

Exemplo: Intel i7-7500U

UFABC

```
$ cpuid | grep -i tlb
   cache and TLB information (2):
      0x63: data TLB: 2M/4M pages, 4-way, 32 entries
            data TLB: 1G pages, 4-way, 4 entries
      0x03: data TLB: 4K pages, 4-way, 64 entries
      0x76: instruction TLB: 2M/4M pages, fully, 8 entries
      0xb5: instruction TLB: 4K, 8-way, 64 entries
      0xc3: L2 TLB: 4K/2M pages, 6-way, 1536 entrie
      ...
```

- Diferentes tarefas podem compartilhar partes dos seus endereços virtuais
  - Mas precisam ser protegidas de acessos não desejados
  - Necessita assistência do SO
- Suporte de hardware para proteção do SO
  - Modo de execução privilegiada - também conhecido como kernel mode
  - Instruções privilegiadas
  - Tabelas de páginas e outras informações de estado estão disponíveis apenas no modo kernel
  - System call exception (syscall no MIPS)

Resumo – A hierarquia de memória

UFABC

- Princípios comuns aplicados em todos os níveis da hierarquia de memória
  - Baseada em torno da ideia de caching
- Em cada nível da hierarquia
  - Política de disposição de blocos
  - Política de busca de blocos
  - Substituição de blocos em casos de misses
  - Política de escrita

Disposição dos blocos

UFABC

- Determinada pela associatividade
  - Mapeamento direto (associatividade de 1 via)
    - Opção única de disposição
  - Associativa em conjuntos de n-vias
    - n escolhas dentro de um conjunto
  - Completamente associativa
    - Qualquer posição
- Associatividade mais alta diminui a taxa de misses
  - Mas aumenta complexidade, custo e tempo de acesso

Associatividade	Método de busca	Comparações
Mapeamento Direto	Índice	1
Associativa em conjuntos de n-vias	Índice (cjto), entradas no cjto	n
Completamente associativa	Busca todas as entradas Tabela de busca completa	Nº de entradas 0

- Caches de hardware
  - Reduzem as comparações para diminuir o custo
- Memória virtual
  - Tabela de busca completa permite o uso de uma estrutura completamente associativa
  - Se beneficia de uma taxa reduzida de misses

- Escolha pela entrada a ser substituída após um miss
  - LRU
    - Complexo e custoso de fazer em hardware com alta associatividade
  - Aleatório
    - Próximo de LRU na prática, mais fácil de implementar
- Memória virtual
  - Aproximadamente uma LRU com suporte de hardware

- Write-through
  - Atualiza tanto os níveis mais altos quanto mais baixos
  - Simplifica a substituição, mas pode requerer um write buffer
- Write-back
  - Atualiza apenas os níveis mais altos
  - Atualiza o nível mais baixo quando o bloco é substituído
  - Precisa manter mais informações sobre o estado de cada bloco
- Memória virtual
  - Apenas write-back é viável, já que o tempo de aceso ao disco (latência) é muito alto

Fontes de misses

- Misses Compulsórios - ou cold start misses
  - Ocorrem no primeiro acesso a um bloco
- Misses por capacidade
  - Ocorrem pois a cache tem um tamanho finito...
  - ... e um bloco que havia sido carrego num momento anterior já foi substituído por necessidade de espaço
- Misses por conflito
  - Apenas possíveis em caches não completamente associativas
  - Ocorrem quando há disputa por entradas em um conjunto
  - Não ocorreriam em uma cache completamente associativa de mesmo tamanho (seria transformado em que tipo de miss?)

Trade-offs no design de caches

UFABC

Escolha de design	Efeito no miss rate	Desvantagem
Tamanho maior de cache	Diminui misses por capacidade	Pode aumentar o tempo de acesso
Maior associatividade	Diminui misses por conflito	Pode aumentar o tempo de acesso
Maior tamanho de bloco	Diminui misses compulsórios	Aumenta o miss penalty. Quando o tamanho do bloco for muito grande pode aumentar o miss rate devido à poluição da cache

Paralelismo e hierarquia de memória: Coerência de Cache

O Problema da coerência de cache

UFABC

- Suponha que tenhamos 2 CPUs que compartilham o mesmo endereçamento físico
  - Têm caches write-through

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

O que é a coerência de cache

UFABC

- Informalmente, dizemos que a cache é coerente se leituras em um endereço devolvem o valor escrito mais recentemente.
- De maneira mais formal, a cache é coerente se ela satisfizer as seguintes propriedades:
  - Regra 1: Leituras após escritas
  - Regra 2: Propagação de modificações
  - Regra 3: Serialização das escritas

Regra 1: Leituras após escritas

UFABC

- Dados:
  - Um processador *P* escreve *V* na posição *X*
  - Não houve nenhuma escrita nesta posição por qualquer outro processador
- Então:
  - Uma leitura posterior feita por *P* em *X* devolve *V*

Regra 2: Propagação de modificações

UFABC

- Dados:
  - P_1 escreve V em X
- Então:
  - Após algum tempo e automaticamente, quando P_2 ler X deve obter V
- Por exemplo: Se a CPU B ler X após o passo 3 no exemplo anterior.

- Dados:
  - P_1 escreve V_0 em X e;
  - Em seguida, P_2 escreve V_1 em X
- Então:
  - Todos os processadores veem as escritas na mesma
- Isto é necessário pois, de outra maneira as caches de cada processador poderiam terminar com valores diferentes em X.

UFABC

- Protocolos de coerência de cache (*cache coherence protocols*) são operações feitas pelas caches dos multiprocessadores para garantir que as caches sejam coerentes.
- Duas operações principais são utilizadas:
  - Migração de dados para as caches locais
    - Reduz a banda para a memória compartilhada
  - Replicação de dados de leitura compartilhada
    - Reduz a contenção para o acesso

UFABC

Há dois tipos principais de protocolos de coerência de cache:

- Snooping Protocols
  - Cada cache monitora o bus em olhando todos as leituras e escritas
  - Implementação mais simples, porém menos escalável
- Protocolos baseados em diretório
  - Caches e memória armazenam o status de compartilhamento das linhas entre as caches em uma estrutura chamada de diretório
  - Protocolo mais escalável, porém mais complexo e com um overhead mais alto

- Caches tem pelo menos (no protocolo chamado MESI) 2 bits para flags a mais em cada uma de suas entradas que armazenam um dos 4 estados:
  - Modificada (suja) - o bloco foi modificado (está diferente da memória principal) e só está disponível nesta cache
  - Exclusiva - O bloco não está sujo e não há nenhuma cópia em qualquer outra cache
  - Compartilhada - O bloco é o mesmo da memória principal (não foi modificado) e pode estar presente em outra cache
  - Inválida - A entrada não contém dados válidos

- Neste protocolo, as caches obtém acesso exclusivo a um bloco quando desejam escrever
  - Fazem um broadcast no bus avisando a todas as outras caches para invalidarem suas linhas
  - Leituras subsequentes à mesma linha pelas outras caches vão causar um miss
    - A cache que tiver o dado atualizado o fornece para as demais caches (já que elas estão monitorando o bus)

	M Modificada	E Exclusiva	S (*Shared*) Compartilhada	I Inválida
Linha válida?	Sim	Sim	Sim	Não
A RAM está...	desatualizada	válida	válida	-
Há cópias em outras caches	Não	Não	Talvez	Talvez
Uma escrita nessa linha	Fica na cache	Fica na cache	Vai pro barramento e atualiza a cache	Vai diretamente para o barramento

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

- Mais informações veja a seção 17.3 do [WS].

UFABC

- Os **modelos de consistência da memória** definem quando as escritas passam a ser vistas pelos outros processadores
  - ”Ser vista”significa que leituras passam a devolver os valores mais recentemente escritos por outros processadores
  - É inviável que isso seja feito de maneira instantânea
- Algumas premissas:
  - Uma escrita é considerada como completa apenas quando todos os outros processadores já a viram
  - Um processador não reordena operações de escrita com outras operações de acesso à memória
- Consequências:
  - Se P escreve V_0 em X e depois escreve V_1 em Y
  - Então todos os processadores que conseguem ver Y com o valor $V1$ também conseguem ver X com o valor V_0
  - Processadores podem reordenar leituras, mas não escritas

Conclusões

- Endereçamento por bytes ou por palavras
  - Exemplo: cache com 32 bytes, mapeamento direto, blocos de 4 bytes
    - Byte 36 é mapeado para o bloco 1
    - Palavra 36 é mapeada para o bloco 4
- Ignorância dos efeitos do sistema de memória quando escrever ou gerar código
  - Exemplo: fazer a iteração de laços por colunas e não por linhas
  - Saltos muito longos nos vetores resultam em baixa localidade

Pegadinhas 2

UFABC

- Em um multiprocessador com L2 ou L3 compartilhada
  - Menos associatividade do que cores resulta em misses por conflito
  - Quanto mais cores mais associatividade é necessária
- Usar AMAT para avaliar o desempenho de processadores fora de ordem
  - Ignora o efeito de acessos não bloqueantes
  - Em vez disto, o ideal é avaliar o desempenho usando simulação

Conclusões

UFABC

- Memórias rápidas são pequenas, memórias grandes são lentas
  - Gostaríamos de ter memórias rápidas e grandes 😢
  - Caches nos dão essa ilusão 🙂
- Princípio da localidade
  - Programas usam uma pequena parte do seu espaço de memória com mais frequência
- Hierarquia de memória
  - Cache L1 $\leftrightarrow$ Cache L2 $\ldots$ DRAM $\ldots$ Disco
- O design do sistema de memória é crítico para multiprocessadores.