Hierarquia de Memória

7.1 Introdução

* Um programa não acessa todo o seu código ou todos os seus dados de uma vez, pois, se assim fosse, seria impossível acelerar a maioria dos acessos à memória e ainda ter grande capacidade de armazenamento.
* Princípio de Localidade: diz que um programa acessa apenas uma parte relativamente pequena do seu espaço de endereçamento em um certo instante.
  + Localidade Temporal: se um dado é referenciado, ele será referenciado novamente em breve.
  + Localidade Espacial: se um dado é referenciado, dados que possuem endereços próximos a ele tendem a ser referenciados em breve.
* A maioria dos programas contém loops, então as instruções e os dados provavelmente são acessados repetidamente, exemplificando a localidade temporal. Além disso, como as instruções normalmente são acessadas sequencialmente, exemplificamos também a localidade espacial. Outro exemplo de localidade espacial são os acessos aos elementos de um array.
* Hierarquia de Memória: múltiplos níveis de memória com diferentes velocidades e tamanhos; as memórias mais rápidas são mais caras por bit do que as memórias mais lentas, e por isso são menores; quanto maior a distância entre a memória e o processador, maior é o seu tamanho e tempo de acesso.
* O objetivo é utilizar uma memória com a tecnologia mais barata, porém que também tenha a melhor velocidade disponível.
* Uma hierarquia de memória consiste em vários níveis, porém os dados só podem ser transferidos entre níveis de memórias adjacentes.
* Bloco (block): a menor unidade de informação que pode estar ou não presente em um nível de memória.
* Acerto (hit): se um dado que foi requisitado está no nível de memória mais próximo ao processador.
* Erro (miss): se um dado que foi requisitado não está no nível de memória mais próximo do processador.
  + O nível de memória abaixo é acessado para obter o dado necessário.
* Taxa de Acertos (hit rate): fração de acessos à memória que são encontrados no nível de memória mais próximo ao processador; é usado frequentemente como medida do desempenho da hierarquia de memórias.
* Taxa de Erros (miss rate): fração de acessos à memória que não são encontrados no nível de memória mais próximo ao processador.
* Tempo de Acerto (hit time): tempo que leva para acessar o nível de memória mais próximo ao processador, incluindo o tempo necessário para determinar se o acesso é um hit ou um miss.
* Penalidade de Erro (miss penalty): tempo que leva para que o dado não encontrado seja recuperado do nível abaixo de memória para o nível mais próximo ao processador, incluindo também o tempo que leva para que esse dado chegue ao processador.
* Como a memória mais próxima do processador é menor e mais rápida, o tempo de acerto é bem menor do que a penalidade de erro.
* A hierarquia de memória se aproveita da localidade temporal mantendo dados recentemente usados mais próximos ao processador. E se aproveita da localidade espacial movendo blocos que são compostos por várias palavras sequenciais na memória.
* DRAM (Dynamic Random Acess Memory):
  + Baixo custo por bit.
  + Perda de informação após algum tempo: necessidade de refreshing.
* SRAM (Static Random Acess Memory):
  + Pequeno tempo de acesso.
  + Não existe necessidade de refreshing.
  + Alto custo por bit.
* Existem três tipos de tecnologia usadas na hierarquia de memória:
  + A memória principal: implementada como DRAM (dynamic random access memory).
  + Níveis mais próximos do processador (caches): implementados como SRAM (static random access memory).
    - DRAM custa menos por bit do que SRAM, porém é mais lenta.
  + Disco: o maior e mais lento nível na hierarquia.

7.2 O Básico das Caches

* Cache foi o nome escolhido para representar o nível da hierarquia de memória entre o processador e a memória principal.
  + É usado também para se referir à níveis que tiram proveito do Princípio da Localidade.
* Mapeamento Direto (direct mapped): a maneira mais simples de descobrir onde um dado deve ser armazenado na cache; tem como base o endereço que o dado ocupada na memória principal; cada localização na memória principal é mapeada diretamente para exatamente um local da cache.
  + (block address) mod (number of cache blocks in the cache)
  + Esse tipo é bastante usado, pois se o número de entradas da cache for uma potência de dois é possível acessar um dado usando os bits menos significativos.
* Como saber se o dado procurado está na cache: adicionamos um campo chamado de tag na cache; a tag contém os bits mais significativos do endereço do dado na memória principal, ou seja, apenas os bits que não foram usados para indexar a cache.
* Como identificar se um dado na cache contém informação válida: adicionamos um bit chamado bit de validade; ele indica se uma posição da cache contém o dado de um endereço válido; se o bit for 0, aquele dado não é localizado.

**Acessando a Cache:**

* Caso um novo dado tenha que ser armazenado na cache, porém o local para qual ele foi mapeado já esteja ocupado, ele é escrito em cima da informação que já existia no endereço. Esse comportamento faz com que a cache tire proveito da localidade temporal (dado mais recentemente acessado sobrescreve dado mais antigo). Como só existe um local para armazenar o dado (mapeamento direto), só existe um dado à ser sobrescrito.
* Para determinar o endereço de um dado na cache:
  + - 2 bits para o byte offset;
    - x bits necessários para indexar qual o bloco da cache (depende do tamanho da cache);
    - y bits necessários para indexar qual a palavra dentro do bloco da cache (depende da quantidade de palavras dentro de um bloco da cache);
    - z bits para a tag (depende do tamanho do endereço da memória principal);
    - 1 bit para o bit de validade;
  + *Depende tanto da configuração da cache, quanto do tamanho do endereço na memória principal.*
* Aumentando o tamanho do bloco: exploramos a localidade espacial e diminuímos a taxa de erros (miss rate); devemos ter cuidado, pois se aumentarmos demais, o tamanho de um bloco se torna uma fração grande da cache e muitos blocos serão substituídos antes mesmo de todas as suas palavras serem acessadas; também podemos acabar aumentando a penalidade de erro (miss penalty), pois o tempo para carregar um bloco será maior,(pois seu tamanho é maior), ou seja, a melhoria na taxa de acertos não consegue suprir a piora na penalidade de erros, diminuindo o desempenho.
  + Podemos diminuir a penalidade de erro usando o método early restart, que consiste em voltar a executar imediatamente quando o dado chegar na cache, invés de esperar que o bloco todo seja carregado.
    - Método bastante usado para instruções, já que na maioria das vezes são executadas sequencialmente.
  + Outra alternativa é o requested word first/critical word frist, que reorganiza a memória de forma que o dado necessário seja transferido para a cache primeiro; só depois disso é que o resto do bloco é transferido começando pelo endereço logo após o dado solicitado e circulando até o endereço anterior; essa técnica pode ser um pouco mais rápida que o early restart.
* Se o processador não pode acessar a cache porque está ocorrendo alguma transferência, ele deve aguardar (stall).

**Lidando com Erros da Cache:**

* A unidade de controle deve detectar e processar o erro, carregando esse dado da memória principal para a cache.
* Para lidar com erros da cache, além da unidade de controle, usamos um controlador separado que inicia o acesso à memória principal para carregar o dado na cache.
* Para um erro na cache, devemos fazer um stall no processador todo (similar o stall em um pipeline), congelando os componentes dos registradores temporários e dos registradores visíveis ao programador, enquanto aguardamos pela memória principal.
* Se um acesso à uma instrução resulta em um erro, o dado do registrador de instruções é inválido. Logo, devemos carregar a instrução apropriada na cache. Para isso, devemos fazer com que o nível mais abaixo da hierarquia faça uma leitura desse dado.
  + **1.** Enviamos o PC-4 para a memória.
  + **2.** Fazemos uma leitura na memória e esperamos para que o acesso seja finalizado.
  + **3.** Escrevemos o dado na cache (além da tag e do bit de validade).
  + **4.** Reiniciamos a execução da instrução no primeiro estágio, que causará uma nova busca de instrução, dessa vez, encontrando-a na cache.
* O controle da cache em um acesso de dados é basicamente idêntico, em um erro, devemos fazer um stall no processador até que a memória principal responda com o dado.

**Lidando com Escritas na Cache:**

* Digamos que foi realizada uma instrução de store e o dado foi escrito apenas na cache, sem modificar a memória principal. Como a cache e a memória principal agora possuem valores diferentes, dizemos que está ocorrendo uma inconsistência.
* Write-through: sempre escrever o dado tanto na cache quanto na memória principal em instruções de store.
* O que ocorre em um erro de escrita (write miss): primeiro buscamos o bloco da memória principal para a cache; depois que ele é armazenado na cache, fazemos a sobrescrita com o novo valor tanto na cache quanto na memória principal.
* Mesmo esse design sendo bastante simples, ele não fornece um bom desempenho, pois toda escrita de dado causaria uma escrita na memória principal, que é algo bastante demorado.
* Uma solução é o uso de um buffer de escrita (write buffer), que guarda o dado enquanto ele espera para ser escrito na memória principal. Depois que o dado é escrito (na cache e no buffer), o processador continua sua execução normalmente. Quando a escrita na memória é finalizada, o buffer é esvaziado.
  + Se o buffer de escrita estiver lotado quando o processador realiza uma escrita, devemos realizar um stall até que exista uma posição livre no buffer.
* Se a taxa com que a memória completa escritas é menor que a taxa com a qual o processador gera leituras, o método que utiliza buffer de escrita não é eficiente.
* Write-back: quando uma escrita ocorre, o novo valor é escrito no bloco da cache; o bloco modificado só é escrito no nível da hierarquia abaixo quando ele vai ser substituído na cache; aumenta o desempenho, porém é mais complexo.
  + Usa um bit adicional, o dirty bit, para marcar blocos alterados na cache.

**The Intrinsity FastMATH Processor:**

* Split cache: quando um nível da hierarquia é composto por duas caches independentes que operam em paralelo; uma lidando com instruções e outra lidando com dados.
* Vantagens: melhor capacidade de otimizações; menos structural hazards.
* Desvantagem: maior taxa de falta.
* Como ele possui duas caches separadas, são necessários sinais de controles distintos para ler e escrever em cada cache.
  + 1. Envia o endereço para a cache apropriada. O endereço vem do PC (para uma instrução) ou da ULA (para um dado).
  + 2. Se ocorre um acerto, o dado/a instrução necessária está disponível na cache.
  + 3. Se ocorre um erro, enviamos o endereço para a memória principal e esperamos para que a memória retorne o dado/a instrução. Escrevemos na cache e fazemos a leitura normalmente.
* Oferece tanto write-through quanto write-back, o sistema operacional é quem decide qual estratégia usar.

**Sistema de Memória para dar Suporte às Caches:**

* Erros na cache são consertados pela memória principal (DRAM). As DRAMs são feitas com ênfase em densidade e não em tempo de acesso. Porém, podemos reduzir a penalidade de erro se aumentarmos a bandwidth (taxa de transferência de dados) da memória principal para a cache, para que a transferência de blocos seja mais eficiente.
* Essa redução permite que blocos maiores sejam usados sem diminuir o desempenho.
* O processador é conectado à memória principal através de um bus. A taxa de clock do bus é normalmente mais lenta do que a do processador.
* Existem dois métodos para aumentar o bandwidth: tornar a memória mais larga e interleaving.
  + Tornar a memória mais larga: se também tornarmos o bus mais largo, conseguiremos ter um acesso paralelo à todas as palavras de um bloco; diminuímos tanto o tempo de acesso quanto o tempo de transferência; necessidade de multiplexadores.
  + Interleaving: apenas torna a memória mais larga, não alteramos o bus; logo, ainda pagamos um preço para transmitir cada palavra; organizamos os chips da memória em bancos para ler ou escrever múltiplas palavras com um único tempo de acesso ao invés de ler/escrever uma palavra de cada vez; ao enviarmos o endereço para todos os bancos, eles leem simultaneamente; as escritas também podem ser feitas em paralelo; necessita pouco hardware adicional.

7.3 Medindo e Melhorando o Desempenho da Cache

* **CPUTime** é dividido entre os ciclos de clock que o processador usa para executar o programa e os ciclos de clock que o processador passa esperando pelo sistema de memória (na ocorrência de erros/misses).
* **MemoryStallClockCycles** podem ser definidos como a soma de stalls motivados por leituras e por escritas.
* **ReadStallCycles** podem ser definidos em termos do número de acessos de leitura por programa, da penalidade de erro e da taxa de erro.
* **WriteStallCycles** são mais complicados de definir, pois possui duas fontes: um erro na escrita (onde deverá ser feita a leitura do bloco antes de escrever) e stalls quando o buffer de escrita está lotado.
  + Write-Through:
  + Write-Back: a leitura e a escrita possuem penalidades iguais (o tempo que leva para ler um bloco da memória).
* Se o processador é melhorado e se torna mais rápido, porém o sistema de memória continua o mesmo: a quantidade de tempo gasto em stalls da memória vai ser uma porcentagem muito maior do tempo de execução, diminuindo ainda mais o desempenho.
* Aumentando a taxa de clock sem mudar o sistema de memória: aumenta o dano causado no desempenho pelos erros na cache.
* Uma forma de aumentar o tempo de hit é aumentar o tamanho da cache; uma cache maior pode levar mais tempo para ser acessada.
* **Tempo de acesso médio:**

**Reduzindo o Número de Erros com uma Alocação mais flexível dos Blocos:**

* Completamente Associativo (fully associative): um bloco pode ser armazenado em qualquer endereço da cache; para achar um determinado bloco, todas as entradas da cache devem ser checadas, pois o bloco pode estar em qualquer lugar; para tornar a busca mais prática, ela é feita em paralelo com um comparador associado a cada entrada da cache; esses comparadores aumentam o custo do hardware, fazendo com que esse método seja eficiente apenas para caches com um número pequeno de blocos.
* Associativa por Conjunto (set associative): existe um número fixo de locais (pelo menos 2) em que um bloco pode ser armazenado; um set-associative com n locais para um bloco é chamado de n-way set-associative; consiste em um número de conjuntos, onde cada conjunto contém n blocos; cada bloco da memória é mapeado para apenas um conjunto e pode ser armazenado em qualquer elemento desse conjunto; combina o mapeamento direto com o completamente associativo.
  + (block number) mod (number of sets in the cache)
* No método Associativa por Conjunto, as tags de todos os elementos do conjunto devem ser comparadas. Já no método Completamente Associativo, as tags de todos os elementos da cache devem ser comparadas.
* A vantagem de aumentar o grau de associatividade é que isso normalmente diminui a taxa de erros e torna o programa mais rápido.
* Porém, a desvantagem é que pode aumentar o tempo de hit (hit time), além de ser cara por conta das comparações em paralelo.

**Localizando um Bloco na Cache:**

* Associativa por Conjunto: cada bloco em cada conjunto contém a sua tag, como no mapeamento direto; depois do dado ser mapeado em um conjunto, checamos as tags de todos os elementos daquele conjunto para determinar se o dado está presente ou não; agora, o index informa a qual conjunto o bloco pertence; as comparações são feitas em paralelo.
  + Aumentando a associatividade aumentamos o número de blocos por conjunto, que é o número de comparações simultâneas que devem ser feitas. Cada aumento por um fator de dois na associatividade dobra o número de blocos por conjunto e diminui pela metade o número de conjuntos. Diminui o tamanho do index em 1 bit e aumenta o tamanho da tag em 1 bit.
* Completamente Associativa: efetivamente existe apenas um conjunto e todos os blocos devem ser checados em paralelo; como não existe index, o endereço inteiro (menos o block offset, que determina a posição da palavra dentro do bloco) é comparado com todos os blocos, ou seja, possui o papel da tag; em outras palavras, procuramos na cache inteira sem indexar.
* Mapeamento Direto: é necessário apenas um comparador, pois o bloco pode estar em apenas um local; acessamos a cache simplesmente pelo index.
* Os custos de um modelo associativo são os comparadores extras e qualquer possível delay por causa dessas comparações.
* A escolha entre esses três modelos vai depender no custo de um erro e também do custo de implementar a associatividade, tanto em questão de tempo quanto em questão de hardware extra.

**Escolhendo qual Bloco Substituir:**

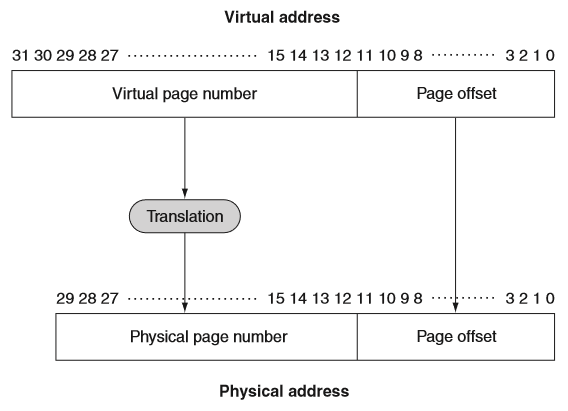
* Mapeamento Direto: o bloco solicitado pode ir para apenas um lugar, então, o bloco que ocupa aquela posição deve ser substituído.
* Em uma cache associativa, temos a escolha de onde posicionar o bloco, logo também temos a escolha de qual bloco substituir.
* Em uma cache Completamente Associativa, todos os blocos são candidatos para a substituição.
* Em uma cache Associativa por Conjunto, devemos escolher dentre os blocos do conjunto selecionado.
* O critério mais utilizado é o **Least Recently Used (LRU)**, o bloco substituído é aquele que não foi usado por mais tempo. Devemos ter o controle de quando cada elemento foi usado por último.
* Outras políticas: First-In-First-Out (FIFO) e Least Frequently Used (LFU).

**Reduzindo a Penalidade de Erro usando Caches Multinível:**

* Para aumentar o desempenho, vários computadores possuem um nível adicional de cache.
* Esse segundo nível é acessado quando um erro ocorre no primeiro nível de cache.
* Se o segundo nível contém o dado desejado, a taxa de penalidade para o primeiro nível será o tempo de acesso do segundo nível, que será muito menor do que o tempo de acesso da memória principal.
* Se nem o primeiro e nem o segundo nível possuem o dado, a memória principal é acessada e uma taxa de penalidade maior se faz presente.
* A penalidade de erro do primeiro nível de cache é significativamente diminuída pela presença do segundo nível de cache, permitindo que o primeiro seja menor e tenha uma taxa de erro maior.
* O primeiro nível de cache é normalmente menor em caches multiníveis. Por causa disso, o primeiro nível utiliza um tamanho de bloco menor.
* O segundo nível de cache é normalmente maior, já que o tempo de acesso não é tão importante. Com um tamanho maior, normalmente, ela utiliza um tamanho de bloco também maior.
* Caches Multinível possuem diversas complicações. Existem diferentes tipos de erros e taxas de erros correspondentes. Além disso, as caches podem ter tamanho de blocos e associatividades diferentes.
* O primeiro nível de cache se preocupa mais com o tempo de acerto (hit time) e o segundo nível se preocupa mais com a taxa de erro (miss rate).

7.4 Memória Virtual

* A memória principal pode agir como uma “cache” para o armazenamento secundário (disco). Essa técnica é chamada de memória virtual.
* Os principais objetivos para a criação dessa técnica foram: permitir um compartilhamento eficiente e seguro de memória entre múltiplos programas; diminuir as restrições que um tamanho limitado de memória oferece.
* Quando vários programas estão rodando ao mesmo tempo, a total quantidade de memória principal necessitada por todos esses programas pode ser muito maior do que a quantidade de espaço disponível. Porém, apenas uma fração dessa memória principal está sendo ativamente usada.
* A memória principal precisa conter apenas as partes ativas dos programas, ou seja, partes que estão sendo executadas no momento.
* A memória virtual nos permite compartilhar a memória principal, ao invés de compartilharmos apenas o processador.
* Para permitir esse compartilhamento, precisamos proteger os programas uns dos outros, assegurando que um programa pode ler e escrever apenas nas porções da memória principal que foram designadas para ele.
* Os programas compartilhando a memória mudam dinamicamente durante suas execuções. Por causa disso, nós restringimos cada programa à um espaço de endereçamento.
* A memória virtual implementa a tradução do espaço de endereçamento do programa para endereços físicos (endereços na memória principal). Essa tradução força a proteção dos espaços de endereçamento de cada programa.
* A memória virtual administra dois níveis da hierarquia de memória: a memória principal e o disco.
* Um bloco de memória virtual é chamado de página. Um erro na memória virtual é chamado de falta de página (page fault), ou seja, quando a página desejada a ser acessada não se encontra na memória principal.
* Com a memória virtual, o processador produz um endereço virtual (corresponde à localização no espaço virtual) que é traduzido por uma combinação de hardware e software para um endereço físico, que é usado para acessar a memória principal.
* Tradução/Mapeamento de Endereço (address translation/mapping): processo que transforma um endereço virtual em endereço físico.
* Realocação (relocation): mapeia os endereços virtuais usados por um programa para diferentes endereços físicos antes do acesso à memória ser realizado; essa alocação nos permite carregar o programa em qualquer lugar da memória.
* O endereço físico é dividido em número da página física (physical page number) e em page offset.
  + O número da página física consiste nos bits mais à esquerda (mais significativos) do endereço físico. Enquanto o page offset consiste nos bits mais à direita (menos significativos).
  + O número de bits no page offset determina o tamanho da página.
  + O número de páginas endereçáveis com o endereço virtual deve ser igual ao número de páginas endereçáveis pelo endereço físico.



* Uma falta de página leva milhões de ciclos de clock para ser processada. Logo, algumas decisões no design da memória virtual são importantes:
  + As páginas devem ser grandes o suficiente para amortecer o tempo de acesso alto.
  + Organizações que reduzem falta de páginas são essenciais, como o posicionamento Completamente Associativo de páginas na memória.
  + Falta de páginas podem ser administradas pelo software (algoritmos).
  + Write-through não funciona, pois escritas demoram demais. Memória virtual usa write-back.
* Segmentação: um endereço é formado por duas partes, o número do segmento e um offset do segmento; o registrador do segmento é mapeado para um endereço físico, e o offset é somado para encontrar o endereço físico real; como os segmentos podem variar em tamanho, é necessário checar se o offset se encontra dentro do segmento.

**Carregando uma Página na Memória Principal e a Encontrando Novamente:**

* Por causa da altíssima penalidade para uma falta de página, tentamos reduzir a frequência de faltas otimizando o posicionamento das páginas.
* Se usarmos o método Completamente Associativo, permitindo que um página seja mapeada para qualquer espaço físico, o SO pode escolher substituir qualquer uma das páginas quando uma falta ocorre.
* O SO pode usar diversos algoritmos de substituição para tentar substituir a página que não será usada por mais tempo.
* A dificuldade para usar o método Completamente Associativo está em localizar a página, já que ela pode estar em qualquer lugar. Por isso, existe uma tabela de páginas (page table) que é mantida na memória principal.
  + Ela é indexada com o número da página virtual para descobrirmos o correspondente número de página físico.
  + Cada programa possui sua própria tabela de páginas.
  + Para indicar a localização da tabela, o hardware inclui um registrador que aponta para o começa dela; chamamos esse registrador de registrador da tabela de páginas (page table register).
* A tabela de páginas, o PC e os registradores consistem no estado (state) do programa. Se quisermos que outro programa use o processador, devemos antes salvar o estado corrente.
* Um bit de validade é usado em cada entrada da tabela de páginas. Se o bit é igual a 0, a página não está presente na memória e ocorre uma falta de página. Se o bit é igual a 1, a página se encontra na memória e essa entrada contém o seu número físico.

**Falta de Páginas:**

* Se o bit de validade de uma página for 0, ocorre uma falta de página. Logo, o SO deve receber o controle (exceções).
* O SO deve achar a página solicitada no nível abaixo da hierarquia (disco) e decidir onde colocá-la na memória principal.
* Apenas o endereço virtual não nos diz onde a página está no disco. Devemos ter controle da localização de cada página no disco.
* Como não sabemos com antecedência quando uma página da memória deverá ser substituída, o SO cria uma espaço no disco para todas as páginas de um processo quando esse processo é criado. Esse lugar no disco é chamado de área de troca (swap area). Nesse mesmo instante, cria também uma estrutura de dados que pode ser ou não parte da tabela de páginas, com o objetivo de gravar onde cada página está guardada no disco.
* Quando ocorre uma falta de página, se todos as páginas na memória estão sendo usadas, o SO deve escolher uma para ser substituída. O SO tenta escolher a página que não será usada novamente por mais tempo. Em geral, é usado o algoritmo Least Recently Used (LRU). Ao encontrar a página menos recentemente usada, a página é escrita na área de troca e é substituída.
* Para ajudar o SO a utilizar o LRU, alguns computadores possuem um bit de referência (use bit/reference bit) para cada página, que é setado sempre que uma página é utilizada. O SO zera periodicamente todos esses bits.
* Existem várias técnicas usadas para reduzir a quantidade de espaço que as tabelas de páginas ocupam:
  + **1.** Manter um registrador limite que restringe o tamanho da tabela para determinado processo. Se o número virtual de página for maior que o valor desse registrador, entradas devem ser adicionadas à tabela. Isso permite que a tabela só cresça em uma direção caso o processo precise de mais espaço.
  + **2.** Dividir a tabela de páginas e deixá-la crescer nos dois sentidos (do maior endereço para baixo e do menor endereço para cima). Nesse caso, existem duas tabelas de páginas e dois registradores limites, um para cada segmento. Os bits mais significativos de um endereço determina qual segmento e qual tabela de página usar.
  + **3.** Tabela de Páginas Invertida: aplicar uma função de hashing no endereço virtual para que a tabela de páginas seja apenas do tamanho da quantidade de páginas físicas na memória; o processo de busca é um pouco mais complicado, pois não estamos apenas indexando cada página.
  + **4.** Tabelas de Páginas Hierarquizadas: o primeiro nível mapeia grandes blocos fixos de endereços virtuais; esse blocos às vezes são chamados de segmentos e esse primeiro nível de mapeamento é chamado de tabela de segmentos; cada entrada da tabela indica se alguma página naquele segmento está alocada e, se está, aponta para a tabela de páginas daquele segmento; a tradução de endereço ocorre olhando primeiro na tabela de segmentos usando os bits mais significativos do endereço; se o segmento é válido, o próximo set de bits mais significativos é usado para indexar a tabela de páginas indicada pela tabela de segmentos.
  + **5.** Dividir a tabela de páginas em páginas: permitir que as tabelas de páginas estejam em espaços de endereçamento virtuais.

**Escritas no nível abaixo da hierarquia:**

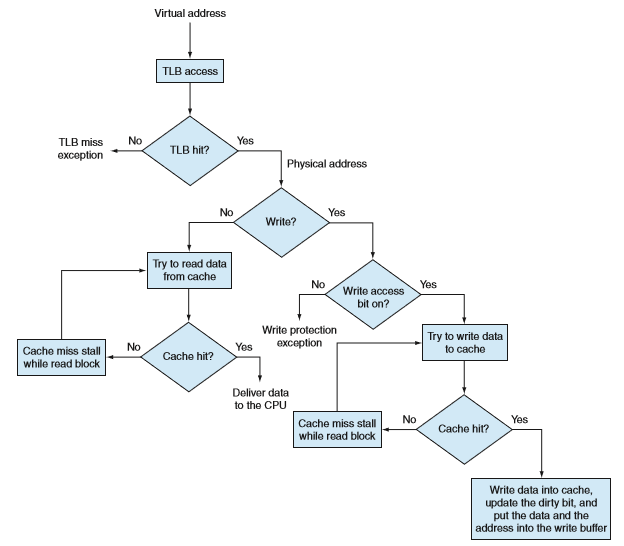
* Escritas no nível abaixo da hierarquia (disco) levam milhões de ciclos de clock, logo, construir um buffer de escrita é completamente impraticável.
* Memória virtual utiliza o método write-back, ou seja, copiar a página para o disco apenas quando ela é substituída na memória. Como copiamos a página, chamamos de copy-back.
* Como a taxa de transferência do disco é menor do que o tempo de acesso, copiar uma página inteira é mais eficiente do que escrever palavra por palavra.
* Mesmo sendo melhor, ainda é custosa. Logo, copiamos para o disco apenas as páginas que foram modificadas, para isso adicionamos um dirty bit à tabela de páginas. Esse bit é setado quando a página é modificada. Então, antes de substituir uma página, o SO deve checar seu dirty bit para copiá-la, ou não, para o disco.

**Acelerando a Tradução de Endereço (TLB):**

* Como as tabelas de páginas são armazenadas na memória, cada acesso na memória por um programa pode levar o dobro de tempo: um acesso para obter o endereço físico e outro para obter o dado em si.
* Para melhorar o desempenho, podemos depender da localidade de referências à tabela de páginas.
* Quando é feita uma tradução de endereço, ela provavelmente será necessária novamente em um futuro próximo.
* Para isso, incluímos uma cache especial que armazena os endereços recentemente traduzidos, chamada translation-lookaside buffer (TLB), evitando acessos à tabela de páginas.
* Cada entrada da TLB possui uma tag que contém uma porção do número da página virtual e uma parte de dado que contém o número da página físico. Além disso, possui também um dirty bit e um bit de referência (reference bit).
* Em toda referência, procuramos o número de página virtual na TLB. Se acontece um acerto (hit), usamos o número de página físico da table e colocamos o bit de referência como 1.
* Se ocorrer um erro (miss), devemos determinar se é uma falta de página ou apenas um erro na TLB.
* Se a página existe na memória, então o erro na TLB indica apenas que a tradução não está salva. Nesses casos, o processador realiza a tradução, salva na TLB e faz a referência novamente.
* Porém, antes de salvar na TLB ele deve selecionar qual entrada substituir. Como temos o dirty bit e o bit de referência, eles devem ser copiados para a tabela de páginas.
* Como erros na TLB são muito mais frequentes do que reais faltas de páginas e não podemos gastar muito com um algoritmo de substituição para a TBL, normalmente a substituição é feita aleatoriamente.
* Porém, se a página não está presente na memória, o processador invoca o SO usando uma exceção.
* Como a TLB possui muito menos entradas do que o número de páginas na memória, erros na TLB são muito mais frequentes do que falta de páginas.
  + Erros na TLB podem ser resolvidos por hardware ou por software.
* São usados vários tipos de associatividades nas TLBs. Para TLBs pequenas é usado o Completamente Associativo, pois o seu custo não é tão grande neste caso. Para TLBs grandes, normalmente é usada associatividades pequenas.

**A TLB do Intrinsity FastMATH:**

* Quando ocorre um erro na TLB, o hardware salva o número da página que foi referenciada em um registrador especial e causa uma exceção.
* O SO é chamado e ele lida com esse erro.
* Para achar o endereço físico da página solicitada, a rotina de erro da TLB indexa a tabela de páginas usando o número de página virtual. Com isso, após calcular o endereço físico, a TLB é atualizada.
* Uma verdadeira falta de página ocorre se a entrada da tabela de páginas não possui um endereço físico válido.
* Além disso, a TLB possui um bit de acesso de escrita (write access bit). Ele previne que o programa escreva em páginas que só podem ser acessadas para leitura. Se o programa tenta escrever em uma página e o bit de acesso de escrita for 0, uma exceção ocorre.



**Integrando Memória Virtual, TLBs e Caches:**

* A memória virtual e o sistema de cache trabalham juntos como uma hierarquia para que o dado só possa estar na cache se estiver na memória principal.
* O SO apaga o conteúdo de qualquer página na cache quando ela é migrada para o disco. Ao mesmo tempo, o SO modifica a tabela de páginas e a TLB, para que uma tentativa de acesso à algum dado da página, gere uma falta.
* No melhor dos casos, um endereço virtual é traduzido pela TLB e enviado para a cache, onde o dado é encontrado e usado pelo processador. No pior dos casos, pode ocorrer um erro (miss) nos três níveis: TLB, tabela de páginas e cache.

**Implementando Proteção com Memória Virtual:**

* Um das funções mais importantes da memória virtual é a possibilidade de compartilhar uma única memória principal com múltiplos processos, e mesmo assim manter a proteção entre todos esses processos e o SO.
* O mecanismo de proteção deve garantir que mesmo múltiplos processadores estarem usando a mesma memória principal, um processo não pode escrever no espaço de endereçamento de outro ou do SO.
* Como cada processo possui seu espaço de endereçamento virtual individual, se o SO mantiver as tabelas de páginas organizadas para que cada página mapeie para páginas físicas separadas, um processo não poderá acessar o dado de outro. O SO deve garantir também que um processo de usuário não possa mudar o mapeamento da tabela.
* Se P1 quer ler uma página de P2, P2 pede ao SO para criar uma entrada na tabela de página de P1 para a página virtual solicitada que aponte para a mesma página física que P2 deseja compartilhar. O SO pode usar o bit de acesso de escrita para proibir P1 de escrever na página de P2.
* Quando o SO decide parar de executar P1 para executar P2 (troca de contexto), ele deve garantir que P2 não possa acessar as páginas de P1. Devemos limpar as entradas da TLB que pertencem a PI. Porém, se a taxa de troca de contextos for alta, isso se torna ineficiente.
  + Uma alternativa é estender o espaço de endereçamento virtual e adicionar um identificador (process identifier/task identifier). O identificador é concatenado com a tag da TLB, para que um acerto na TLB só ocorra se o número da página e o processo forem os mesmos. Essa modificação elimina a necessidade de limpar a TLB em toda troca de contexto.

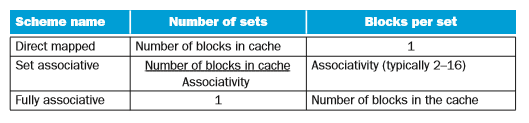
**Lidando com Erros na TLB e Falta de Páginas:**

* Um erro na TLB ocorre quando nenhuma entrada da TLB é compatível com o endereço virtual. Isso pode significar duas coisas:
  + **1.** A página está presente na memória e precisamos criar uma entrada para ela na TLB.
  + **2.** A página não está presente na memória e precisamos transferir o controle para o SO para que ele lide com a falta de página.
* Quando processamos o erro na TLB, procuramos na tabela de páginas para trazer para a TLB. Se a entrada na tabela de páginas possui o bit de validade igual a 0, então a página solicitada não está na memória. Se o bit de validade for igual a 1, podemos simplesmente trazer o endereço virtual para a TLB junto com o endereço físico já calculado.
* Um erro na TLB pode ser resolvido por software ou por hardware.
* O tempo entre o início da rotina de tratamento da exceção e o tempo que o processador leva para salvar todo o estado do processo, o SO fica bastante vulnerável.
* Se outra exceção ocorrer, ela pode sobrescrever o EPC, tornando impossível retornar a execução depois de tratar a primeira exceção.
* Para evitar isso, podemos ativar e desativar exceções. Quando uma exceção ocorrer, o processador ativa um bit que desativa todas as outras exceções.
* Depois de tratar a exceção, podemos ativar as exceções novamente, ou seja, desativar o bit mencionado acima.
* A partir do momento que o SO sabe qual o endereço virtual que causou a falta de página, deve completar três passos:
  + **1.** Procurar na tabela de página a entrada da página que causou a falta e encontrar a localização dela no disco.
  + **2.** Escolher uma página física para substituir; se a página escolhida tiver dirty bit igual a 1, devemos copiá-la para o disco antes de sobrescrever.
  + **3.** Começar uma leitura para trazer a página solicitada do disco para o endereço físico determinado.
* Como o passo 2 e 3 podem demorar muito, normalmente o SO seleciona outra processo para utilizar o processador enquanto são realizados os acessos ao disco.
* Quando o acesso ao disco é finalizado, o SO pode retornar para o processador o processo que causou a falta e executar novamente a instrução que causou a exceção.
* Exceções de faltas de páginas para acessos à dados são difíceis de implementar por três motivos:
  + 1. Elas ocorrem no meio da execução da instrução.
  + 2. A instrução não pode ser finalizada antes da exceção ser tratada.
  + 3. Depois de lidar com a exceção, a instrução deve ser recomeçada como se nada tivesse ocorrido.
* Tornar cada instruções reiniciável (restartable) para que a exceção seja tratada e a instrução reinicie depois é uma solução para esse problema. Como cada instrução escreve apenas no final da execução da instrução, podemos impedir que a instrução escreva e recomeçá-la.
* Técnicas para diminuir a taxa de erros:
  + **1.** Usar maiores tamanhos de páginas para tirar proveito da localidade espacial.
  + **2.** Usar uma tabela de páginas Completamente Associativa para que uma página virtual possa estar localizada em qualquer lugar da tabela.
  + **3.** SO fazer uso de técnicas como LRU junto com um bit de referência para escolher páginas para serem substituídas.
* A diferença de desempenho entre memória e disco diz que se um programa acessa muito mais a memória virtual do que a memória física, ele deve rodar muito lentamente.
  + Esse programa pode estar frequentemente tendo que fazer a troca de páginas (swapping) entre memória e disco. Isso é chamado de trashing.

7.5 Um Framework comum para a Hierarquia de Memórias (Revisão)

**Onde um bloco pode ser armazenado?**

* Existem três formas de posicionamento: Mapeamento Direto, Associativo por Conjunto e Completamente Associativo.
* A vantagem de aumentar o grau de associatividade é que isso normalmente diminui a taxa de erro (miss rate). Isso ocorre pois reduzimos os erros dos blocos que competem pelo mesmo local.



* Com o crescimento do tamanho da cache, a melhora causada por maiores graus de associatividade diminui um pouco, pois a taxa de erro de uma cache maior já é menor, logo, não há muito o que se melhorar.
* A principal desvantagem da associatividade é o alto custo e o tempo de acesso lento.

**Como um bloco é encontrado?**

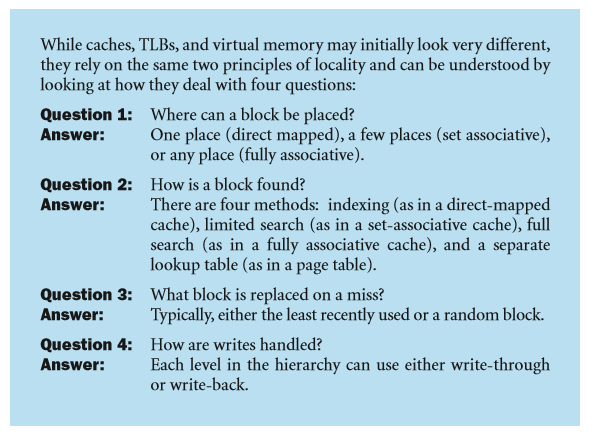
* A escolha de como localizar um bloco depende de que método foi usado para o armazenamento, pois o método nos diz o número de possíveis localizações.
* A escolha entre os três tipos de mapeamento depende do custo de um erro e do custo de implementar a associatividade (tempo de acesso e hardware extra).
* Caches Completamente Associativas são proibidas para tamanhos pequenos, pois o custo dos comparadores não se torna tão grande e a taxa de erro diminui consideravelmente.
* Nos sistemas com Memória Virtual, temos uma tabela de páginas para indexar a memória. Porém, o seu uso requer um acesso à memória adicional (ler tabela + ler dado). A escolha do método Completamente Associativo para o posicionamento na tabela de páginas possui 3 motivos:
  + 1. Como erros são muito caros, precisamos de uma associação completa.
  + 2. O método Completamente Associativo permite que o software use algoritmos de substituição mais sofisticados para reduzir ainda mais a taxa de erro.
  + 3. Um tamanho de página grande garante que o tempo de acesso à tabela de páginas seja relativamente pequeno.
    - Logo, memórias virtuais quase sempre utilizam o método Completamente Associativo.
* Associatividade por Conjunto normalmente é usada para caches e TLBs, onde o acesso combina a indexação e a busca dentro de um conjunto pequeno.
* Alguns sistemas usaram o Mapeamento Direto por causa do seu tempo de acesso rápido e sua simplicidade. O tempo de acesso é melhor, pois para achar um determinado bloco não precisamos realizar comparações.

**Qual bloco devemos substituir num erro na cache?**

* Quando um erro ocorre numa cache Completamente Associativa, devemos decidir qual bloco substituir. Nesse caso, todos os blocos são candidatos.
* Se a cache é Associativa por Conjunto, devemos escolher dentre os blocos de um conjunto.
* Já em caches com Mapeamento Direto, só temos uma opção de qual bloco substituir, tornando o processo mais simples.
* Estratégias para substituição:
  + 1. Aleatório (random): os blocos candidatos são selecionados aleatoriamente, provavelmente usando o próprio hardware.
  + 2. Least recently used (LRU): o bloco que é substituído é o que não foi usado por mais tempo.
    - Na realidade, LRU é muito cara de implementar.
* Em associatividades maiores, LRU é aproximado ou é utilizado o método aleatório.
* Em memória virtual, LRU é sempre aproximado, pois como a penalidade de erro é muito grande, qualquer melhora é significativa.

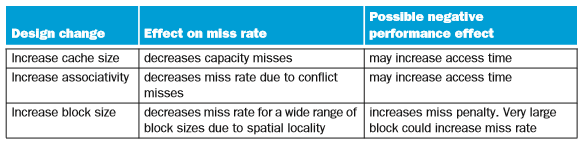
O que ocorre em uma escrita?

* Duas opções básicas:
  + 1. Write-through: a informação é escrita no bloco da cache e no bloco do nível da hierarquia abaixo.
  + 2. Write-back: a informação é escrita apenas no bloco da cache; o bloco modificado só é escrito no nível abaixo de hierarquia quando ele for substituído.
    - Memórias virtuais sempre usam.
* Vantagens do Write-back:
  + As palavras são escritas na memória usando o ritmo do da cache (apenas quando ela está lotada).
  + Múltiplas escritas em um único bloco fazem necessária apenas uma escrita para o nível abaixo de hierarquia.
  + Quando um bloco é escrito no nível abaixo de hierarquia, o sistema pode fazer uso eficiente de uma taxa de transferência alta, já que o bloco inteiro é escrito.
* Vantagens do Write-through:
  + Erros são mais simples e menos custosos, pois eles nunca necessitam que um bloco seja escrito no nível abaixo de hierarquia.
  + É um sistema mais fácil de implementar, apenas de necessitar de um buffer de escrita para se tornar mais eficiente.



**Os três tipos de erros:**

* Erros compulsórios (compulsory misses): erros causados pelo primeiro acesso à um bloco que nunca esteve na cache.
  + Aumentar o tamanho do bloco pode resolver esse problema, porém devemos fazer isso com cautela para não aumentar a penalidade de erro e diminuir o desempenho geral.
* Erros de capacidade (capacity misses): erros causados quando a cache não pode conter todos os blocos necessários durante a execução do programa; quando um bloco é substituído e depois é solicitado novamente.
  + Aumentar o tamanho da cache pode resolver esse problema, porém devemos fazer isso com cautela para não aumentar o tempo de acesso e diminuir o desempenho geral.
* Erros de conflito (conflict misses): erros que ocorrem em um Mapeamento Direto ou Associatividade por Conjunto, onde múltiplos blocos competem pelo mesmo espaço; esse tipo de conflito é eliminado num modelo Completamente Associativo.
  + Aumentar a associatividade pode resolver esse problema, porém devemos fazer isso com cautela para não aumentar o tempo de acesso e diminuir o desempenho geral.



Armazenamento e Entrada/Saída

8.1 Introdução

* Sistemas de E/S focam na dependabilidade e no custo. Enquanto o processador e a memória focam no desempenho e no custo.
  + Dependabilidade: é a propriedade que define a capacidade dos sistemas computacionais de prestar um serviço que se pode confiar (sem perda de informações por exemplo).
* Sistemas de E/S também devem planejar para a possibilidade de expandir o sistema. Isso está relacionado com a capacidade de armazenamento.
* O desempenho depende de vários aspectos do sistema: as características do dispositivo, a conexão entre o dispositivo e o resto do sistema, a hierarquia de memória e o SO.
* Algumas características de sistemas de E/S:
  + Comportamento (behavior): input (ler), output (escrever) ou armazenamento/storage (pode ser re-lido ou re-escrito).
  + Parceria (partner): pode ser um humano ou uma máquina do outro lado do dispositivo; esse parceiro fornece o dado ou lê o dado.
  + Taxa de dado (data rate): a taxa com a qual o dado é transferido entre o dispositivo de E/S e a memória principal ou o processador.
* Para medir o desempenho de E/S dependemos da aplicação. Se a preocupação for a taxa de transferência de dado, devemos avaliar:
  + A quantidade de dado que conseguimos transferir em um intervalo de tempo.
  + Quantas operações de E/S podem ser feitas por unidade de tempo.
* Em outras aplicações, a preocupação é o tempo de resposta:
  + Se os pedidos de E/S são extremamente grandes, o tempo de resposta vai depender da taxa de transferência.
  + Porém, na maioria dos casos, os acessos são pequenos e o sistema de E/S com a menor latência por acesso terá o melhor tempo de acesso.

8.2 Armazenamento em Disco e Dependabilidade

* Um disco magnético é composto por um prato (platter) rotatória coberto por uma superfície magnética que usa uma cabeça (head) móvel para ler e escrever no disco.
* Armazenamento em disco é não-volátil, ou seja, o dado continua armazenado mesmo quando o computador é desligado.
* Um disco magnético é composto por vários pratos (platters), onde cada prato possui duas superfícies de armazenamento.
* Cada superfície é dividida em círculos, chamado de tracks.
* Cada track é dividida em setores (sectors) que contém as informações; um setor é a menor quantidade de dados que pode ser lida ou escrita no disco.
* As cabeças (heads) de cada superfície estão conectadas e se movem juntas, para que toda cabeça esteja sobre a mesma track de todas as superfícies.
* O termo cilindro é usado para se referir à todas as tracks que estão abaixo da cabeça em algum momento.
* Para acessar o dado, o SO deve direcionar o disco à uma série de processos:
  + O primeiro passo é posicionar a cabeça sobre a track desejada. Essa operação é chamada de seek e o tempo que ela leva é chamado de seek time.
    - Esse tempo depende da distância que a track desejada se encontra da cabeça.
  + O segundo passo é esperar para que o disco rotacione e o setor desejado esteja abaixo da cabeça. Esse tempo é chamado de latência rotacional (rotational latency/delay).
    - Normalmente consideramos esse valor o tempo que leva para rotacionar metade do disco.
  + O terceiro passo é o tempo de transferência de um bloco de bits, no caso, um setor.
  + Um controlador de disco (disk controller) é quem administra o controle do disco e as transferências entre ele e a memória.
    - Esse controlador também interfere no tempo de acesso ao disco, o tempo do controlador, que é o overhead que o controlador impõe para realizar uma E/S.
* A maioria dos controladores de disco incluem caches. Essas caches possibilitam o rápido acesso ao dado que foi recentemente transferido. Elas usam o write-through e não atualizam quando ocorre um erro na cache.

**Dependabilidade, Confiabilidade e Disponibilidade:**

* Dependabilidade: é a qualidade de entrega de serviço para que confiança possa ser depositada nesse serviço.
* Para medir a dependabilidade precisamos comparar o comportamento esperado com o comportamento realizado:
  + Realização de Serviço (service accomplishment): o comportamento realizado é igual ao esperado.
  + Interrupção de Serviço (service interruption): o comportamento realizado é diferente do esperado.
  + A transição do estado 1 para o 2 é causada por falhas.
  + A transição do estado 2 para o 1 é causada por restaurações.
* Confiabilidade: é medida pela contínua realização de serviço; ou, equivalentemente, medida pelo tempo para falhas (MTTF).
  + Interrupções de Serviço são medidas pelo tempo para reparar uma falha (MTTR).
  + Tempo médio entre as falhas: tempo para falhas (MTTF) + tempo para reparar uma falha (MTTR).
* Disponibilidade: é medida relacionando a realização de serviços com a alternância entre os dois estados (realização e interrupção).
  + Disponibilidade: tempo para falhas (MTTF) / ( tempo para falhas (MTTF) + tempo para reparar uma falha (MTTR) ).
    - Ou seja, disponibilidade = (MTTF) / (MTTF + MTTR).
* Para aumentar o tempo para falhas (MTTF) podemos melhorar a qualidade dos componentes ou fazer o sistema de forma que continue operando mesmo com componentes em falha.
  + **1.** Evitar falha (fault avoidance): prevenir a falha na construção.
  + **2.** Tolerar a falha (fault tolerance): usar redundância para que o serviço consiga ser realizado independente das falhas; uso de redundância ou bypass de componentes com falhas.
  + **3.** Previsão de falha (fault forecasting): prevendo a presença e a criação de falhas.

**RAID:**

* Redundant Arrays of Inexpensive Disks: organização de discos que usam um array de discos pequenos e baratos para aumentar o desempenho e a confiabilidade.
* Substituindo alguns discos grandes por vários discos pequenos, o desempenho aumentaria, pois existiram mais cabeças para leitura/escrita e existiriam vantagens no custo e no espaço físico ocupado.
* O custo da redundância é pequeno comparado com grandes discos. Logo, a dependabilidade é melhor se você utilizar um array redundante de discos.
* **RAID 0 (No Redundancy):** espalhar um dado em múltiplos discos, dessa forma acessaremos vários discos ao mesmo tempo para obter o dado; esse processo é chamado de striping; isso aumenta o desempenho para grandes acessos, pois vários discos estão sendo acessados ao mesmo tempo.
* **RAID 1 (Mirroring):** também chamado de shadowing; usa o dobro de discos que o RAID 0 usa; sempre que um dado é escrito no disco, ele também é escrito em um disco redundante, para que sempre existam duas cópias da informação; se um disco falha, o sistema vai para o seu “espelho” e lê a informação de lá; é o método mais caro de solução RAID.
* **RAID 2 (Error Detecting and Correcting Code):** usa um método de detecção e correção de erro semelhante aos métodos utilizados nas memórias; caiu em desuso.
* **RAID 3 (Bit-Interleaved Parity):** criação de um protection group de discos; ao invés de ter uma cópia completa do dado original de cada disco, precisamos salvar apenas a informação suficiente para restaurar em caso de falha; leituras e escritas podem ir para qualquer disco do grupo, com um disco extra que guarda a informação para caso ocorra uma falha; paridade (parity) é uma estratégia usada; “é como se o disco redundante tivesse a soma de todos os dados dos outros discos, logo quando uma falha ocorre, subtraímos os dados dos discos bons e encontramos a informação que foi perdida”; muitos discos devem ser lidos para determinar qual o dado que foi perdido; essa técnica assume que demorando mais para recuperar um dado e gastando menos espaço redundante, conseguiremos atingir o maior desempenho.
* **RAID 4 (Block-Interleaved Parity):** usa a mesma lógica do RAID 3, porém acessa os dados de forma diferente; a paridade é guardada como blocos e associada a conjunto de blocos de dados; um pequeno acesso vai para apenas um disco no grupo de proteção enquanto acessos grandes vão para todos os discos do grupo; como a paridade é apenas uma “soma da informação”, em escritas devemos ler o dado antigo do disco, comparar o dado antigo com o novo dado e ver quais bits mudaram, ler a paridade antiga, mudar os bits correspondentes e escrever o novo dado e a nova paridade.
* **RAID 5 (Distributed Block-Interleaved Parity):** no RAID 4, mantendo os bits de paridade em um único disco, pode ocorrer o bottleneck de escritas, ou seja, vários discos tentando atualizar os bits de paridade ao mesmo tempo; para resolver esse problema, podemos espalhar a informação de paridade entre todos os discos; dessa forma, podem ocorrer múltiplas escritas simultaneamente contanto que os blocos de paridade estejam em discos diferentes.
* **RAID 6 (P + Q Redundancy):** pouco usado nos dias de hoje, porém importante para proteção contra falhas de hardware esperadas, erros humanos e falhas relacionadas a problemas ambientais.

8.4 Bus e Outras Conexões entre Processadores, Memória e Dispositivos de E/S

* Em um computador, os vários subsistemas devem possuir interfaces um com os outros.
  + A memória e o processador precisam se comunicar; o processador e os dispositivos de E/S também.
  + Por muitos anos, isso foi feito por meio de bus.
* Um bus é uma área de comunicação compartilhada que utiliza um conjunto de fios para conectar múltiplos subsistemas.
* Vantagens:
  + Versatilidade (novos dispositivos podem ser facilmente adicionados).
  + Baixo custo.
* Desvantagem:
  + Pode criar um bottleneck de comunicação, limitando a máxima taxa de transferência (throughput).
  + A velocidade máxima de um bus é determinada por fatores físicos: o tamanho do bus e o número de dispositivos.
  + Além disso, seu design é complicado pois um único bus deve suportar latências e taxas de transferências diferentes.
* Um bus normalmente é composto por um conjunto de linhas de controle e um conjunto de linhas de dados.
  + As linhas de controle são usadas para enviar sinais de pedidos e de reconhecimentos; além disso, serve também para indicar que tipo de informação está nas linhas de dados; por último, serve para implementar o protocolo do bus (bus protocol).
    - Além desse protocolo, precisamos de outro para determinar quem irá usar o bus após essa transferência.
  + As linhas de dados carregam a informação entre a fonte e o destino. Essa informação pode ser um dado, um comando ou um endereço.
* Transação no Bus (bus transaction): uma sequência de operações que inclui um pedido (envio do endereço) e envio/recebimento da informação.
  + Transação de Leitura (read transaction): transfere dado da memória para o processador ou dispositivo de E/S.
  + Transação de Escrita (write transaction): transfere dado para a memória; realiza escrita.
  + Input: definido pela perspectiva do processador; colocar dado do dispositivo para a memória, onde o processador pode ler.
  + Output: definido pela perspectiva do processador; obter dado da memória em que o processador escreveu para um dispositivo.
* Processor-Memory Buses (buses para processador e memória): são pequenos, com alta velocidade.
* I/O Buses (buses de E/S): são grandes e podem ter vários tipos de dispositivos conectados à eles; normalmente não interagem diretamente com a memória, usam um Processor-Memory Bus ou um Backplane Bus.
* Backplane Buses: feito para permitir que processadores, memória e dispositivos de E/S coexistam em um único bus.
* Existem dois esquemas básicos para a comunicação em um bus:
  + Síncrono: inclui um clock nas linhas de controle e um protocolo fixo para comunicação que é relativo ao clock.
    - Desvantagem: todo dispositivo no bus deve rodar com a mesma taxa de clock.
    - Vantagem: simplicidade e alto desempenho.
    - Processor-Memory Buses normalmente são síncronos.
  + Assíncrono: não possui clock e por isso pode acomodar uma variedade de dispositivos; para coordenar a transmissão de dados entre a fonte e o destino, usamos o handshaking protocol.
  + Handshaking protocol: consiste em uma série de passos onde a fonte e o destino só passam para o próximo passo se os dois concordarem; esse protocolo é implementado com um conjunto adicional de linhas de controle; consideramos as seguintes linhas de controle:
    - ReadReq: usada para indicar uma solicitação de lida da memória; o endereço é colocado nas linhas de dados nesse momento; quem precisa do dado é quem ativa esse sinal.
    - DataRdy: usada para indicar que o dado está pronto nas linhas de dados; quem fornece o dado é quem ativa esse sinal.
    - Ack: usado para sinalizar que o sinal ReadReq ou o sinal DataRdy foi lido pela outra parte da comunicação.
* O bus pode ter um mestre único (o processador):
  + Vantagem: simplicidade.
  + Desvantagem: processador coordena todas as transações de barramento; degradação do desempenho.
* Ou o bus pode ter múltiplos mestres (o processador e alguns dispositivos de E/S):
  + Necessidade de esquema de arbitragem.
* Se dois ou mais dispositivos querem se tornar mestres do barramento ao mesmo tempo, a arbitragem decide qual mestre terá o controle do barramento num dado instante.
  + Arbitragem Centralizada: árbitro decide qual mestre controla o barramento.
  + Arbitragem Descentralizada: arbitragem distribuída por seleção própria

8.5 Interface do Dispositivo de E/S com o Processador, a Memória e o Sistema Operacional

* O SO tem um papel gigante na administração de E/S. Ele age como uma interface entre o hardware e o programa que solicitou E/S.
* Características de sistemas de E/S:
  + **1.** Múltiplos programas usando o processador compartilham o mesmo sistema de E/S.
  + **2.** Sistemas de E/S usam interrupções para se comunicar sobre as operações.
  + **3.** O controle de um dispositivo de E/S é complexo.
* Funções que o SO deve fornecer:
  + **1.** Deve garantir que um programa de usuário tenha acesso apenas à partes do dispositivo de E/S as quais ele possui direito de acessar.
  + **2.** Deve prover abstrações para o acesso dos dispositivos.
  + **3.** Deve lidar com as interrupções gerados pelos dispositivos.
  + **4.** Deve prover a mesma possibilidade de acesso aos dispositivos de E/S para todos os programas. Além de organizar os acessos em ordem para melhorar a taxa de transferência.
* Para que o SO seja capaz de realizar essas funções, ele precisa poder se comunicar com os dispositivos de E/S e prevenir que os programas de usuário se comuniquem diretamente com o dispositivo. Três tipos de comunicação são necessárias:
  + **1.** O SO deve ser capaz de dar comandos para os dispositivos de E/S.
  + **2.** O dispositivo deve ser capaz de notificar o SO quando uma operação de E/S foi finalizada ou quando ocorre um erro.
  + **3.** O dado deve ser transferido entre a memória e um dispositivo de E/S.

**Dar Comandos para os Dispositivos de E/S:**

* Para dar um comando para um dispositivo de E/S, o processador deve ser capaz de enviar uma ou mais palavras de comando.
* Memory-mapped I/O: parte do espaço de endereçamento é separado para dispositivos de E/S; escritas e leituras nesses endereços são interpretadas como comandos para o dispositivo.
* Special I/O Instructions: as instruções especificam o número do dispositivo de E/S além da palavra de comando (ou local desse comando na memória); o processador se comunica com o dispositivo através de fios normalmente incluídos como parte do bus; o comando em si pode ser transmitido pelas linhas de dados.

**Comunicação com o Processador:**

* Pooling: o processo de fazer checagens periódicas nos bits de status para ver se já está na hora da próxima operação de E/S; é a maneira mais simples de comunicação; o dispositivo coloca a informação em um registrador de status (status register) e o processador deve ir buscar a informação; o processador está em total controle e faz todo o trabalho.
  + Desvantagem: pode desperdiçar muito tempo do processador, pois operações de E/S são muito lentas.
  + Vantagem: E/S controlada pela CPU; simplicidade da implementação.
* Interrupt-Driven I/O: uso de interrupções para notificar para o processador quando um dispositivo de E/S precisa de atenção; interrupção assíncrona; para comunicar ao processador qual o dispositivo que causou a interrupção podemos usar um vetor de interrupções ou um registrador de causa da exceção (Cause).
  + Vantagem: melhor utilização da CPU.

**Níveis de Prioridade da Interrupção:**

* Para lidar com diferentes prioridade de dispositivos de E/S, a maioria dos mecanismos de interrupções possui vários níveis de prioridade. Essas prioridades indicam em que ordem o processador deve processar as interrupções.
  + Interrupções de E/S normalmente tem menor prioridade.

**Transferindo dado entre o Dispositivo e a Memória:**

* Tanto Polling quanto Interrupt-Driven I/O coloca toda a responsabilidade no processador.
* Uma alternativa é realizar a transferência de dados por meio de várias interrupções.
* O SO transfere apenas um pequeno número de bytes do dado por vez para ou do dispositivo. Enquanto o dado está sendo lido ou escrito, o SO pode realizar outra tarefa nesse meio tempo.
* Quando o SO percebe uma interrupção desse dispositivo, ele olha o status procurando por erros. Se não existem, o SO pode passar a próxima parte do dado.
* Quando o último byte do dado é transferido e a operação de E/S foi finalizada, o SO pode informar o programa.
* O processador e o SO fazem todo o trabalho nesse processo, acessando o dispositivo e a memória para cada parte do dado transferida.
* Para realmente remover o papel do processador, foi criado o **Direct Memory Access (DMA)**. Consiste no fato de que o dispositivo transfira o dado diretamente para a memória (ou leia dela) sem envolver o processador.
  + Ainda usamos interrupção, porém apenas para avisar quando a operação foi finalizada ou quando ocorre um erro.
  + É implementada com um controlador especializado que transfere o dado entre o dispositivo e a memória. Esse controlador se torna o bus master (unidade no bus que pode fazer solicitações).
  + Pode ser usado como uma interface para o disco sem consumir o processador por diversos ciclos para uma única operação de E/S.
  + Existem três passos:
    - **1.** O processador inicializa a DMA fornecendo a identidade do dispositivo, a operação que é pra ser feita, o endereço da memória que é a fonte ou destino da transação e o número de bytes que serão transferidos.
    - **2.** A DMA começa a operação no dispositivo. Quando o dado está disponível, ela realiza a transferência.
    - **3.** Quando a tarefa foi finalizada, realiza uma interrupção para sinalizar ao processador.

Multiprocessadores e Clusters

9.1 Introdução

* Thread: pode ser parte de um programa paralelo de múltiplos processos, ou pode ser um programa independente; cada thread tem seu estado (instruções, dados, PC, register state, etc) necessários para sua execução.
* Data Level Parallelism: realizar operações idênticas sobre um conjunto de dados.
* Origem dos multiprocessadores: criar computadores poderosos simplesmente conectando muitos computadores menores existentes.
* Como o software é escalável, alguns multiprocessadores podem suportar operar mesmo com a ocorrência de quebras no hardware; ou seja, se um único processador falhar em um multiprocessador com n processadores, o sistema continua fornecendo o serviço com n-1 processadores.
* Os multiprocessadores possuem o desempenho absoluto mais alto, ou seja, são mais rápidos do que o uni-processador mais rápido que existe.
* Como o microprocessador é hoje o processador mais econômico, as melhores alternativas para atingir um maior desempenho são: um multiprocessador ou um cluster.
* Cluster: composto por vários microprocessadores conectados por uma rede local que funciona como um único e grande multiprocessador.
* Como os processadores operando em paralelo compartilham dados, eles também precisam coordenar quando estão operando em dados compartilhados; caso contrário, um processador poderia começar a processar dados antes que outro tenha acabado de processá-los.
* Essa coordenação é chamada de sincronização; um método usado é o lock:
  + Apenas um processador de cada vez pode adquirir o lock, e outros processadores interessados em compartilhar dados precisam esperar até que o processador original desbloqueie a variável.

**Como os processadores paralelos se comunicam?**

* Os processadores compartilham um único espaço de endereçamento e se comunicam por meio de variáveis compartilhadas na memória, com todos os processadores sendo capazes de acessar qualquer local da memória por meio de loads e stores.
* Os multiprocessadores de espaço de endereçamento único podem ser de dois estilos:
  + Uniform Memory Access (UMA): leva o mesmo tempo para acessar a memória principal independente de qual processador o requisita e de qual word é requisitada; poucos processadores.
  + Nonuniform Memory Access (NUMA): alguns acessos à memória são mais rápidos do que outros, dependendo de que processador pede que word; possuem um desempenho mais alto; maior número de processadores.
* Um modelo alternativo de comunicação usa a troca de mensagens. Ela é necessária para máquinas com memórias privadas, como por exemplo um cluster.
  + Processadores em diferentes computadores desktop se comunicam trocando mensagens por meio de uma rede local; são necessárias rotinas de envio e de recepção de mensagens.
* Os multiprocessadores são construídos em duas organizações básicas:
  + 1. Conectados por um único barramento.
  + 2. Conectados por uma rede.

9.2 Programando Multiprocessadores

* A má notícia é que ainda não se sabe quantas aplicações importantes serão executadas mais rapidamente em multiprocessadores por meio de processamento paralelo, pois poucos programas de aplicação têm sido reescritos para completar tarefas mais rapidamente em multiprocessadores.
* Por que programas de processamento paralelo são mais difíceis de desenvolver?
  + É necessário obter um bom desempenho e eficiência do programa em paralelo em um multiprocessador, pois, caso o contrário, você usaria um uniprocessador, já que é mais fácil de programar.
  + Alguns uniprocessadores (superescalar) tiram vantagem do paralelismo em nível de instrução. Tal inovação reduz a necessidade de reescrever programas para multiprocessadores.
  + Outro motivo é o overhead de comunicação. “Embora n pessoas possam ter o potencial para concluir qualquer tarefa n vezes mais rápido, o overhead de comunicação para o grupo pode impedir isso.”
  + Por último, o programador precisa saber muito sobre o hardware da máquina.

9.3 Multiprocessadores Conectados por um único Barramento

* Vários microprocessadores podem ser convenientemente colocados em um barramento comum por várias razões:
  + 1. Como cada microprocessador é muito menor do que um processador multichip, mais processadores podem ser colocados em um único barramento.
  + 2. As caches podem reduzir o tráfego de barramento.
  + 3. Foram inventados mecanismos para manter as caches e a memória consistentes para multiprocessadores.
* Replicação: para dados compartilhados que estão na memória, são feitas cópias nas caches que estão usando o dado; isso reduz a latência de acesso e utilização simultânea do barramento.
* Migração: o valor mais atualizado da variável é movido entre a cache que tem o valor mais atual e a cache que precisa do dado.
* Coerência de cache: consistência no valor dos dados entre as versões nas caches de vários processadores.
* E/S pode experimentar inconsistências no valor dos dados entre a versão na memória e a versão na cache. Esse problema de coerência de cache se aplica para multiprocessadores também.
* Múltiplos processadores normalmente requerem cópias dos mesmos dados em múltiplas caches.
* Uma solução para esse problema seria fazer com que os dados compartilhados fossem sempre escritos na memória quando modificados (write-through), porém isso deixaria o computador muito lento.
* Os multiprocessadores comerciais baseados em cache usam caches write-back, porque elas reduzem o tráfego de barramento e, portanto, permite mais processadores em um único barramento.
* Os protocolos para manter coerência em múltiplos processadores são chamados de protocolos de coerência de cache.
* **Snooping:** todos os controladores de cache monitoram o barramento para determinar se possuem uma cópia do bloco compartilhado.
  + Os processadores precisam ter a cópia mais recente do dado quando estão lendo, logo, após uma escrita, todos os processadores precisam obter o valor novo.
  + O protocolo de snooping precisa localizar todas as caches que compartilham um objeto a ser escrito.
  + Após a escrita, existem duas alternativas:
    - Invalidar todas as outras cópias.
    - Atualizar todas as outras cópias com o valor que foi escrito.
* Write-invalidate: o processador de escrita faz com que todas as cópias em outras caches sejam invalidadas antes de mudar sua cópia local; o processador de escrita ativa um sinal de invalidação no barramento e todas as caches verificam se elas possuem uma cópia; se possuírem, elas precisam invalidar o bloco contendo a word.
  + Esse esquema permite vários leitores, mas apenas um escritor.
  + Método que os processadores atuais usam.
* Write-update: as cópias das demais caches são atualizadas após a ocorrência de uma escrita em uma das caches; causam maior utilização do barramento.
* *O que acontece se dois processadores tentarem escrever na mesmo word de dados compartilhada no mesmo ciclo de clock?*
  + O árbitro do barramento decide qual processador toma o barramento primeiro e esse processador invalidará ou atualizará a cópia do outro, dependendo do protocolo. O segundo processador, então, realiza sua escrita.
* Cada bloco de uma cache está em um de três estados:
  + **1.** Compartilhado (apenas leitura): esse bloco de cache é limpo (não escrito) e pode ser compartilhado.
  + **2.** Modificado (leitura/escrita): esse bloco de cache é sujo (escrito) e pode não ser compartilhado.
  + **3.** Inválido: esse bloco de cache não possui dados válidos.
* Falha de Leitura: se a cache está lotada e deseja ler um bloco que não está presente nela; o bloco que deverá ser substituído é escrito na memória se estiver no estado Modificado; todas as caches dos outros processadores monitoram a falha de leitura para ver se esse bloco está em sua cache; se tiver alguma cópia e ela estiver no estado Modificado, o bloco é escrito na memória e marcado como Compartilhado; a falha de leitura é satisfeita lendo o bloco da memória e o bloco é definido como Compartilhado; a leitura é feita da memória independentemente; (NO SLIDE DE EDNA DIZ QUE NÃO OCORRE ACESSO À MEMÓRIA, A CACHE QUE TEM O VALOR ATUALIZADO SIMPLESMENTE PASSA O NOVO VALOR PARA A CACHE QUE PRECISA, PORÉM O LIVRO DIZ OUTRA COISA).
* Acertos de Escrita: um acerto de escrita em um bloco Modificado não altera em nada; já em um bloco Compartilhado, faz com que a cache adquira o barramento e envie um sinal de invalidação para bloquear quaisquer outras cópias; o bloco é alterado e seu estado muda para Modificado.
* Falhas de Escrita: uma falha de escrita em um bloco Compartilhado em outras caches faz com que a cache original adquira o barramento, envie um sinal de invalidação para bloquear todas as cópias, leia todo o bloco em que houve falha, modifique parte dele e mude seu estado para Modificado.
* **Directory Based:** o status do compartilhamento de um bloco da memória física é armazenado em um local centralizado chamado de diretório; toda mudança feita deve ser comunicada ao diretório, que cuida de comunicar a quem for necessário (quem tiver cópias dos dados) as alterações feitas; um único diretório mantém o estado de cada bloco na memória principal; as informações no diretório podem incluir quais caches possuem cópias do bloco, se ele é sujo, etc.
* Localizando cópia mais atualizada:
  + Caches Write-through: usa cópia da memória; é mais simples porém causa muitos acessos à memória e maior utilização do barramento.
  + Caches Write-back: deve localizar cópia mais recente nas caches; é mais complicado de implementar; reduz acessos à memória; a maioria dos multiprocessadores usam caches write-back.

9.7 Multiprocessadores no interior de um chip e Multithreading

* Uma alternativa aos multiprocessadores que compartilham uma interconexão é trazer os processadores para dentro do chip.
  + Nesses projetos, os processadores normalmente compartilham algumas das caches e a interface de memória externa.
  + Claramente, as latências associadas à comunicação chip a chip desaparecem quando tudo está no mesmo chip.
* Multithreading: permite que várias threads compartilhem as unidades funcionais de um único processador de um modo sobreposto.
  + Para permitir esse compartilhamento o processador precisa duplicar o estado independente de cada thread.
  + A memória em si pode ser compartilhada por meio de mecanismos de memória virtual, que já suportam multiprogramação.
  + O hardware precisa suportar a capacidade de mudar para uma thread diferente com relativa rapidez. Uma troca de threads deve ser muito mais eficiente do que uma troca de processos.
* Existem dois métodos principais de multithreading:
  + Multithreading fine-grained: comuta entre threads a cada instrução, resultando em execução intercalada de várias threads; essa intercalação é feita de forma circular, saltando quaisquer threads que estejam suspensas no momento.
    - O processador precisa ser capaz de trocar threads a cada ciclo de clock.
    - Vantagem: pode ocultar as perdas de vazão que surgem dos stalls curtos e longos, já que as instruções de outras threads podem ser executadas quando uma thread é suspensa.
    - Desvantagem: torna mais lenta a execução das threads individuais, já que uma thread que está pronta para ser executada sem stalls será atrasada por instruções de outras threads.
  + Multithreading coarse-grained: comuta threads apenas em stalls onerosos (longos), como as falhas de cache de nível 2.
    - Vantagem: reduz a necessidade de tornar a comutação de thread essencialmente rápida e tem muito menos chance de tornar mais lenta a execução de uma thread individual, visto que só serão despachadas instruções de outras threads quando uma thread encontrar um stall oneroso.
    - Desvantagem: é limitado em sua capacidade de sanar perdas de vazão, especialmente de stalls mais curtos.
  + Simultaneous multithreading (SMT): usa os recursos de um processador de despacho múltiplo escalonado dinamicamente para explorar paralelismo em nível de thread ao mesmo tempo em que explora o paralelismo em nível de instrução.
    - Vantagem: a capacidade do escalonamento dinâmica (renomeação de registradores) permite que instruções independentes de threads distintas possam ser executadas ao mesmo tempo, aumentando o desempenho; tolera todos os tamanhos de stalls; não sacrifica tanto o desempenho de uma única thread.
    - Desvantagem: ter que lidar com um banco de registradores maior, necessário para conter vários contextos.

