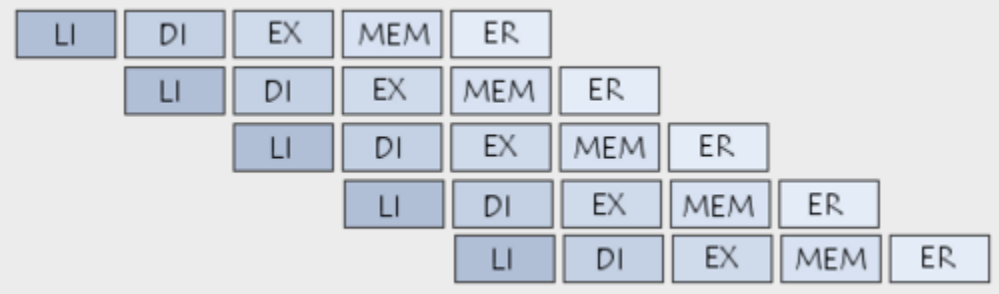
## Cap 14)

# Paralelismo a nível de instrução - Pipeline, Superescalar, Superpipeline




Houve um tempo em que tudo era sequencial, você tinha que terminar uma coisa para começar outra. Isso é uma forma bem ruim de utilizar o tempo, visto que há lacunas entre os estágios de uma atividade.

Atualmente o mundo está mais simultâneo, enquanto você ouve música, você lê, caminha, escreve, etc. Essa simultaneidade que está presente no nosso dia a dia influenciou os computadores modernos, e estes são capazes de executar mais de um programa de forma paralela.

O paralelismo em nível de instrução é, basicamente, o recurso que os computadores possuem para executar várias atividades (instruções) de forma paralela. Há algumas técnicas de implementação do paralelismo de instrução como: pipeline, arquitetura superescalar, superpipeline.

### Pipeline

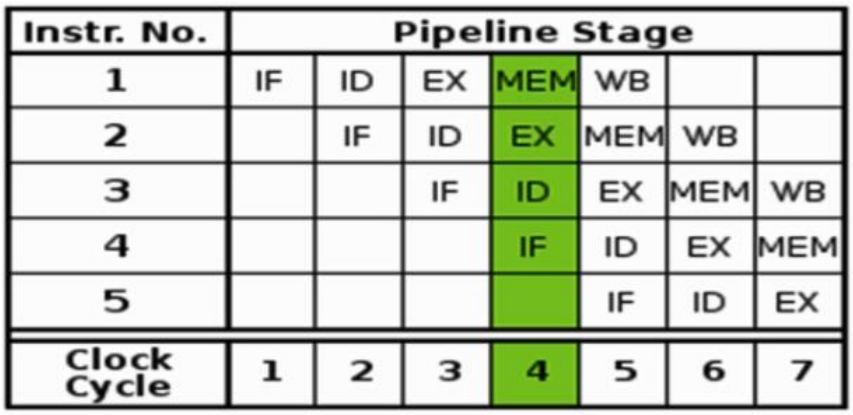
O pipeline é uma técnica muito usada pela arquitetura ***RISC*** que divide a execução da instrução em muitas partes, cada uma manipulada por uma parte dedicada do *hardware*, e todas elas podem ser executadas em paralelo.

Podemos considerar que um modelo universal de pipeline é um com 5 estágios, são eles:

- 1º Estágio: Busca
- 2º Estágio: Decodificação
- 3º Estágio: Execução
- 4º Estágio: Acesso à memória
- 5º Estágio: Gravação em registradores

O pipeline é comparado a uma linha de montagem industrial, onde não se espera que um produto tenha sido concluído para que inicie-se a fabricação do outro. O produto passa por vários estágios de

produção, e em vários processos de produção eles podem ser trabalhados de forma simultânea, como mostra a figura abaixo.



| Instr. No. | Pipeline Stage | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | IF | ID | EX | MEM | WB | | |
| 2 | | IF | ID | EX | MEM | WB | |
| 3 | | | IF | ID | EX | MEM | WB |
| 4 | | | | IF | ID | EX | MEM |
| 5 | | | | | IF | ID | EX |
| Clock Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Não é necessário que a instrução 1 tenha terminado (WB) para que a instrução 2 se inicie (IF). Pela imagem notamos que no 5º ciclo do *clock* todos as instruções estarão sendo executadas de forma paralela.

Como as atividades estão sendo executadas paralelamente é crucial que seus estágios tenham a MESMA duração de tempo.

**Arquitetura Superescalar**

Na arquitetura superescalar o processador executa múltiplas instruções em um ciclo de *clock*. Ele possui apenas um pipeline, mas há várias unidades funcionais. As unidades funcionais que demandam mais ciclos de *clock* são multiplicadas no intuito de forçar a realização daquele estágio em um tempo menor.

Imaginemos que tenha uma atividade com 3 estágios, onde a execução do estágio 1 leva 20 minutos, a execução do estágio 2 também leva 20 minutos, mas a execução do estágio 3 leva 40 minutos. Para que o estágio 3 seja realizado em menos tempo, basta que aumentemos a quantidade de trabalhadores que estão trabalhando naquele estágio. Se antes era 1 trabalhador apenas, ao colocarmos 2 trabalhadores no estágio 3, este estagio será executado em 20 minutos. A ideia do superescalar é a mesma, onde demanda mais tempo, deve haver mais empenho de *hardware* para o fazer em menos tempo.

Em **síntese**, a arquitetura superescalar é um reforço a um ou mais estágios, já que o reforço de *hardware* vai reduzir o tempo de execução de um estágio x.

**ATENÇÃO!** No superescalar **não** há vários processadores trabalhando em paralelo para diminuir o tempo de execução de um programa. O que existe é a duplicação, triplicação, quadruplicação, etc. de

algumas **UNIDADES FUNCIONAIS** no intuito de corrigir uma utilização de tempo demasiado especificamente naquele ponto.

## Superpipeline

Antes de falar do superpipeline é importante saber que um ciclo de *clock* possui dois momentos distintos, um é a subida do ciclo e outro é a descida do ciclo de execução.

O superpipeline otimiza o pipeline forçando a execução de 2 instruções por ciclo de *clock,* executando uma instrução na subida e outra na descida do ciclo.

Essa divisão do ciclo de *clock* é mais econômica do que duplicar o *hardware*.

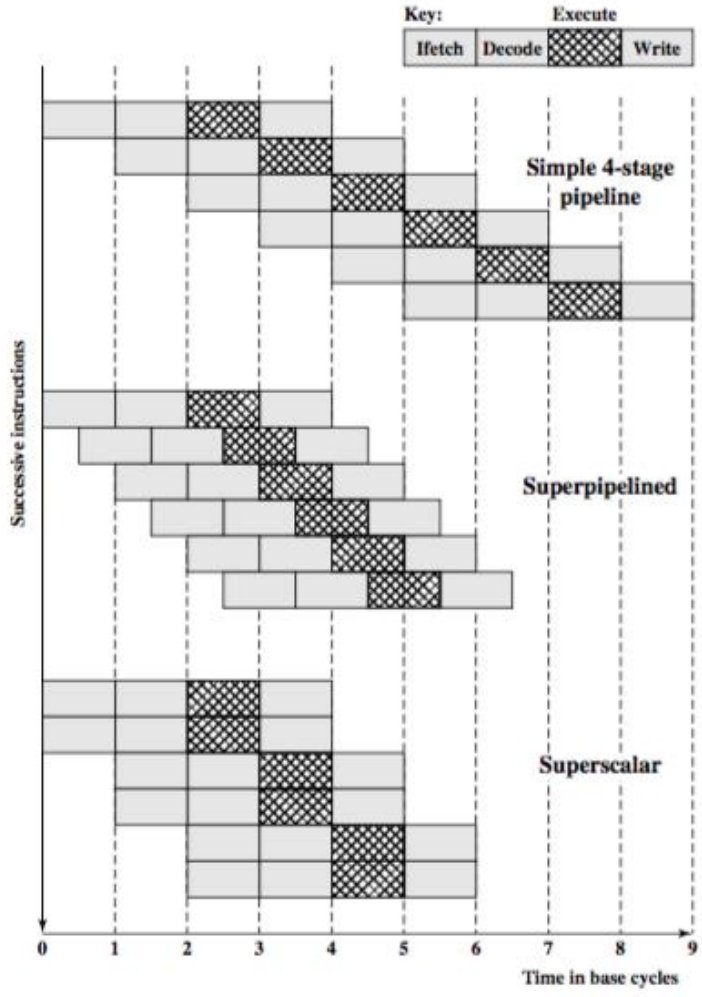



Figure 14.2 Comparison of Superscalar and Superpipeline Approaches

**Limitações**

1) **Dependência de dados**
2) **Dependência procedural**
3) **Conflitos de recursos**
4) Dependência de saída
5) Antidependência

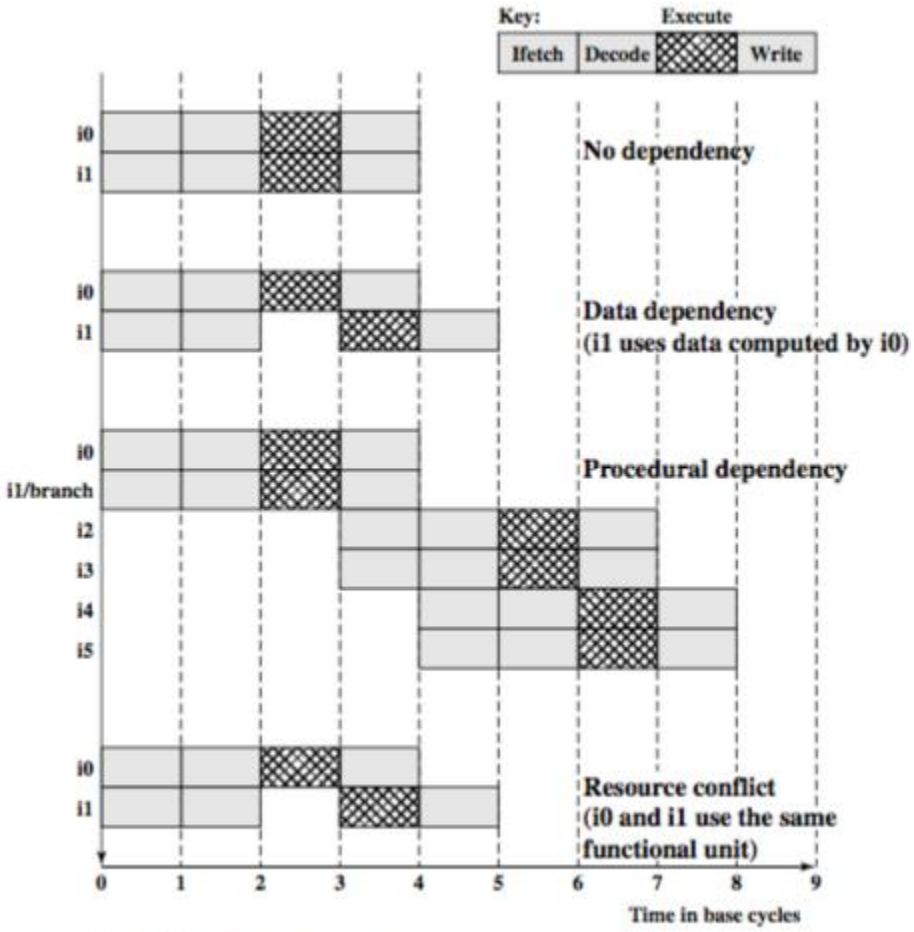



Figure 14.3 Effect of Dependencies

1) É aquele negócio de tipo, pra vc realizar uma instrução vc precisa q a anterior termine de ser executada e gere algum dado. Tipo, r1=r1 + r2 e r3 vai salvar o conteúdo de r1, vc não pode salvar o conteúdo em r3 até que r1 termine de fazer a operação.

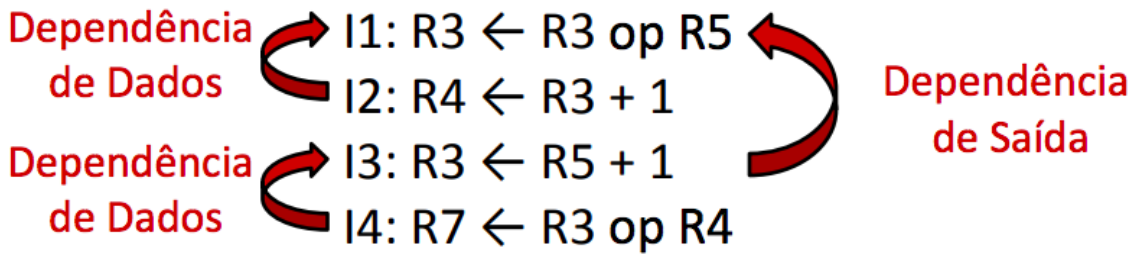
2) Uma instrução seguinte a uma instrução de desvio condicional não pode ser executada até que seja completada a execução da instrução de desvio.

3) Ocorre quando duas ou mais instruções competem, ao mesmo tempo, por um mesmo recurso • Exemplos de recursos:

– Caches
– Barramentos
– Registradores
– Unidades funcionais (O somador da ULA, por exemplo)

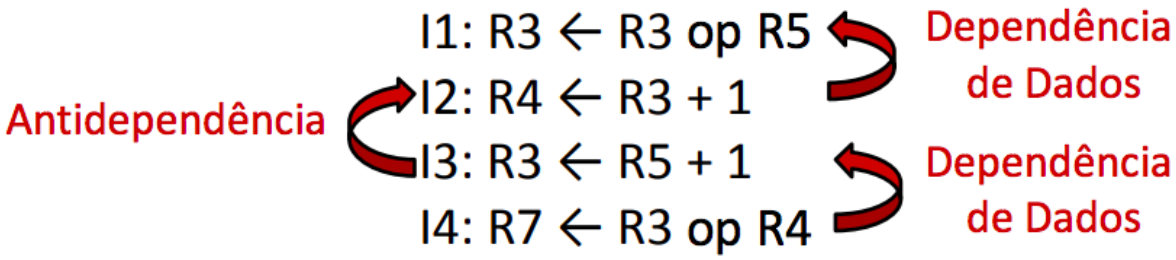
É semelhante, graficamente, à dependência de dados mas pode ser eliminada com a duplicação de recursos, entretanto.

4)




- Se I3 for completada antes de I1, o valor de R3 usado na execução de I4 estará errado
- Portanto, I3 deve ser completada depois de I1 para que os valores de saída produzidos estejam corretos

5)




- Ao invés de a primeira instrução produzir um valor usado pela segunda instrução, a segunda destrói um valor usado pela primeira

## Cap 17

**SMP)** O multiprocessamento simétrico ou SMP (Symmetric Multi-Processing) ocorre em um sistema computacional com vários processadores com memória compartilhada sob controle de um único sistema operacional. Em contraste o multiprocessamento assimétrico emprega sistemas diferentes.

O multiprocessamento simétrico oferece um aumento linear na capacidade de processamento a cada processador adicionado. Não há necessariamente um hardware que controle este recurso, cabe ao próprio sistema operacional suportá-lo.

- **Como funciona?** Os processadores trabalham sozinhos compartilhando os recursos de hardware, geralmente são processadores iguais, similares ou com capacidades parecidas. Todos tem os mesmos privilégios de acesso ao hardware, ao contrário do que acontece em sistemas multiprocessados assimétricos, onde um processador é encarregado de gerenciar e coordenar as tarefas e ações dos demais, o que pode proporcionar melhor controle sobre a sobrecarga ou a ociosidade dos processadores subordinados. Por tratar de grandes aspectos que todos os processadores de forma igualitária, **no multiprocessamento simétrico, qualquer processador pode assumir as tarefas realizadas por qualquer outro processador,** as tarefas são divididas e também podem ser executadas de modo concorrente em qualquer processador que esteja disponível. Os acessos dos processadores aos dispositivos de entrada e saída e a memória são feitos por um mecanismo de intercomunicação constituído por um barramento único.
  A memória principal da máquina é compartilhada por todos os processadores através de um único barramento que os interliga. Por todo acesso à memória principal ser realizado através de um único barramento, aqui temos um ponto de gargalo do sistema, pois o sistema fica limitado a passagem de apenas uma instrução de cada vez pelo barramento, abrindo uma lacuna de tempo entre uma instrução e outra. Memórias caches junto aos processadores diminuem o tempo de latência entre um acesso e outro à memória principal e ajudam também a diminuir o tráfego no barramento. Como estamos falando em mais de um processador, cada um com sua memória cache é imprescindível garantir que os processadores sempre acessem a cópia mais recente da memória cache, isso se chama *coerência de cache* geralmente implementada diretamente por hardware. Um dos métodos de coerência de cache mais conhecido é o *snooping*, quando um dado compartilhado nas caches dos processadores é alterado, todas as cópias das caches são consideradas inválidas e logo após atualizadas mantendo assim a integridade do dado.

  O sistema operacional é quem se encarrega de realizar a interação entre os processadores e as aplicações do sistema. Deixando a existência de múltiplos processadores transparente para os usuários, pois o próprio sincroniza os processos com os processadores.

- **Vantagens**

  sistemas de multiprocessamento simétrico são considerados mais poderosos em relação aos de multiprocessamento assimétricos, abaixo alguns descritivos de comparação entre eles.

- No multiprocessamento simétrico muitos processos podem ser executados ao mesmo tempo sem queda no desempenho, pois o sistema operacional delega as instruções a cada processador;
- Se um processador falhar o sistema não tranca pois qualquer outro processador pode assumir as tarefas daquele que falhou, já no assimétrico por exemplo, se o processador mestre falhar o sistema trava;
- No SMP o usuário pode melhorar o desempenho da máquina simplesmente adicionando um processador.

**Clusters)** Cluster (ou *clustering*) é, em poucas palavras, o nome dado a um sistema que relaciona dois ou mais computadores para que estes trabalhem de maneira conjunta no intuito de processar uma tarefa. Estas máquinas dividem entre si as atividades de processamento e executam este trabalho de maneira simultânea.

Cada computador que faz parte do cluster recebe o nome de *nó* (ou *node*). Teoricamente, não há limite máximo de nós, mas independentemente da quantidade de máquinas que o compõe, o cluster deve ser "transparente", ou seja, ser visto pelo usuário ou por outro sistema que necessita deste processamento como um único computador.

Os nós do cluster devem ser interconectados, preferencialmente, por uma tecnologia de rede conhecida, para fins de manutenção e controle de custos, como a Ethernet. É extremamente importante que o padrão adotado permita a inclusão ou a retirada de nós com o cluster em funcionamento, do contrário, o trabalho de remoção e substituição de um computador que apresenta problemas, por exemplo, faria a aplicação como um todo parar.

A computação em cluster se mostra muitas vezes como uma solução viável porque os nós podem até mesmo ser compostos por computadores simples, como PCs de desempenho mediano. Juntos, eles configuram um sistema de processamento com capacidade suficiente para dar conta de determinadas aplicações que, se fossem atendidas por supercomputadores ou servidores sofisticados, exigiriam investimentos muito maiores.

Não é necessário haver um conjunto de hardware exatamente igual em cada nó. Por outro lado, é importante que todas as máquinas utilizem o mesmo sistema operacional, de forma a garantir que o software que controla o cluster consiga gerenciar todos os computadores que o integram.

**Tipos:**

**Alto desempenho)** Clusters de alto desempenho são direcionados a aplicações bastante exigentes no que diz respeito ao processamento. Sistemas utilizados em pesquisas científicas, por exemplo,

podem se beneficiar deste tipo de cluster por necessitarem analisar uma grande variedade de dados rapidamente e realizar cálculos bastante complexos.

O foco deste tipo é o de permitir que o processamento direcionado à aplicação forneça resultados satisfatórios em tempo hábil

**Alta disponibilidade)** Nos **clusters de alta disponibilidade**, o foco está em sempre manter a aplicação em pleno funcionamento: não é aceitável que o sistema pare de funcionar, mas se isso acontecer, a paralização deve ser a menor possível, como é o caso de soluções de missão crítica que exigem disponibilidade de, pelo menos, 99,999% do tempo a cada ano, por exemplo.

Para atender a esta exigência, os clusters de alta disponibilidade podem contar com diversos recursos: ferramentas de monitoramento que identificam nós defeituosos ou falhas na conexão, replicação (redundância) de sistemas e computadores para substituição imediata de máquinas com problemas, uso de geradores para garantir o funcionamento em caso de queda de energia, entre outros.

**Balanceamento de Carga (Load Balancing))** Em clusters de balanceamento de carga, as tarefas de processamento são distribuídas o mais uniformemente possível entre os nós. O foco aqui é fazer com que cada computador receba e atenda a uma requisição e não, necessariamente, que divida uma tarefa com outras máquinas.

Não basta ao cluster de balanceamento de carga ter um mecanismo meramente capaz de distribuir as requisições - é necessário que este procedimento seja executado de forma a garantir um "equilíbrio" na aplicação. Para tanto, o mecanismo pode monitorar os nós constantemente para verificar, por exemplo, qual máquina está lidando com a menor quantidade de tarefas e direcionar uma nova requisição para esta.

O balanceamento de carga pode ser utilizado em vários tipos de aplicações, mas o seu uso é bastante comum na internet, já que soluções do tipo têm maior tolerância ao aumento instantâneo do número de requisições, justamente por causa do equilíbrio oriundo da distribuição de tarefas.

**OBS:** É válido frisar que uma solução de cluster não precisa se "prender" a apenas um tipo. Conforme a necessidade, pode-se combinar características de tipos diferentes no intuito de atender plenamente à aplicação.

Por exemplo, uma loja na internet pode utilizar um cluster de alta disponibilidade para garantir que suas vendas possam ser realizadas 24 horas por dia e, ao mesmo tempo, aplicar balanceamento de carga para suportar um expressivo aumento eventual no número de pedidos causados por uma promoção.

**Funcionamento)** é possível utilizar computadores "convencionais", como desktops para fins domésticos ou para uso em escritório. Assim, uma universidade ou uma empresa, por exemplo, pode utilizar máquinas que foram substituídas por modelos mais recentes para criar um cluster e, eventualmente, economizar com a aquisição de servidores.

Os nós podem ainda ser *não dedicados* ou *dedicados*. No primeiro caso, cada computador que faz parte do cluster não trabalha exclusivamente nele. No segundo, o nó é utilizado somente para este fim, fazendo com que dispositivos como teclados e monitores sejam dispensáveis - se, por algum motivo, for necessário acessar uma máquina em particular, pode-se fazê-lo via terminal, a partir do nó principal, por exemplo.

Outro elemento importante é o sistema operacional. É essencial que todas os computadores utilizem o mesmo sistema operacional.

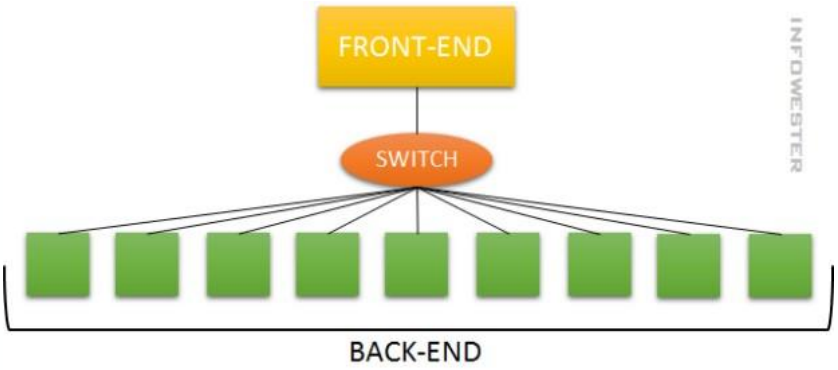
Esta homogeneidade é importante para diminuir a complexidade de configuração e manutenção do sistema, e garantir que os procedimentos rotineiros ao cluster, como monitorização, distribuição de tarefas e controle de recursos sejam executados de maneira uniforme. Para reforçar estes aspectos, pode-se até mesmo adotar sistemas operacionais preparados especialmente para clustering.

N*ó controlador* (ou *nó mestre*). O nome deixa claro: trata-se do já mencionado nó principal, que efetivamente controla o cluster a partir da distribuição de tarefas, do monitoramento e de procedimentos relacionados.

A comunicação entre os nós - que é onde está a delimitação do que constitui o cluster em si - é feita a partir de uma tecnologia de rede local. Os padrões Ethernet (Gigabit Ethernet, Fast Ethernet, etc) são bastante utilizados justamente por serem mais comuns e, portanto, melhor suportados e menos custosos. Mas há outras opções viáveis, entre elas, o Myrinet e o InfiniBand, ambos com características bastante apropriadas para clustering.

**Cluster Beowulf)** Um cluster Beowulf se define, basicamente, pela ênfase nas seguintes características:

- entre os nós, deve haver pelo menos um que atue como mestre para exercer o controle dos demais. As máquinas mestres são chamadas de *front-end*; as demais, de *back-end*. Há a possibilidade de existir mais de um nó no front-end para que cada um realize tarefas específicas, como monitoramento, por exemplo;




- a comunicação entre os nós pode ser feita por redes do tipo Ethernet, mais comuns e mais baratas, como você já sabe;

- não é necessário o uso de hardware exigente, nem específico. A ideia é a de se aproveitar componentes que possam ser encontrados facilmente. Até mesmo PCs considerados obsoletos podem ser utilizá-los;

- o sistema operacional deve ser de código aberto, razão pela qual o Linux e outras variações do Unix são bastante utilizados em cluster Beowulf. O MOSIX, a ser abordado no próximo tópico, é uma opção bastante usada para este fim;

- os nós devem se dedicar exclusivamente ao cluster;

- deve-se fazer uso de uma biblioteca de comunicação apropriada, como a *PVM* (*Parallel Virtual Machine*) ou a *MPI* (*Message Passing Interface*). Ambas são direcionadas à troca de mensagens entre os nós, mas o MPI pode ser considerado mais avançado que o PVM, uma vez que consegue trabalhar com comunicação para todos os computadores ou para apenas um determinado grupo.

## Vantagens e desvantagens dos clusters

Neste ponto do texto, você certamente já compreendeu as vantagens de um cluster. Eis as principais:

- pode-se obter resultados tão bons quanto ou até superiores que um servidor sofisticado a partir de máquinas mais simples e mais baratas (ótima relação custo-benefício);

- não é necessário depender de um único fornecedor ou prestador de serviço para reposição de componentes;

- a configuração de um cluster não costuma ser trivial, mas fazer um supercomputador funcionar poder ser muito mais trabalhoso e exigir pessoal especializado;

- é possível aumentar a capacidade de um cluster com a adição de nós ou remover máquinas para reparos sem interromper a aplicação;

- há opções de softwares para cluster disponíveis livremente, o que facilita o uso de uma solução do tipo em universidades, por exemplo;

- relativa facilidade de customização para o perfeito atendimento da aplicação;

- um cluster pode ser implementado tanto para uma aplicação sofisticada quanto para um sistema doméstico criado para fins de estudos, por exemplo.

Mas, apesar destes benefícios, os clusters não são a solução perfeita para todo e qualquer problema computacional. Há aplicações onde o uso de outra tecnologia se mostra mais adequado. Entre as razões para isso estão:

- a facilidade de expansão do cluster pode ser uma "faca de dois gumes": a quantidade de máquinas pode aumentar tanto que a manutenção se torna mais trabalhosa, o espaço físico pode ficar impróprio, etc;

- a tecnologia de comunicação utilizada pode não oferecer a velocidade de transferência de dados ou o tempo de resposta necessário, dependendo da aplicação;

- um cluster tem como base uma rede local, logo, não se pode acrescentar máquinas que estejam muito distantes geograficamente.

Por estes aspectos, fica evidente que as necessidades e os requisitos de uma aplicação devem ser bem avaliados para que se possa decidir entre a implementação de um cluster ou outra tecnologia. Se o clustering for a opção escolhida, deve-se seguir com a avaliação, desta vez para se decidir sobre as soluções e recursos disponíveis.

## Coerência de cache e protocolo MESI)

### 1) Coerência de cache

1. INTRODUÇÃO - Manter uma cache coerente é garantir que os dados armazenados estejam sincronizados com as atualizações realizadas ao longo do processamento das aplicações pelo processador. Dados podem ser alterados, removidos ou adicionados durante um acesso à cache. Em

sistemas multiprocessados, que atendem maiores demandas de processamento, cada processador pode ter sua própria cache, chamada cache privada. O uso de cache privada em sistemas multiprocessados pode gerar problemas relacionados à coerência das caches, pois podem existir os mesmos dados copiados em diferentes caches e estes dados podem ser atualizados de forma diferente pelos processadores.

2. O PROBLEMA DE COERÊNCIA DE CACHE - O mecanismo de cache representa a idéia da memória virtual em que os dados armazenados ficam mais próximos do processador do que da memória principal. O objetivo principal da cache é prover os dados ao processador de uma forma mais rápida do que a memória principal. Multiprocessadores usam várias caches, uma para cada processador. Em um design de multiprocessadores com várias caches, múltiplas cópias dos mesmos dados podem existir em caches diferentes. Essa existência de cópias do mesmo dado é que gera o problema de coerência de cache. Um sistema será coerente se todas as leituras por qualquer processador retornam o valor produzido pela última operação de escrita, sem importar qual processador realizou a escrita.

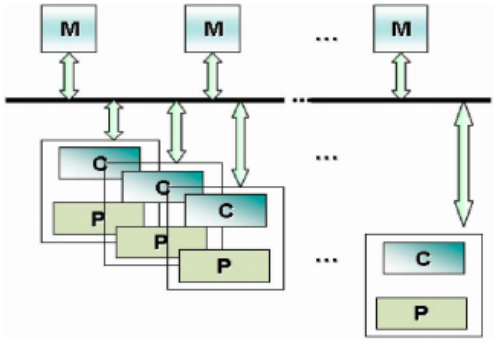



**Figura 1. Cache em multiprocessadores**

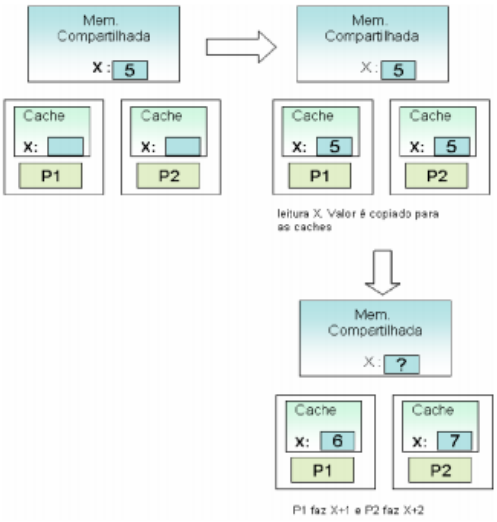



**Figura 2. O problema de coerência de cache**

3. COMO RESOLVER O PROBLEMA DE COERÊNCIA DE CACHE - Existem vários esquemas criados para resolver o problema de coerência da cache, alguns deles baseados em software e outros em hardware. Esquemas mais simples se baseiam na limitação do tipo de dado que pode ir para a cache, permitindo que apenas dados para leitura ou mesmo dados não-compartilhados (privados ao processador) possam ser armazenados na cache. No caso dos esquemas baseados em software, delega-se ao compilador ou ao sistema operacional a responsabilidade de garantir a coerência. A vantagem dos esquemas baseados em software é evitar que um outro hardware seja necessário para realizar o tratamento do problema de coerência. No caso dos esquemas baseados em hardware, as inconsistências são identificadas pelo hardware durante a execução, deixando assim que o tratamento seja transparente ao compilador ou ao sistema operacional. **A coerência por hardware é a mais utilizada**. Os esquemas de coerência são denominados protocolos de coerência de cache

4.1 Esquemas de coerência baseados em software - Esquemas de coerência de cache baseados em software são mais baratos e, na maioria das vezes, menos complexos do que os baseados em hardware. Porém podem demandar mais suporte por parte do programador. Alguns esquemas são implementados como parte do compilador e, portanto, rodam em tempo de compilação. Outros são parte do sistema operacional e, portanto, rodam em tempo de execução. Também existem esquemas que operam em ambos, compilador e sistema operacional. Os esquemas que rodam em tempo de compilação são classificados como estáticos e os que rodam em tempo de execução são classificados como dinâmicos.

**One-Time Identifier** - No esquema One-time Identifier, cada página compartilhada possui um identificador único associado à página, chamado onetime identifier. Um campo de identificador é adicionado para cada linha da cache e para cada entrada da TLB. A cada atualização da TLB um novo valor de identificador é colocado no campo de identificador. Quando uma linha é acessada pela primeira vez, os dados são copiados da memória principal e o valor do identificador da entrada correspondente à TLB é copiado para o campo de identificador da entrada correspondente à cache no registro de identificadores. Todos os acessos posteriores irão comparar o valor do identificador one-time da entrada de identificadores da TLB com o valor do identificador da entrada da cache no registro. Caso os valores sejam iguais, tem-se um hit. Caso contrário, tem-se um miss.

**Version Control** - Este esquema utiliza versionamento como forma de garantir a coerência. Cada nova escrita de uma dado compartilhado gera uma nova versão do conteúdo. Os processadores incrementam o número da versão corrente a cada escrita que ocorra. Na cache, cada linha também possui um número de versão que é atribuído a cada alocação da cache. Para todo acesso ao dado compartilhado, os números de versão na cache e o corrente do processador são comparados. Caso o número da versão na cache seja menor, ocorre um miss, caso contrário, ocorre um hit.

4.2 Protocolos de coerência baseados em hardware - Embora sejam mais complexos, os protocolos de coerência baseados em hardware são utilizados principalmente em sistemas multiprocessadores comerciais. Tais protocolos lidam com a coerência em tempo de execução, sendo assim considerados dinâmicos. Os protocolos podem ser classificados em Snoopy e Directory.

- Protocolos Snoopy - Estes protocolos fazem uso da distribuição no tratamento do problema e coerência de cache. Seu procedimento é baseado em ações dos controladores de caches locais e suas informações sobre o estado dos dados na cache. Utiliza-se broadcast (transmissão) para comunicar às demais caches as ações executadas sobre os dados compartilhados. Como os protocolos Snoopy são baseados em broadcast, são mais indicados a multiprocessadores baseados em barramento, pois o barramento irá favorecer o broadcast. Um problema detectado nos protocolos Snoopy faz com que eles sejam substituídos por outros protocolos, tais como os protocolos Directory: barramentos não são escaláveis quando se tem um número muito grande (dezenas) de processadores. Protocolos Snoopy podem ser do tipo Single bus e Multiple bus. Os Single bus podem usar políticas de Write-invalidate ou Writeupdate. A seguir descreveremos um exemplo relacionado a cada tipo de protocolo Snoopy:

Single bus – Write-invalidate - Synapse

A política de Write-invalidate permite apenas um escritor e vários leitores. Write-update atualiza todas as cópias do dado compartilhado antes de realizar a escrita. Como um protocolo que utiliza a política Write-invalidate, Synapse faz uso de três estados: Invalid, Valid e Dirty. Qualquer cache que possua uma cópia em estado Dirty é a "dona" do bloco. Se não houver nenhum Dirty, a memória é a "dona". Para situações de miss e hit, o Synapse possui o seguinte comportamento:

·Read Hit: não há problemas de coerência

·Read Miss: o bloco é copiado para a cache e a CPU muda o estado da linha na cache para Valid.

·Write Hit: o barramento notifica todos os processadores que a linha é Invalid. A linha na cache é marcada como Dirty e todos os outros processadores marcam suas linhas correspondentes como Invalid.

·Write Miss: idêntico ao caso anterior

Multiple bus – Wisconsin Multicube

São xˆk processadores com ligações N para N com os barramento: cada barramento conecta x processadores e cada processador está conectado com k barramentos, onde k é a dimensão do multicube. O multicube consiste de cache de dois níveis: cache do processador e snooping cache. A consistência entre os dois níveis é mantida através da estratégia de write-through. Uma linha estará sempre em um dos estados globais: unmodified ou modified. Existem também alguns modos locais: local, modified e invalid. O protocolo possui quatro tipo de transações: leitura, leitura-alteração, allocate e write-back.

Protocolos Directory

É utilizado um diretório, geralmente na memória principal, que armazena o estado global do conteúdo das várias caches. A cada requisição da cache local, o controlador central faz uma análise do diretório buscando qual a cache que possui a cópia do dado e libera a tranferência dos dados da memória principal para as caches. Este controlador realiza a atualização do estado da informação. O controlador central recebe as informações sobre tarefas que possam afetar o estado dos dados. Protocolos de coerência baseados em hardware Um problema detectado nos protocolos Directory é a necessidade por espaço que tende a aumentar à medida que o número de processadores aumenta.

Directory – Full Map

Seu funcionamento se dá através da concatenação de um vetor de bits para cada bloco de memória. Esse vetor de bits possui um bit por cache. Cada bit indica a presença ou ausência do bloco na cache correspondente. Este método possui eficiência com relação ao tempo.

Directory – Limited Directory

Esse protocolo limita o crescimento do diretório em um fator constante de crescimento. Para isso, restringe-se o número de cópias simultâneas de bloco dados para a cache. Quando uma cache requer a cópia de um dado para escrita, o módulo de memória invalida as demais cópias existentes nas outras caches.

**Protocolo MESI)**

O **protocolo MESI** é um protocolo de coerência de cache e coerência de memória, que foi introduzido pela Intel no processador Pentium para "*dar suporte a uma escrita de volta na cache mais eficiente, além da escrita-direta, anteriormente usada no processador Intel 486*".

Cada linha da cache é marcada com um dos quatro estados seguintes (codificados em dois bits adicionais):

- **M** - linha Modificada: a linha da cache está presente apenas na cache atual, e é diferente da memória principal. A cache é solicitada a escrever os dados de volta na memória principal antes de permitir qualquer outra leitura do estado da memória principal, já não válido.
- **E** - linha Exclusiva: a linha de cache está presente apenas na cache atual, mas está *limpa*; ela é igual à da memória principal.
- **S** - Shared line, linha compartilhada: Indica que esta linha de cache pode ser armazenada em outras caches da máquina.
- **I** - linha Inválida: indica que esta linha de cache é inválida.

Uma cache pode satisfazer uma leitura de qualquer estado, exceto o inválido. Uma linha inválida deve ser buscada (nos estados compartilhada e exclusiva) para satisfazer uma leitura.

Uma escrita só pode ser executada se a linha de cache estiver no estado modificado ou exclusivo. Se ela estiver no estado compartilhado, todas as outras cópias em cache devem ser invalidadas primeiro. Essa invalidação é geralmente feita através de uma operação de transmissão conhecida como ***Read For Ownership (RFO)*.**

Uma cache pode rejeitar uma linha não-modificada a qualquer momento, modificando-a para o estado inválido. Uma linha modificada deve ser, antes, escrita de volta.

Uma cache que guarda uma linha no estado modificado deve *interceptar* (intercept) todas as tentativas de leitura (de todas as outras CPUs no sistema) o local correspondente da memória principal e inserir os dados que ela armazena. Isso é geralmente feito forçando a leitura ao *back off* (por exemplo, para abortar a transmissão entre memória e barramento), escrevendo então os dados na memória principal, e mudando a linha de cache para o estado compartilhado.

Uma cache que tem uma linha no estado compartilhado deve interceptar todas as transmissões inválidas de outras CPU'S, e rejeitar a linha (alterando-a para o estado inválido) correspondente.

Uma cache que tem uma linha no estado exclusivo deve também rejeitar todas as transações de leitura de todas as outras CPUs, e mudar a linha para o estado compartilhado.

Os estados modificado e exclusivo são sempre precisos: por exemplo, eles encontram uma situação de posse da verdadeira linha de cache no sistema. O estado compartilhado pode ser impreciso: se outra CPU rejeitar uma linha compartilhada, e esta CPU se tornar a única proprietária desta linha de cache, a linha não será promovida ao estado exclusivo (porque não é prático, no caso de interceptação de transmissão, transmitir todas as substituições das linhas de cache a todas as CPUs).

Neste sentido, o estado exclusivo é uma otimização oportuna: se a CPU quiser modificar uma linha de cache que está no estado compartilhado, uma transmissão de barramento é necessária para invalidar todas as outras cópias na cache. O estado exclusivo permite modificar uma linha de cache sem nenhuma transmissão por barramento.

## Sistemas com múltiplos processadores)

### Introdução

Sistemas com múltiplos processadores possuem duas ou mais UCP interligadas e que funcionam em conjunto na execução de tarefas independentes ou no processamento simultâneo de uma mesma tarefa.

Inicialmente os sistemas operacionais executavam um programa de cada vez mas com o surgimento dos sistemas operacionais com suporte a múltiplos processadores o processamento paralelo foi expandido.

### A motivação para o uso de múltiplos processadores

Existem alguns fatores que motivaram o desenvolvimento de sistemas com múltiplos processadores:

- O elevado custo para o desenvolvimento de processadores mais rápidos levou ao desenvolvimento de sistemas com múltiplos processadores ao invés de sistemas com um único processador de alto desempenho.
- Aplicações que requerem alto desempenho ou poder computacional. As aplicações atuais cada vez mais demandam poder de processamento e desempenho que seriam dificilmente alcançados por uma arquitetura com um único processador com alto desempenho.

**Os diversos benefícios dos sistemas com multi processadores**

Inicialmente o desempenho foi o principal fator para o desenvolvimento dos sistemas multi processados porém posteriormente as empresas perceberam outros benefícios, antes não atingidos, que os sistemas multi processados oferecem.

- **Aumento da Confiabilidade** - Com mais de um processador, caso haja falha de hardware em um processador os demais processadores mantem os sistemas em funcionamento.
- **Escalabilidade** - Se antes era necessário trocar o sistema computacional por um outro sistema computacional com mais poder de processamento, e isso custava muito caro, com os sistemas multi processados basta adicionar novos processadores conforme a demanda.
- **Alta disponibilidade** - Pelo mesmo motivo do aumento da confiabilidade, os sistemas multi processados oferecem alta disponibilidade dos serviços pois em caso de falha de um ou mais processadores, os outros processadores garantem a disponibilidade do serviço.
- **Balanceamento de carga** - Todo o processamento pesado pode ser distribuído pelos vários processadores disponíveis no sistema, dividindo a carga do processamento e conseguindo melhores resultados em desempenho.

Atualmente os servidores de banco de dados, servidores de arquivos, servidores web usam sistemas com múltiplos processadores justamente para atender a estes requisitos de infraestrutura.

**Tipos de Sistemas Computacionais**

Os sistemas são classificados conforme o grau de paralelismo no processamento de instruções. Um modelo proposto define quatro tipos de sistemas computacionais:

**SISD ( Single Instruction Single Data )** - Sistemas que suportam uma única sequencia de instruções e apenas uma sequencia de dados. Os sistemas com um único processador estão nessa categoria. Estes sistemas simulam paralelismo com a utilização de uma técnica conhecida como pipeline.

**SIMD ( Single Instruction Multiple Data )** - Sistemas que com uma única sequencia de instruções tratam multiplas sequencias de dados.

**MISD ( Multiple Instruction Single Data ) -** Sistemas que permitem a execução de múltiplas sequencias de instruções em uma única sequencia de dados. Não existe até o momento nenhum sistema computacional criado com esta arquitetura.

**MIMD ( Multiple Instruction Multiple Data )** - Sistemas que permitem a execução de múltiplas sequencias de instruções sobre múltiplas sequências de dados. Esta categoria engloba os sistemas com multiplos processadores. Esse tipo de máquina possui uma unidade de controle para cada processador que executam instruções independentemente um dos outros e podem trocar informações entre si por meio de uma rede de comunicação. *Várias unidades de controle buscam instruções na memória e as repassam para o processador a elas associados. Os processadores comunicam-se entre si por meio de uma memória compartilhada ou por meio de uma rede de comunicação.*

**Sistemas Fortemente acoplados e fracamente acoplados**

Os sistemas com múltiplos processadores usam a arquitetura MIMD e podem ser classificados em **fortemente acoplados ou fracamente acoplados**. *Nos sistemas fortemente acoplados* os processadores compartilham a memória principal e são controlados por apenas um único sistema operacional.

*Nos sistemas fracamente acoplados* temos dois ou mais sistemas computacionais independentes conectados por uma rede de comunicação, tendo cada sistemas seus processadores, memória principal, dispositivos E/S e sistema operacional.

A grande diferença entre estes dois sistemas é que em sistemas fortemente acoplados existe apenas um espaço de endereçamento compartilhado por todos os processadores, chamado de memória compartilhada. Nos sistemas fracamente acoplados cada sistema tem sua memória principal, seu espaço de endereçamento individual e a comunicação entre os sistemas é feita através de troca de mensagens.

**Sistemas com Multiprocessadores Simétricos ( SMP )**

São sistemas fortemente acoplados que compartilham o mesmo espaço de endereçamento e são gerenciados por um único sistema operacional. São também conhecidos como sistemas SMP.

O tempo de acesso à memória principal pelos vários processadores é uniforme. Não importa a localização física do processador. Esta arquitetura é chamada por UMA ( Uniform Memory Access ).

Sistemas SMP implementam a simetria dos processadores, onde todos os processadores realizam as mesmas funções. Poucas funções ficam a cargo de um processador central, como por exemplo, a inicialização do sistema. Como todos os processadores executam as mesmas funções existe um melhor balanceamento da carga e das operações de E/S.

**Clusters**

São sistemas fracamente acoplados formados por vários sistemas computacionais chamados nós que são conectados por uma rede que precisa ser de alto desempenho. Cada nó da rede é

denominado membro do cluster e possui seus próprios recursos como processadores, memória, dispositivos E/S e sistema operacional.

Geralmente os membros do cluster são do mesmo fabricante por questões de incompatibilidade de sistemas operacionais.

Como cada nó do cluster possui sua memória principal a comunicação entre os nós é feita através de troca de mensagens.

Geralmente a rede de conexão é dedicada aos nós do cluster e o acesso aos serviços é feito através de uma outra rede.

Este sistema oferece alta disponibilidade e tolerância a falhas. O usuário que acessa um cluster não tem conhecimento dos nós que compõem o cluster. Do ponto de vista do usuário é como se ele tivesse acessando um único sistema fortemente acoplado.

**Sistemas Operacionais de Rede ( SOR )**

Os sistemas operacionais de rede são o melhor exemplo de um ambiente fracamente acoplado. Cada sistema, nó da rede, possui seus próprios recursos de hardware e são independentes dos demais nós da rede, sendo interconectados por uma rede formando uma rede de computadores.

Os SOR são usados tanto em redes locais como em redes Ethernet e redes distribuídas ( WAN ), sendo a comunicação feita através de uma interface de rede que possibilita o acesso aos demais componentes da rede.

Não existe limite quanto ao número de nós que podem fazer parte da rede de computadores. Como cada nó possui seu próprio sistema operacional e os sistemas operacionais podem ser diferentes existe um protocolo que garante a comunicação entre os nós. No caso o protocolo TCP/IP.

**Os SORs e o Modelo Cliente Servidor**

A grande maioria dos SORs e seus protocolos de rede implementa o modelo cliente-servidor. Neste modelo existe um ou mais servidores que oferecem serviços como servidor de impressão, servidor de arquivos, servidor de correio eletrônico, servidor de banco de dados. Os servidores oferecem os serviços aos clientes da rede. O Windows Server e o Novell Netware são exemplos de SORs voltados para oferecer este tipo de serviço.

**Sistemas Distribuídos**

Um sistema distribuído é um conjunto de sistemas autônomos que são interconectados por uma rede de comunicação e que funciona como se fosse um sistema fortemente acoplado mas de fato não é. Cada componente de um sistema distribuído possui seus próprios recursos e sistema operacional. Os sistemas operacionais dos componentes que compõem um sistema distribuído podem ser heterogêneos.

O que diferencia um sistema distribuído dos demais sistema fracamente acoplados é que existe um relacionamento mais forte entre seus componentes. Do ponto de vista físico, o hardware é

independente, fracamente acoplado, porém do ponto de vista lógico, existe um relacionamento forte entre o software existente nos sistemas.

Os componentes de um sistema distribuidos podem ser conectados por uma rede local ou através de uma rede distribuída. A escalabilidade de um sistema distribuído é, ao menos à princípio, ilimitada pois basta acrescentar novos componentes à rede em função da necessidade. Para o Usuário e suas aplicações é como se houvesse um único sistema fortemente acoplado ao invés de uma rede conectando sistemas heterogêneos e independentes. Este conceito é chamado de imagem única do sistema.

**Transparência**

Em um sistema distribuído quando um usuário se conecta ao sistema, independente da localização física dos componentes, o usuário terá acesso a todos os arquivos, diretórios e demais recursos de forma transparente. Ao executar uma aplicação o usuário nem mesmo saberá em quais componentes a sua aplicação está rodando. Caso ocorra um erro em um desses componentes o usuário não terá conhecimento ficando sob responsabilidade do sistema operacional a resolução de todos os problemas.

Este conceito é chamado transparência. A partir deste conceito, o conjunto de componentes parece ser um sistema único, criando a imagem única do sistema.

A transparência aqui é feita a partir de vários aspectos de um sistema distribuído:

- Transparência de acesso
- Transparência de localização
- Transparência de concorrência
- Transparência de desempenho
- Transparência de escalabilidade
- Transparência a falhas
- Transparência de migração

**A tolerância a falhas**

Para que um sistema distribuído ofereça transparência em todos os aspectos acima mencionados é necessário que haja tolerância a falhas. Falhas tanto de hardware quanto de software. Neste caso o sistema tem que garantir que em caso de problema em algum componente as aplicações continuem sendo processadas sem qualquer interrupção do usuário, de forma totalmente transparente.

A tolerância a falhas de hardware é garantida através de redundância de componentes como fontes duplicadas, vários processadores, técnicas de RAID, memória com detecção e correção de erros, etc. Ainda temos redundância dos meios de conexão, placas de rede, linhas de comunicação e dispositivos de rede.

A tolerância a falhas de software é mais difícil de implementar. Caso haja uma falha no sistema operacional, a aplicação deve continuar como se não houve havido falha, sem que o usuário perceba que houve a falha. Neste caso, como as aplicações estão distribuídas em vários sistemas, caso ocorra

algum problema com um dos componentes é possível que um deles assuma de forma transparente o papel do sistema defeituoso.

Por exemplo: Ao acessar um sistema de suporte houve falha na conexão com o banco de dados. O servidor de banco de dados, o serviço de banco de dados estava fora do ar. Neste caso o sistema de suporte se conecta a outro servidor de banco de dados, servidor de contingencia e o sistema de suporte prossegue como se não tivesse havido falha na conexão ao banco de dados.

**NUMA)**

- *Acesso uniforme à memória (UMA):* O tempo de acesso à memória por um processador é o mesmo para todas as regiões da memória. Os tempos de acesso experimentados por diferentes processadores também são iguais.

- *Acesso não-uniforme à memória (NUMA):* O tempo de acesso à memória por um processador difere conforme a região de memória que está sendo usada. Isso vale para todos os processadores; entretanto, as regiões de memória para as quais o acesso é mais lento ou mais rápido são diferentes para diferentes processadores.

NUMA (Non-Uniform Memory Access) Nessas arquiteturas a memória é dividida em tantos blocos quanto forem os processadores do sistema, e cada bloco de memória é conectado via barramento a um processador com memória local. O acesso aos dados que estão na memória local é muito mais rápido que o acesso aos dados em blocos de memória remotos.

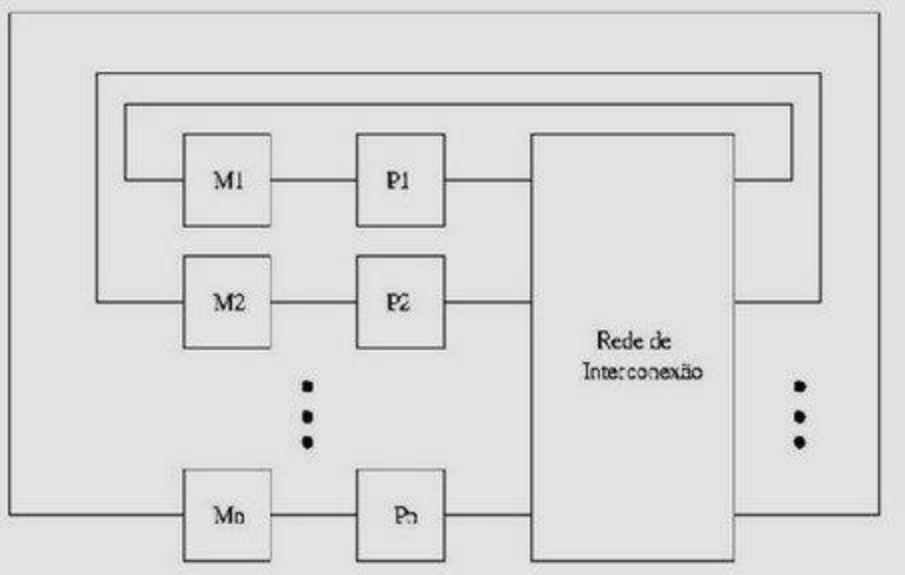
No multiprocesador NUMA o tempo de acesso à memória compartilhada varia de acordo com a localização da palavra de memória em relação ao processador.

- O tempo de acesso à posições de memória não é uniforme para todas as posições e todos os processadores;
- Normalmente, a memória compartilhada, ou parte dela, é destruída entre os processadores como memória local;
- Três padrões de acesso à memória são observados:

    - Acesso à memória local - mais rápido;
    - Acesso à memória global - intermediário;
    - Acesso à memória remota - mais lento.

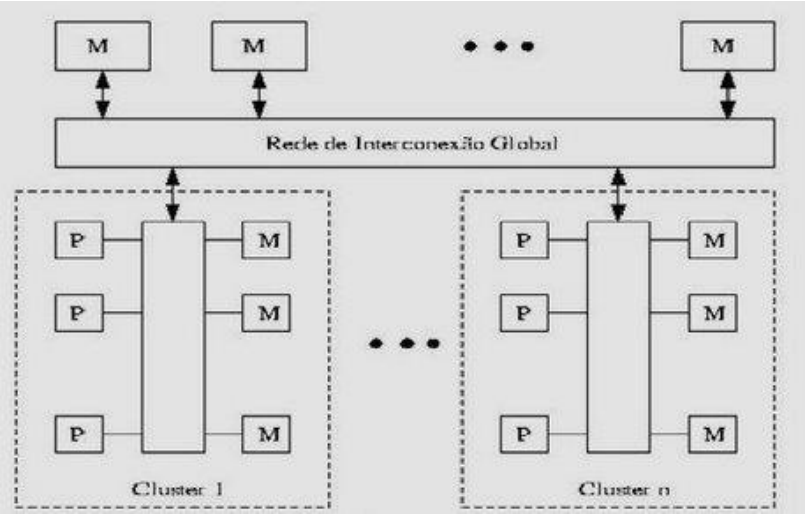
Dois modelos de NUMA serão mostrados nas figuras abaixo.

Na primeira figura a memória compartilhada está fisicamente distribuída por todos os processadores, neste caso denominadas de memórias locais e estão interligadas por uma rede de interconexão. O acesso por um processador à sua memória local é mais rápido do que o acesso à uma memória local de um outro processador usando a rede de interconexão.




- Máquina-MIMD NUMA: Cada processador possui sua memória local. Acessso à memória remota é permitido e realizado por meio de uma rede de interconexão.

Já na arquitetura apresentada a seguir, os processadores e memórias são organizados em hierarquia, e divididos em vários clusters. Cada cluster em si é uma UMA ou NUMA e são interligados a uma memória compartilhada global. Todos os processadores pertencentes a um mesmo cluster fazem acesso uniforme às memórias compartilhadas no interior do cluster. Todos os clusters têm o mesmo tempo de acesso à memória compartilhada global. Entretanto, o acesso à uma memória no mesmo cluster é menor do que o tempo de acesso à memória global.




- Máquina-MIMD NUMA: Organização hierarquizada em cluster

**Vantagens e desvantagens -** A principal vantagem de um sistema CC-NUMA é que ele pode disponibilizar desempenho efetivo em níveis de paralelismo mais altos que o fornecido por sistemas SMP, sem requerer mudanças substanciais no software. Com múltiplos nós NUMA, o tráfego no barramento em qualquer nó individual é limitado à demanda que esse barramento é capaz de tratar. Porém, se houver grande número de acessos a posições de memória localizadas em nós remotos, o desempenho cairá substancialmente.

Existem também algumas desvantagens na abordagem CC-NUMA. Uma delas é que um sistema CC-NUMA não se apresenta de forma transparente como um SMP; são requeridas modificações de software para migrar sistemas operacionais e aplicações de um sistema SMP para um sistema CC-NUMA. Tais modificações incluem alocação de páginas, alocação de processos e balanceamento de carga pelo sistema operacional. Uma segunda preocupação é a disponibilidade. Essa é uma questão bastante complexa e depende de exata implementação do sistema CC-NUMA.

## Multithreading)

Na arquitetura do computador , o multithreading é a capacidade de uma unidade de processamento central (CPU) ou um único núcleo em um processador multi-core para executar vários processos ou threads simultaneamente. Esta abordagem difere do multiprocessamento , assim como o multithreading dos processos e threads compartilha os recursos de um único ou múltiplos núcleos: as unidades de computação, a CPU esconde e o buffer de lookaside de tradução (TLB).

Onde os sistemas de multiprocessamento incluem várias unidades de processamento completas, o multithreading visa aumentar a utilização de um único núcleo usando o nível de linha e o paralelismo de nível de instrução. Como as duas técnicas são complementares, às vezes são combinadas em sistemas com várias CPUs multithreading e em CPUs com múltiplos núcleos de multithreading. Duas técnicas principais para computação de throughput são *multithreading* e *multiprocessamento* .

**Vantagens**

Se um segmento obtém muitas falhas de cache, os outros tópicos podem continuar aproveitando os recursos de computação não utilizados, o que pode levar a uma execução geral mais rápida, pois esses recursos estariam ociosos se apenas um único segmento fosse executado. Além disso, se um segmento não pode usar todos os recursos de computação da CPU (porque as instruções dependem do resultado do outro), executar outro segmento pode impedir que esses recursos se tornem inativos.

Se vários segmentos funcionam no mesmo conjunto de dados, eles podem realmente compartilhar seu cache, levando a um melhor uso ou sincronização de cache em seus valores.

**Desvantagens**

Múltiplos tópicos podem interferir uns com os outros ao compartilhar recursos de hardware, como caches ou buffers lookaside de tradução (TLBs). Como resultado, os tempos de execução de um único segmento não são melhorados, mas podem ser degradados, mesmo quando apenas um segmento está sendo executado, devido a freqüências mais baixas ou estágios de pipeline adicionais que são necessários para acomodar o hardware de troca de threads.

Do ponto de vista do software, o suporte de hardware para multithreading é mais visível para o software, exigindo mais mudanças nos programas de aplicativos e nos sistemas operacionais do que no multiprocessamento. As técnicas de hardware usadas para suportar multithreading freqüentemente são paralelas às técnicas de software usadas para multitarefa em computador . O agendamento de threads também é um grande problema no multithreading.

**Tipos**

**Multithreading de grão grosso**

O tipo mais simples de multithreading ocorre quando um segmento é executado até que seja bloqueado por um evento que normalmente criaria uma barraca de latência longa. Em vez de aguardar a conclusão da barraca, um processador enfocado passaria a execução para outro segmento que estava pronto para ser executado. Somente quando os dados do thread anterior haviam chegado, o segmento anterior seria colocado de volta na lista de threads prontas para execução .

O objetivo do suporte de hardware de multithreading é permitir uma troca rápida entre um segmento bloqueado e outro thread pronto para ser executado. Para mudar eficientemente entre threads ativos, cada thread ativo precisa ter seu próprio conjunto de registros. Por exemplo, para alternar rapidamente entre dois threads, o hardware do registro precisa ser instanciado duas vezes.

**Interleaved multithreading (grão fino)**

O propósito do multithreading intercalado é remover todos os pontos de dependência de dados da tubulação de execução. Uma vez que um segmento é relativamente independente de outros tópicos, há uma menor chance de uma instrução em uma etapa de pipelining que precisa de uma saída de uma instrução mais antiga na tubulação. Conceitualmente, é semelhante ao multitarefa preventivo usado nos sistemas operacionais; uma analogia seria que a fatia de tempo dada a cada thread ativo é um ciclo de CPU.

O processador troca de contexto em todo ciclo e apenas uma instrução de cada contexto está presente no pipeline a cada instante de tempo.

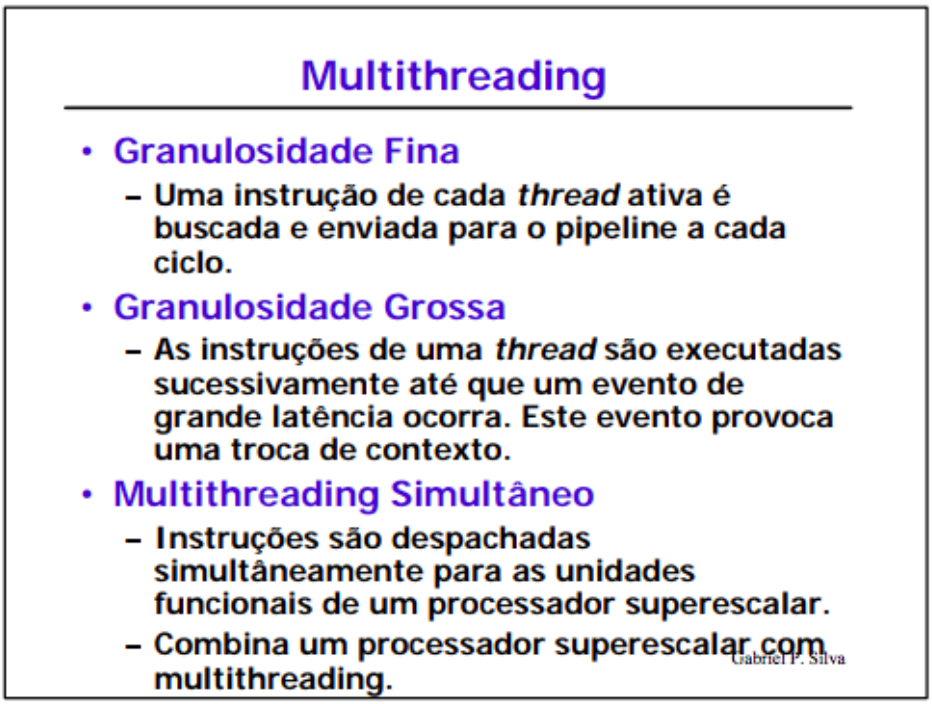
Este tipo de multithreading foi chamado de processamento de barril, no qual os bastões de um barril representam os estágios da tubagem e os seus segmentos de execução. Multilização intercalada, de grão fino ou de corte de tempo são terminologia mais moderna. Além disso, uma vez que existem mais threads sendo executados simultaneamente no pipeline, os recursos compartilhados, como caches e TLBs, precisam ser maiores

**Multithreading simultâneo (SMT)**

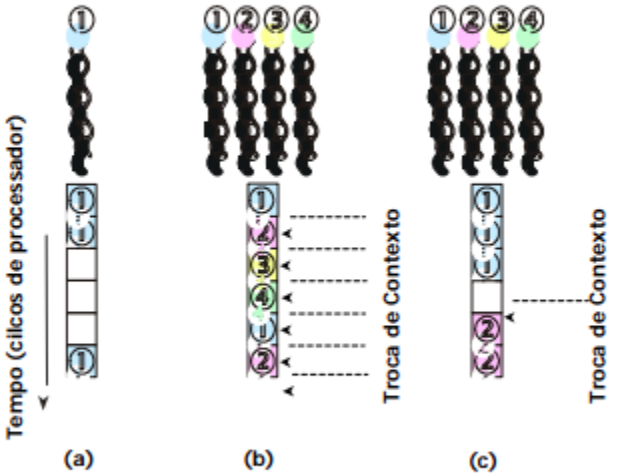
O tipo mais avançado de multithreading se aplica aos processadores superscalares. Enquanto um processador superscalar normal emite várias instruções de uma única thread em cada ciclo de CPU, em multithreading simultâneo (SMT), um processador superscalar pode emitir instruções de vários threads em cada ciclo de CPU. Reconhecendo que qualquer thread único tem uma quantidade limitada de paralelismo de nível de instrução , esse tipo de multithreading tenta explorar o paralelismo disponível em vários segmentos para diminuir o desperdício associado a slots de problemas não utilizados.

Para distinguir os outros tipos de multithreading de SMT, o termo " multithreading temporal " é usado para denotar quando as instruções de um único segmento podem ser emitidas por vez.

Além dos custos de hardware discutidos para multithreading intercalados, a SMT possui o custo adicional de cada etapa do encanamento, rastreando a identificação do segmento de cada instrução que está sendo processada. Novamente, os recursos compartilhados, como caches e TLBs, devem ser dimensionados para o grande número de threads a serem processados.



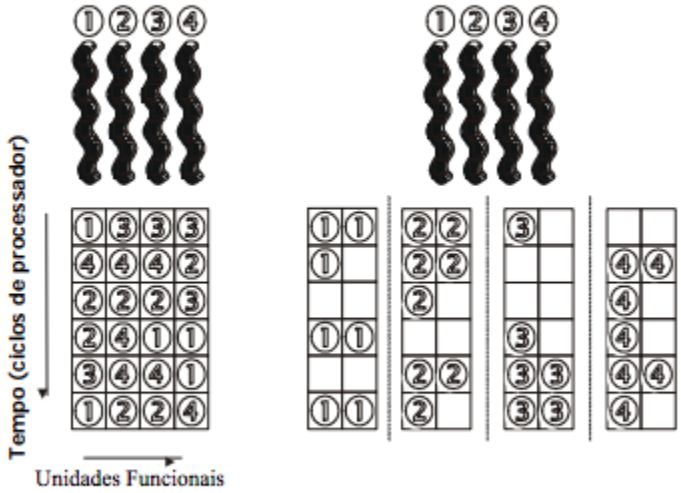
# Multithreading




(a) Processador c/ Pipeline Simples
(b) Multithreading c/ Granulosidade Fina
(c) Multithreading c/ Granulosidade Grossa

Gabriel P. Silva

# Multithreading Simultâneo (SMT)




Gabriel P. Silva

## Cap 18)

### Increase in Parallelism

The organizational changes in processor design have primarily been focused on increasing instruction-level parallelism, so that more work could be done in each clock cycle. These changes include, in chronological order.

• Pipelining:Individual instructions are executed through a pipeline of stages so that while one instruction is executing in one stage of the pipeline, another instruction is executing in another stage of the pipeline.

• Superscalar: Multiple pipelines are constructed by replicating execution resources. This enables parallel execution of instructions in parallel pipelines, so long as hazards are avoided.

• Simultaneous multithreading (SMT): Register banks are replicated so that multiple threads can share the use of pipeline resources.

For each of these innovations, designers have over the years attempted to increase the performance of the system by adding complexity. In the case of pipelining, simple three-stage pipelines were replaced by pipelines with five stages, and then many more stages, with more stages, there is the need for more logic.

With superscalar organization, performance increases can be achieved by increasing the number of parallel pipelines. Again, there are diminishing returns as the number of pipelines increases. More logic is required. Eventually, a single thread of execution reaches the point where hazards and resource dependencies prevent the full use of the multiple pipelines available. This same point of diminishing returns is reached with SMT, as the complexity of managing multiple threads over a set of pipelines limits the number of threads and number of pipelines that can be effectively utilized.

There is a related set of problems dealing with the design and fabrication of the computer chip. Amounts of the chip area is occupied with coordinating and signal transfer logic. This increases the difficulty of designing, fabricating, and debugging the chips. The increasingly difficult engineering challenge related to processor logic is one of the reasons that an increasing fraction of the processor chip is dedicated to the simpler memory logic. Power issues, discussed next, provide another reason.

### Power Consumption

To maintain the trend of higher performance as the number of transistors per chip rise, designers have resorted to more elaborate processor designs (pipelining, superscalar, SMT) and to high clock frequencies. Unfortunately, power requirements have grown exponentially as chip density and clock frequency have risen.

One way to control power density is to use more of the chip area for cache memory. Memory transistors are smaller and have a power density an order of magnitude lower than that of logic.

encapsulated in a rule of thumb known as Pollack's rule. In other words, if you double the complexity in a processor core, then it delivers only 40% more performance. In principle, the use of multiple cores has the po- tential to provide near-linear performance improvement with the increase in the number of cores.

Power considerations provide another motive for moving toward a multi- core organization. Because the chip has such a huge amount of cache memory, it becomes unlikely that any one thread of execution can effectively use all that memory. Even with SMT, you are multithreading in a relatively limited fashion and cannot therefore fully exploit a gigantic cache, whereas a number of relatively independent threads or processes has a greater opportunity to take full advantage of the cache memory.

## Software on Multicore

The potential performance benefits of a multicore organization depend on the abil- ity to effectively exploit the parallel resources available to the application. Let us focus first on a single application running on a multicore system. Recall from Chapter 2 that Amdahl's law states that:

Speedup = time to execute program on a single processor/time to execute program on N parallel processors

$$= 1 /((1 - f) + f/N)$$

The law assumes a program in which a fraction (1 - f) of the execution time in- volves code that is inherently serial and a fraction f that involves code that is infinitely parallelizable with no scheduling overhead.

If only 10% of the code is inherently serial (f = 0.9), the running the program on a multicore system with 8 processors yields a performance gain of only a factor of 4.7. In addition, software typically incurs overhead as a result of communication and distribution of work to multiple processors and cache coherence overhead. This results in a curve where performance peaks than then begins to degrade because of the increased burden of the overhead of using multiple processors.

However, software engineers have been addressing this problem and there are numerous applications in which it is possible to effectively exploit a multicore system. Reducing the serial fraction within hardware architectures, operating systems, middleware, and the database application software. Database management systems and database applications are one area in which multicore systems can be used effectively. Many kinds of servers can also effectively use the parallel multicore organization, because servers typically handle numerous relatively independent transactions in parallel. In addition to general-purpose server software, a number of classes of applications benefit directly from the ability to scale transfer tax with the number of cores.

## MULTICORE ORGANIZATION

At a top level of description, the main variables in a multicore organization are as follows:

- The number of core processors on the chip
- The number of levels of cache memory
- The amount of cache memory that is shared

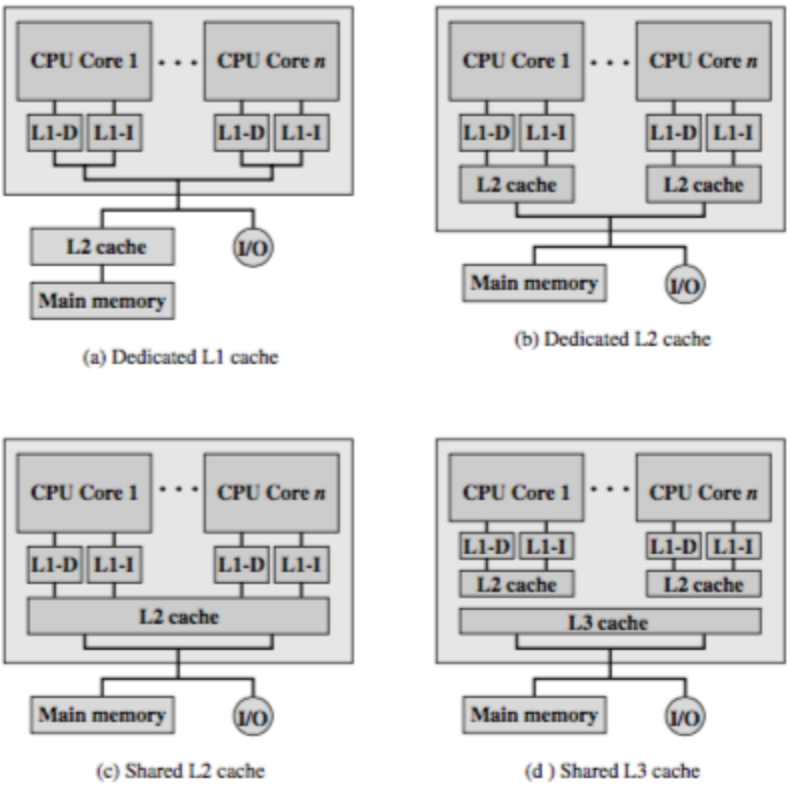



Figure 18.8 Multicore Organization Alternatives

This is an organization found in some of the earlier multicore computer chips and is still seen in embedded chips. In this organization, the only on-chip cache is L1 cache, with each core having its own dedicated L1 cache. Almost invariably, the L1 cache is divided into instruction and data caches.

The use of a shared L2 cache on the chip has several advantages over exclusive reliance on dedicated caches:

1. Constructive interference can reduce overall miss rates. That is, if a thread on one core accesses a main memory location, this brings the frame containing the referenced location into the shared cache. If a thread on another core soon thereafter accesses the same memory block, the memory locations will already be available in the shared on-chip cache.
2. A related advantage is that data shared by multiple cores is not replicated at the shared cache level.
3. With proper frame replacement algorithms, the amount of shared cache allocated to each core is dynamic, so that threads that have a less locality can employ more cache.
4. Interprocessor communication is easy to implement, via shared memory locations.
5. The use of a shared L2 cache confines the cache coherency problem to the L1 cache level, which may provide some additional performance advantage.

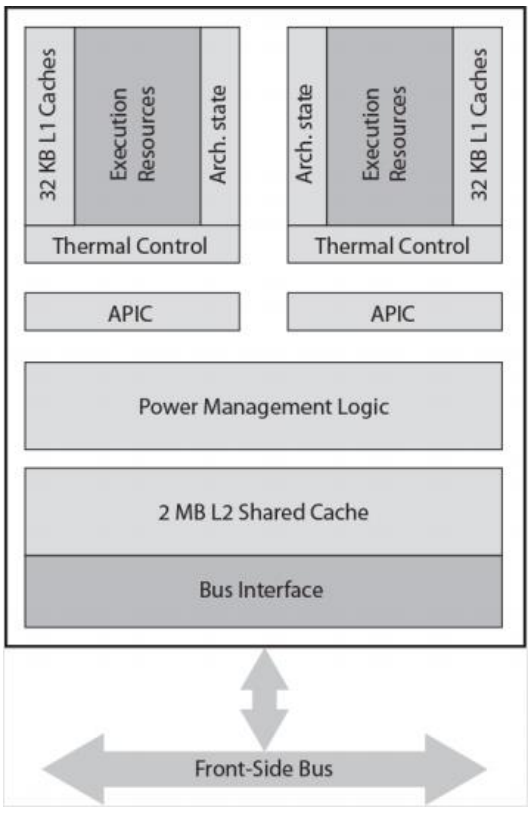
A potential advantage to having only dedicated L2 caches on the chip is that each core enjoys more rapid access to its private L2 cache. This is advantageous for threads that exhibit strong locality.

As both the amount of memory available and the number of cores grow, the use of a shared L3 cache combined with either a shared L2 cache or dedicated per- core L2 caches seems likely to provide better performance than simply a massive shared L2 cache.

Another organizational design decision in a multicore system is if the individual cores will be superscalar or will implement simultaneous multithreading (SMT). For example, the Intel Core Duo uses superscalar cores, whereas the Intel Core i7 uses SMT cores. SMT has the effect of scaling up the number of hardware- level threads that the multicore system supports. Thus, a multicore system with four cores and SMT that supports four simultaneous threads in each core appears the same to the application level as a multicore system with 16 cores. As software is developed to more fully exploit parallel resources, an SMT approach appears to be more attractive than a superscalar approach.

## Organização Intel x86 Multicore - Core Duo (1)

- Dois processadores x86 superescalares, L2 cache compartilhada
- L1 dedicada por núcleo
  - —32KB instrução e 32KB dados
- Unidade de controle térmica por núcleo
  - —Gerencia a dissipação do calor do chip
  - —Maximiza desempenho dentro de algumas restrições
  - —Ergonomia melhorada
- Advanced Programmable Interrupt Controller (APIC)
  - —Interrupções entre núcleos
  - —Encaminha interrupções para o núcleo apropriado



## Organização Intel x86 Multicore - Core i7

- Novembro 2008
- 4 processadoresr x86 SMT
- L2 dedicada, L3 cache compartilhada
- Pré busca especulativa para caches
- Controlador de memória DDR3 no chip
- Caminho de interconexão rápida