ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES III

UNIDADE: ARQUITETURAS PARALELAS

Conteúdo:

1 INTRODUÇÃO

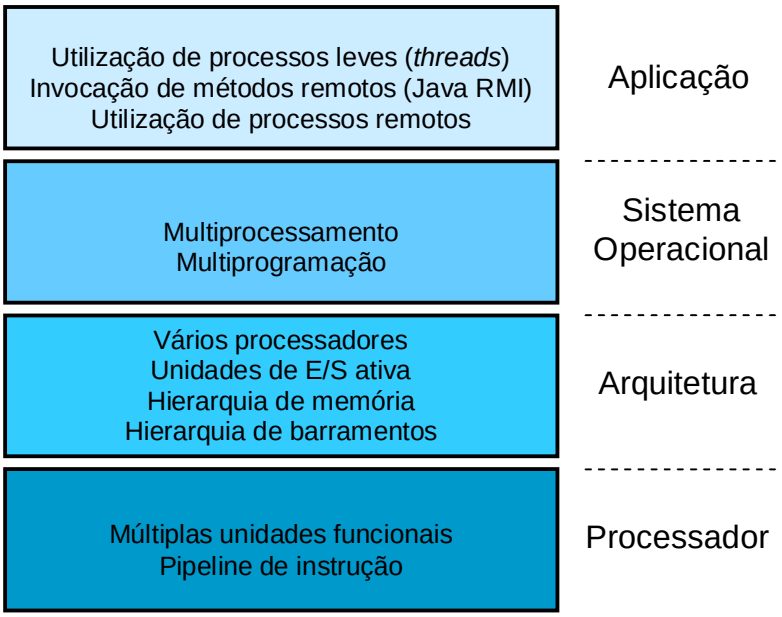
- Apesar do crescente aumento de desempenho dos PC's encontrados no mercado (máquina denominadas convencionais) existem usuários especiais que executam aplicações que precisam de ainda mais desempenho
- Ex:
  - Previsão do tempo
  - Procura de petróleo
  - Simulações físicas
  - Matemática computacional
- Se executadas em máquinas convencionais estas aplicações precisariam de várias semanas ou até meses para executar e em alguns casos extremos nem executariam por causa de falta de memória
- Ex: Compro o melhor PC que encontro no mercado, utilizo um simulador para prever o tempo daqui a três (3) dias ... A previsão pode só ficar pronta depois de uma semana (7 dias) !!!
- Processamento de Alto Desempenho (PAD) é uma área da computação que se preocupa com estes problemas complexos demais para máquinas convencionais
- Estes usuários só têm duas alternativas:
  - Simplificam o seu modelo e acabam tendo resultados menos precisos aumentando a margem de erro (Ex: a simulação de tempo indica uma alta probabilidade de chuva e não chove ☹)
  - Executam a sua aplicação em máquinas mais poderosas do que as convencionais, denominadas arquiteturas especiais ou arquiteturas paralelas
- As **arquiteturas especiais** obtêm um melhor desempenho replicando o número de unidades ativas (normalmente processadores)
- Infelizmente um aumento do número de processadores na arquitetura acaba invariavelmente complicando ainda mais os problemas já encontrados em arquiteturas convencionais e criando novos complicadores
  - A dificuldade que se tinha para alimentar um processador com dados fica ainda maior para vários processadores
  - A programação de máquinas especiais é mais complicada, pois o problema tem que ser particionado (quebrado em partes e distribuído) entre as várias unidades ativas para que execute mais rápido
- O tipo de processamento que ocorre nestas máquinas é denominado **processamento paralelo** pelo fato de várias unidades ativas atuarem em paralelo no mesmo problema com o objetivo de reduzir o tempo total de execução
- Posso ter a implementação deste conceito em vários níveis:

  Superpipeline, multithreaded (hyperthreading), multicore, multiprocessador, *cluster*, *cluster of clusters*, grade computacional

- Os problemas que veremos se aplicam em todos estes níveis, independentemente da plataforma de execução escolhida

1.1 Processamento Paralelo (PP)

- Definição: várias unidades ativas colaborando na resolução de um mesmo problema
- As várias unidades ativas cooperam para resolver o mesmo problema, atacando cada uma delas uma parte do trabalho e se comunicando para a troca de resultados intermediários ou no mínimo para a divisão inicial do trabalho e para a junção final dos resultados
- Exemplos de programas paralelos:
  - Uma aplicação escrita em C ou Java com várias *threads*
  - Uma aplicação escrita em Java usando RMI (Remote Method Invocation)
  - Uma aplicação escrita em C que foi quebrada em vários processos que se comunicam por *sockets*
- Um programa que não foi **preparado** para executar com várias unidades ativas (implementado com apenas um processo que não dispara múltiplas *threads*) **não** executa mais rápido em uma máquina dual!!!
- O programa não é automaticamente "quebrado" pelo sistema operacional e só executa em um único processador não se aproveitando de outros processadores que possam estar disponíveis na arquitetura (no caso da máquina dual)
- **Motivação** para o uso de processamento paralelo
  1. Desempenho
  - Espero reduzir o tempo de execução devido a utilização de diversas unidades ativas na resolução do problema
  2. Tolerância a falhas
  - Espero reduzir a probabilidade de falhas no cálculo, pois cada unidade ativa calcula o mesmo problema e faço uma eleição no final
  3. Modelagem
  - Espero reduzir a complexidade da modelagem e conseqüentemente da implementação da aplicação utilizando uma linguagem que expresse paralelismo (em situações onde o problema é em sua essência paralelo)
  4. Aproveitamento de recursos
  - Espero aproveitar melhor os recursos disponíveis na rede executando uma aplicação com múltiplos processos
- Exploração de paralelismo está presente nos diversos **níveis** de um sistema:



- **Grão de Paralelismo:** é um conceito muito importante pois seu entendimento é fundamental para a modelagem de programas paralelos
  - Grão grosso: o trabalho a ser feito pode ser particionado em unidades de trabalho grandes. Mesmo pagando um alto custo de comunicação é grande a chance de se obter ganho de desempenho delegando estas unidades de trabalho para outras unidades ativas (o custo do envio é compensado pelo ganho de tempo em atacar o problema com mais unidades)
  - Grão médio: o trabalho a ser feito só pode ser particionado em unidades de trabalho médio. Em caso de um alto custo de comunicação pode ser difícil se obter ganho de desempenho delegando estas unidades de trabalho para outras unidades ativas (o custo do envio não é necessariamente compensado pelo ganho de tempo em atacar o problema com mais unidades)
  - Grão fino: o trabalho a ser feito só pode ser particionado em unidades de trabalho pequenas. Em caso de um alto custo de comunicação não vale a pena delegar estas unidades de trabalho para outras unidades ativas (o custo do envio não é compensado pelo ganho de tempo em atacar o problema com mais unidades)
- **Processamento Paralelo x Distribuído:** questões mais conceituais dependem muito de cada autor e de sua respectiva formação:
  - Em ambas as áreas tenho os mesmos complicadores (custo de comunicação, distribuição dos dados, dependências), mas a motivação é diferente [De Rose]
  - No caso de **Processamento Paralelo** a motivação foi o ganho de desempenho e as unidades ativas estão normalmente na mesma máquina resultando em custos de comunicação menores (baixa latência)
  - No caso de **Processamento Distribuído** a motivação foi a modelagem e o aproveitamento de recursos e as unidades ativas estão normalmente mais afastadas (em uma rede local ou até na Internet) resultando em custos de comunicação maiores (alta latência)

- Poderia supor que se quero aumentar o desempenho de uma aplicação só precisaria executar o programa com mais unidades ativas ... mas infelizmente tenho **complicadores** ☹:
  - Dependências de dados
  - Distribuição dos dados
  - Sincronização
  - Áreas críticas
- Exemplo: construção de um muro
  - Um pedreiro faz o muro em 3 horas, dois pedreiros fazem em 2 horas, três pedreiros em 1 hora e meia, 4 pedreiros em duas horas (aumentou o tempo !!!)
- A quantidade de trabalho a ser feita limita o número de unidades ativas que podem ser usadas de forma eficiente
- Um número muito grande de unidades ativas para uma quantidade limitada de trabalho faz com que os recursos mais se atrapalhem do que se ajudem (o que faz o tempo aumentar e não diminuir !!!)
- A incidência de muitos complicadores faz com que o ganho de desempenho não seja proporcional ao acréscimo de unidades ativas utilizadas (a duplicação do número de pedreiros de 1 para dois não reduziu o tempo de execução pela metade no exemplo acima)
- Os complicadores no caso da construção do muro são os seguintes:
  - O muro só pode ser feito de baixo para cima (dependência de dados)
  - Os tijolos têm que ser distribuídos entre os pedreiros (distribuição dos dados)
  - Um pedreiro não pode levantar o muro do seu lado muito na frente dos outros pedreiros (sincronização – ritmo de subida do muro é dado pelo pedreiro mais lento)
  - Se só existir um carrinho com cimento este será disputado por todos os pedreiros que farão uma fila para acessar o cimento. Enquanto um pedreiro acessa o cimento os outros perderão tempo esperando nessa fila (áreas críticas)

1.2 Medidas Básicas de Desempenho

- Índices que indicam o desempenho de diferentes aspectos de um programa paralelo
  - Desempenho da aplicação
  - Desempenho da rede de interconexão

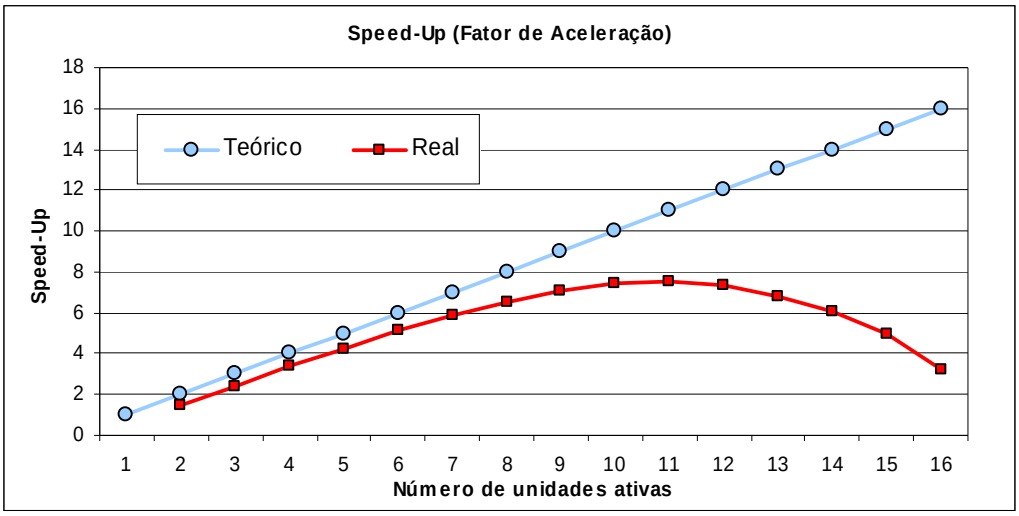
1.2.1 Desempenho da aplicação

Speed-Up (fator de aceleração)

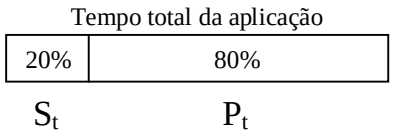
- Indica quantas vezes o programa paralelo ficou mais rápido que a versão seqüencial
- É calculado pela razão entre o melhor tempo seqüencial e o tempo da versão paralela

$$SU_p(w) = \frac{T(w)}{T_p(w)}$$

- Onde ***p*** é o número de unidades ativas utilizadas e ***w*** o trabalho que foi calculado



- Se ***SU* > 1** a versão paralela reduziu o tempo de execução (ficou mais rápido que a seqüencial) e se ***SU* < 1** a versão paralela aumentou o tempo de execução (ficou mais lenta que a seqüencial)
- Cada aplicação tem a sua curva (mais ou menos acentuada) dependendo do trabalho e da incidência de complicadores (aplicação mais ou menos amarrada)
- Toda aplicação tem um número de unidades ativas ideal para a obtenção do melhor desempenho (*sweetspot*) não sendo verdade que quanto mais unidades ativas melhor
- Lei de Amdahl´s (Engenheiro Gene Amdahl, final dos anos 70) define um máximo teórico para o Speed-Up em função das componentes não paralelizáveis (S_t) e paralelizáveis (P_t) de uma aplicação



$$SpeedUp_{max} = \frac{1_t}{S_t + (P_t/n)} = n \underset{P_t \to 0}{\to} \infty = \frac{1_t}{S_t}$$

– Para o exemplo acima o Speed-Up máximo fica:

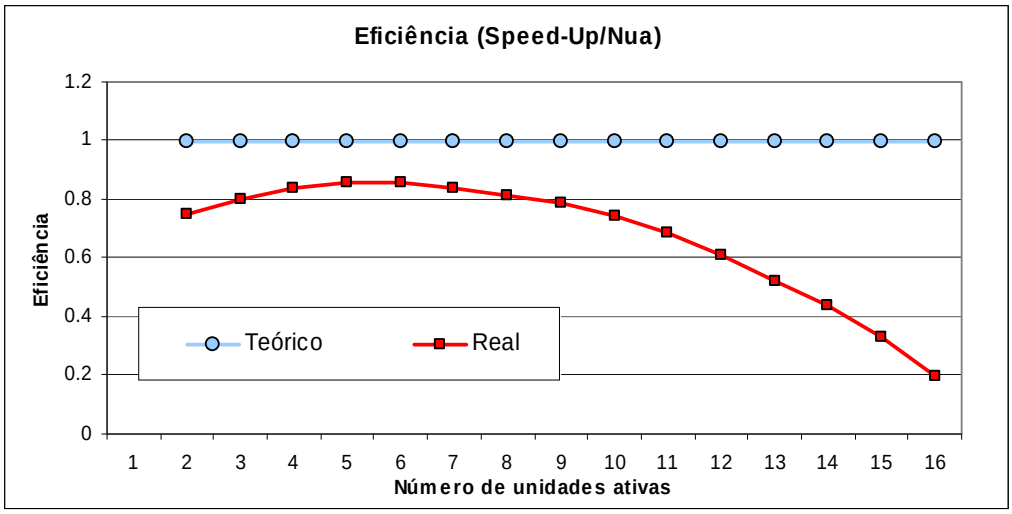
$$SpeedUp_{max} = \frac{1}{S\%} = \frac{1}{0.2} = 5$$

Eficiência

- Indica como foi a taxa de **utilização média** das unidades ativas utilizadas
- Mostra se os recursos foram bem aproveitados
- É calculado pela razão entre o *Speed-Up* e o número de *unidades ativas* utilizadas

$$E_p(w) = \frac{SU_p(w)}{p}$$

- Onde ***p*** é o número de unidades ativas utilizadas e ***w*** o trabalho que foi calculado



- O ideal seria que cada unidade ativa tivesse ficado 100% do tempo ativa (linha azul)
- Normalmente as unidades ativas ficam parte de seu tempo esperando por resultados de vizinhos o que reduz sua taxa de utilização
- A melhor taxa de utilização média não significa o menor tempo de execução (nas figuras o menor tempo de execução ocorreu com 11 unidades ativas e a melhor taxa de utilização média com 5)
- Exercício: A divisão de obras da PUCRS realiza a pintura da sala de aula nos seguintes tempos, dependendo do número de pintores envolvidos: 1 pintor 30 min, 2 pintores 20 min, 3 pintores 15 min, 4 pintores 14 min. Calcule o fator de aceleração (Speed-Up) e a Eficiência para estes casos.

Pintores	Tempo (min)	Speed-Up	Eficiência
1	30	$SpeedUp_1 = \frac{30}{30} = 1$	$E_1 = \frac{1}{1} = 1$
2	20	$SpeedUp_2 = \frac{30}{20} = 1,5$	$E_2 = \frac{1,5}{2} = 0,75$
3	15	$SpeedUp_3 = \frac{30}{15} = 2$	$E_3 = \frac{2}{3} = 0,67$
4	14	$SpeedUp_4 = \frac{30}{14} = 2,14$	$E_4 = \frac{2,14}{4} = 0,54$

  - Dependendo do critério usado a melhor escolha seria 4 pintores (mais rápido), ou 3 (melhor custo-benefício, pois de 3 para 4 o fator de aceleração aumentou pouco com a eficiência caindo muito. Ou seja, o quarto pintor colaborou muito pouco!)

1.2.2 Desempenho da rede de interconexão

Latência

- É tempo necessário para enviar uma mensagem através da rede de interconexão
- Unidade: medida de tempo, Ex: 4 micro segundos (4 μs)

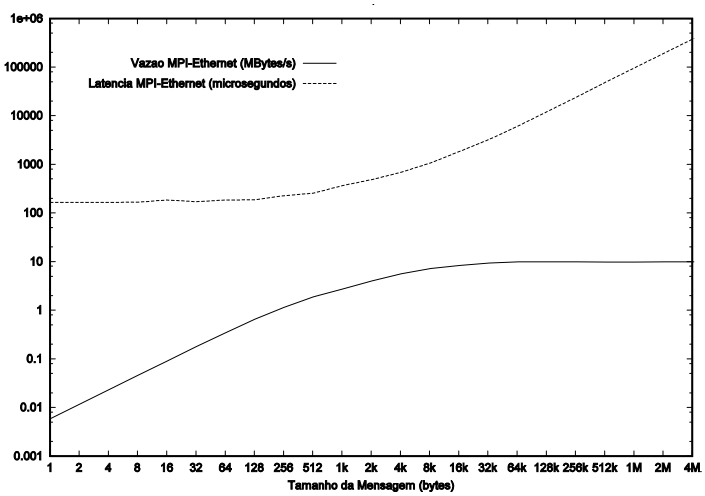
- Inclui o tempo de empacotamento e desempacotamento dos dados mais o tempo de envio propriamente dito
- A latência do envio de uma mensagem de 1 Byte entre duas máquinas rodando GNU/Linux ligadas por tecnologia Fast-Ethernet é de aproximadamente 150 µs (ver figura abaixo)
- A latência aumenta a medida que a quantidade de dados a serem enviados aumenta, mas não de forma linear, pois a componente do tempo referente ao custo de empacotamento e desempacotamento não varia tanto em relação ao tamanho da mensagem como a componente de custo de envio pela rede (ver figura abaixo)

Vazão

- Expressa a capacidade da rede de "bombear" dados entre dois pontos
- Unidade: quantidade de dados por unidade de tempo, Ex: 10 MBytes/segundo (10 MB/s)
- A vazão é afetada pela "largura" do canal de comunicação (expressa normalmente em bits) e pela freqüência da transmissão dos dados (expressa em MHz) segundo a seguinte fórmula:

$$V = L * F$$

- A melhor vazão entre dois pontos medida em uma rede Fast-Ethernet fica em torno de 10 MBytes/s, bastante próximo do seu limite teórico de 12,5 MBytes (100 MBits/s) e foi obtida com uma mensagem de aproximadamente 64 KBytes



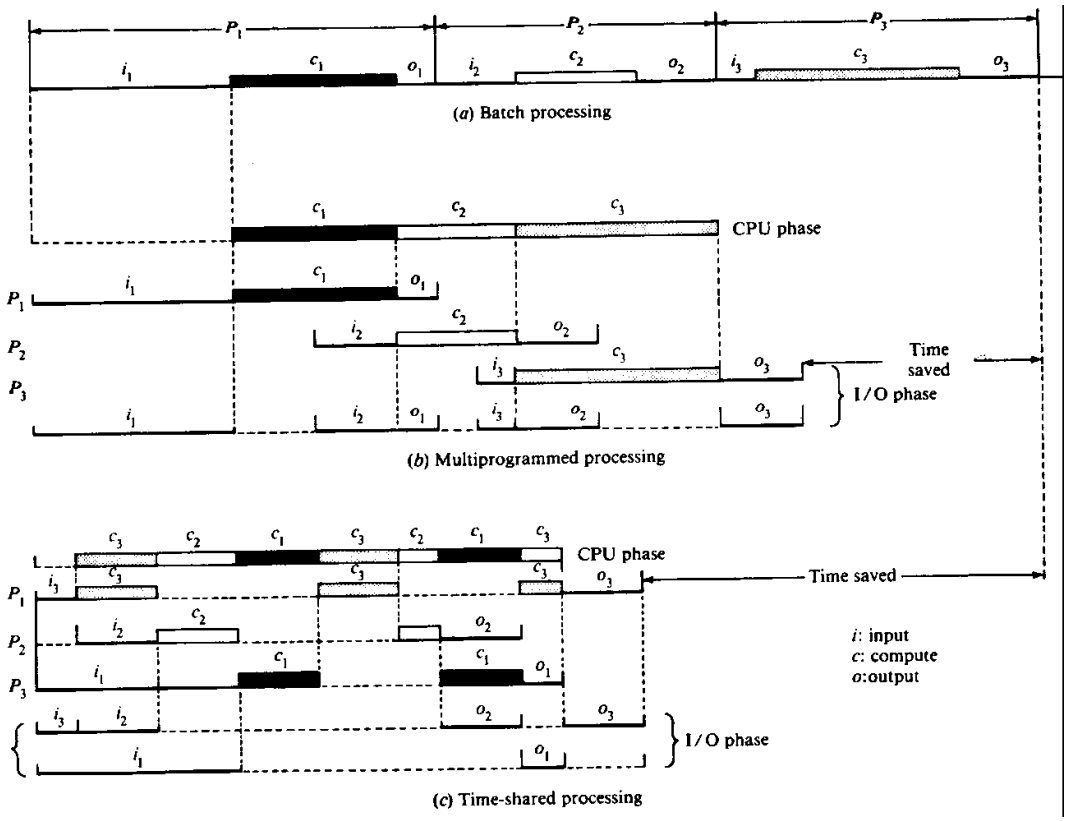
2 PARALELISMO EM ARQUITETURAS COM UM ÚNICO PROCESSADOR

- Para termos processamento paralelo precisamos de várias unidades ativas
- O processador é uma unidade ativa, mas não é a única em uma arquitetura moderna
- Temos hoje partes autônomas dentro do processador (Ex: unidades lógicas e aritméticas) e subsistemas autônomos fora do processador (Ex: unidades de E/S)
- Portanto podemos ter processamento paralelo também em máquinas com apenas um processador

- Serão vistas 5 formas de paralelismo em arquiteturas com um único processador (monoprocessadas)
  - Multiprogramação
  - *Pipelines* Especiais
  - Máquinas Vetoriais
  - *Hyperthreading*
  - *Multicore*

2.1 Multiprogramação

- Durante sua execução um processo intercala entre fases de cálculo e fases de E/S
- A quantidade, o tamanho, e a ordem dessas fases varia de processo para processo
- Se o processo fizer muita E/S é denominado ***I/O bound*** e se fizer muito cálculo é denominado ***CPU bound***
- Se o sistema operacional estiver executando apenas um processo por vez quando este fizer E/S o processador vai ficar parado até que a operação seja concluída
- Na técnica de multiprogramação são mantidos ativos vários processos ao mesmo tempo de forma que aumente a chance do sistema operacional de sempre encontrar um processo pronto para rodar (que não esteja esperando por E/S)
- Como os processos ativos se revezam na utilização da CPU, se a técnica for bem aplicada todos os usuários acreditam que estão rodando "ao mesmo tempo"
- O objetivo maior da multiprogramação é aumentar a taxa de utilização do processador, ou seja, mantê-lo ocupado a maior parte do tempo
- A multiprogramação só se tornou possível a partir do momento que as unidades de E/S se tornaram ativas e passaram a executar as operações de forma autônoma liberando o processador para outras tarefas
- Sendo assim pode se dizer que na multiprogramação ocorre processamento paralelo entre o processador e as unidades de E/S, todas unidades ativas envolvidos na resolução de um problema (executar um grupo de programas)
- O escalonador do sistema operacional é responsável por escolher dentre os processos ativos qual vai para que unidade ativa do sistema
- O exemplo abaixo demonstra como a multiprogramação acelera a execução de um conjunto de processos sobrepondo sua execução (b) em relação a execução seqüencial do mesmo conjunto de programas (a)



- A execução (c) do exemplo utiliza três otimizações:
  - Alteração da ordem de chegada
  - Fatias de tempo
  - Utilização de múltiplas unidades de E/S

2.1.1 Alteração da ordem de chegada

- Por uma questão de justiça o sistema operacional deveria tratar as requisições pela ordem de chegada de forma que o primeiro a chegar fosse o primeiro a ser atendido e também a terminar a sua execução (FIFO – *First In First Out*)
- Porém em alguns casos é interessante para o desempenho do sistema como um todo que esta **ordem de chegada** não seja respeitada (posso conseguir uma redução no tempo médio de atendimento das requisições tornando o sistema mais eficiente)
- SJF (*Smallest Job First*), por exemplo, evita que um processo demorado seja executado primeiro (mesmo que chegue antes) o que elevaria o tempo de atendimento dos processos restantes
- Sempre que são feitas alterações na ordem de atendimento tem que se tomar cuidado para não postergar indefinidamente uma requisição (sempre passo alguém na frente e acabo nunca atendendo o processo mais pesado, Poe exemplo, no caso de SJF)

2.1.2 Fatias de tempo (*time sharing*)

- Uma outra questão crítica relacionada a multiprogramação é o tempo de resposta

- Pode acontecer que um processo *CPU bound* ganhe a CPU e demore muito para fazer E/S de forma que o escalonador não seja chamado para dar chance aos outros processos
- Em uma situação dessas os processos que estão esperando na fila de pronto ficam muito tempo sem ter acesso a CPU e o tempo de resposta médio do sistema aumenta consideravelmente quebrando a ilusão de que os processos estão rodando "ao mesmo tempo"
- Para evitar que um processo possa ficar por muito tempo utilizando a CPU foi definido o conceito de fatia de tempo.
- Uma **fatia de tempo** (*time slice*) é o tempo máximo que um processo pode executar antes de dar a vez ao próximo da fila de pronto
- Esta técnica é denominada ***time sharing*** (compartilhamento do tempo de CPU)
- Se a causa da troca de um processo foi o estouro de sua fatia de tempo o processo não é colocado na fila de espera por E/S (nenhuma operação de E/S foi solicitada) mas sim no final da fila de pronto
- Para que o escalonador possa retirar um processo dessa forma o sistema tem que suportar **preempção**
- Uma fatia muito grande faz com que o comportamento do sistema fique parecido com FIFO, uma fatia muito pequena tende a SJF (*Smallest Job First*)
- Uma fatia muito pequena resulta em muita troca de processos o que acaba tendo um alto custo para o sistema por causa do salvamento de contexto (sistema guarda os registradores e algumas tabelas para que possa retomar o processo onde parou)
- Preciso achar um tamanho de fatia que seja um compromisso entre qualidade e custo

2.1.3 Utilização de múltiplas unidades de E/S

- O aumento do número de unidades de E/S pode levar a um aumento de desempenho da multiprogramação, pois o escalonador pode tratar vários pedidos de E/S ao mesmo tempo
- Como as unidades de E/S são **unidades ativas**, na prática está se aumentando o número de unidades que participam do processamento paralelo, o que tende a aumentar o *Speed-Up* (fator de aceleração) reduzindo o tempo de resolução do problema (no caso a execução de vários processos)

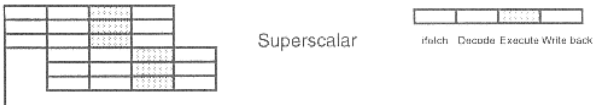
2.2 *Pipelines* Especiais

- A técnica de *pipeline* se baseia na quebra de um procedimento em estágios para que possa ocorrer sobreposição temporal e, como resultado, redução do tempo de execução
- Cada um desses estágios é executado por uma unidade ativa independente, e todas juntas estão atuando na resolução de um mesmo problema (Ex: executar as instruções de um programa), ou seja, de uma certa forma pode ser considerado processamento paralelo
- A técnica de *pipeline* só acelera a execução de um conjunto de operações !!!
- O tempo necessário para a execução de uma operação continua o mesmo (ou fica até um pouco maior por causa do maior controle entre os estágios)
- Ex: um *pipeline* de instruções só acelera a execução de um programa inteiro e não a execução de uma única instrução

- Um processador RISC (*Reduced Instruction Set Computer*) tem um CPI (Ciclos por Instrução) perto de 1 depois que o *pipeline* está cheio (uma instrução fica pronta no final do *pipeline* a cada ciclo)
- A grande vantagem do uso de *pipeline* é que se pode obter uma grande aceleração sem a necessidade que o programador altere o código do programa (transparente)
- Como a técnica de *pipeline* já foi estudada em disciplinas anteriores aqui serão tratadas apenas as técnicas de *pipeline* especiais (utilizam paralelismo para acelerar a execução das operações)
- Com os *pipelines* especiais é possível executar, por exemplo, uma instrução em menos de um ciclo CPI (Ciclos Por Instrução) <1 ou IPC (Instruções Por CIclo) > 1
- É importante destacar que obviamente não é possível executar uma instrução em menos de um ciclo e que o CPI só fica menor do que 1 porque que várias instruções são executadas ao mesmo tempo (em paralelo) de forma que o tempo médio de execução fique menor do que 1
- Para que um CPI médio menor do que 1 seja possível são necessárias as seguintes condições:
    1. Hardware replicado disponível (executar instruções em paralelo)
    2. Ler várias instruções ao mesmo tempo
    3. Evitar as dependências entre essas as instruções
- Vamos analisar os seguintes *pipelines* especiais:
    - Superescalar
    - Superpipeline
    - Super-super

2.2.1 Superescalar

- Processador possui vários *pipelines* de instrução replicados
- A número de *pipelines* replicados dá o **grau** do *pipeline* **superescalar** (no caso da figura o pipeline superescalar tem grau 3)
- Em teoria, dessa forma pode-se processar vários fluxos de instrução em paralelo se não ocorrer conflito entre eles (como, por exemplo, o acesso à memória se todos os *pipes* executarem uma instrução de load – hazard estrutural)



- Considerando que ***n*** é o grau do *pipeline* superescalar o CPI (Ciclos Por Instrução) **teórico** resultante é dado por:

$$CPI = \frac{1}{n}$$

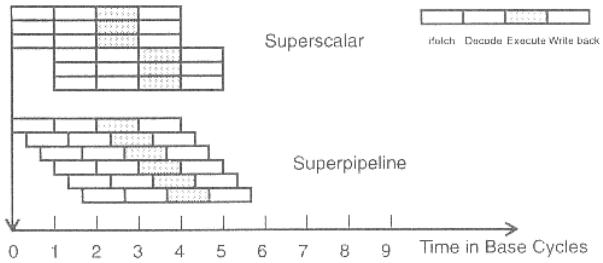
- No *pipeline* superescalar de grau 3 da figura acima o CPI é de 1/3 = 0.33, significando que, no melhor caso, a cada 1/3 de ciclo uma instrução fica pronta em média
- Considerando que ***n*** é o grau do *pipeline* superescalar o IPC (Instruções prontas Por Ciclo) **teórico** resultante é dado por:

$$IPC = n$$

- No *pipeline* superescalar de grau 3 da figura acima o IPC é de 3, significando que, no melhor caso, 3 instruções ficam prontas a cada ciclo em média

2.2.2 Superpipeline

- Processador quebra os estágios do *pipeline* de instrução em sub-estágios (*pipeline* dentro de cada estágio do *pipeline* original)
- Como o pipeline resultante acaba ficando com mais estágios (considerando os sub-estágios) esta técnica também é chamada de pipeline profundo
- A número de sub-estágios utilizados na quebra dá o **grau** do **superpipeline** (no caso da figura o superpipeline tem grau 3)
- Em teoria, dessa forma pode-se aumentar a sobreposição do pipeline e o ritmo do sistema é dado pelo tempo de execução de um sub-estágio (que é uma fração do tempo do estágio original)
- Esta é uma forma de aumentar a freqüência do sistema, mas a cada ciclo é feito menos trabalho do que no pipeline original (um sub-estágio)



- Considerando que ***m*** é o grau do superpipeline o CPI (Ciclos Por Instrução) **teórico** resultante é dado por:

$$CPI = \frac{1}{m}$$

- No superpipeline de grau 3 da figura acima o CPI é de 1/3 = 0.33, significando que, no melhor caso, a cada 1/3 de ciclo original uma instrução fica pronta em média
- Considerando que ***n*** é o grau do superpipeline o IPC (Instruções prontas Por Ciclo) **teórico** resultante é dado por:

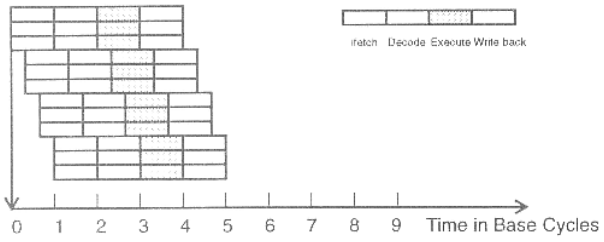
$$IPC = m$$

- No superpipeline de grau 3 da figura acima o IPC é de 3, significando que, no melhor caso, 3 instruções ficam prontas a cada ciclo original em média
- Esta técnica também pode ser usada para balancear um pipeline se os estágios mais demorados foram quebrados em um maior número de sub-estágios que os mais rápidos

2.2.3 Pipeline Super-super

- Processador possui vários *pipelines* de instrução replicados e cada um deles é um superpipeline (*pipeline* profundo) resultado da combinação das duas técnica anteriores
- A número de superpipelines replicados multiplicado pelo número de sub-estágios em que foram quebrados dá o **grau** do *pipeline* **super-super** (no caso da figura o pipeline super-super tem grau 3 * 3 = 9)

- Em teoria, dessa forma pode-se processar vários fluxos de instrução em paralelo se não ocorrer conflito entre eles e ainda aumentar a sobreposição



- Considerando que ***n*** é o número de superpipelines replicados e ***m*** o número de sub-estágios em que foram quebrados o CPI (Ciclos Por Instrução) **teórico** resultante é dado por:

$$CPI = \frac{1}{n * m}$$

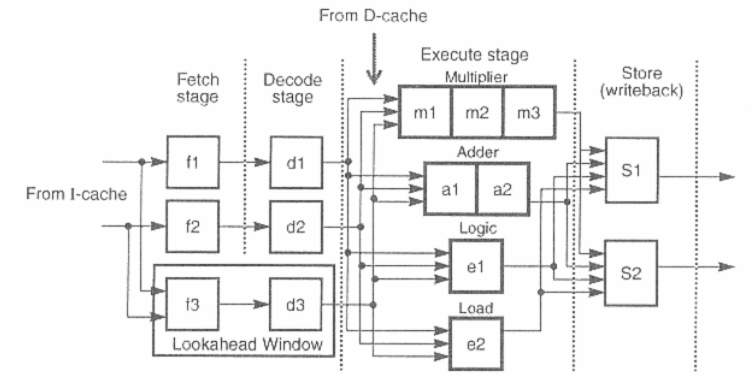
- No *pipeline* super-super de grau 9 da figura acima o CPI é de 1/9 = 0.11, significando que, no melhor caso, a cada 1/9 do ciclo original uma instrução fica pronta em média
- Considerando que ***n*** é o número de superpipelines replicados e ***m*** o número de sub-estágios em que foram quebrados o IPC (Instruções prontas Por Ciclo) **teórico** resultante é dado por:

$$IPC = n * m$$

- No *pipeline* super-super de grau 9 da figura acima o IPC é de 9, significando que, no melhor caso, 9 instruções ficam prontas a cada ciclo original em média

2.2.4 Estudo de Casos – Processador Superescalar de grau 2

- A figura abaixo apresenta o diagrama de blocos de um processador superescalar de grau 2



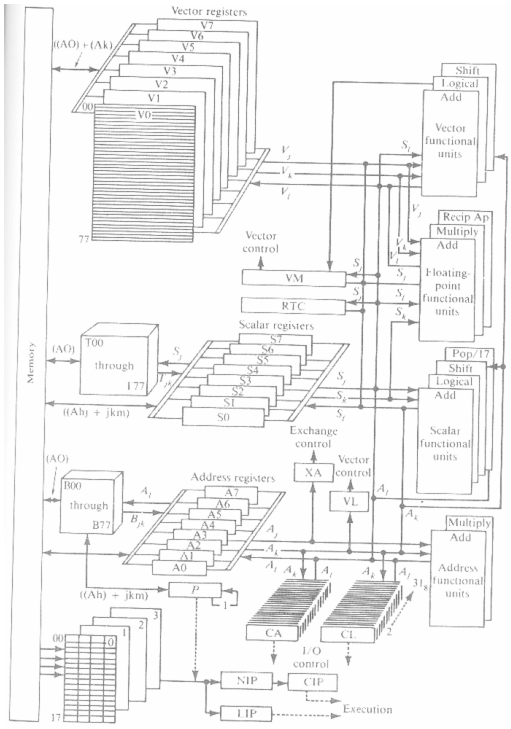
- Para reduzir o custo 4 unidades funcionais são compartilhadas no estágio de execução
- Caso o compilador verifique que uma destas unidades está livre e que uma instrução que necessite esta unidade se encontra na fila para ser executada a janela de ultrapassagem (*lookahead window*) é utilizada para que uma terceira instrução seja executada

2.3 Máquinas Vetoriais

- Arquitetura otimizada para a execução de operações vetoriais (presentes, por exemplo, em aplicações que simulam fenômenos físicos e matemáticos como simulação do tempo)
- O processador possui um conjunto de instruções especiais denominadas vetoriais para operações com o tipo vetor como, por exemplo:
  - VADD, VMUL (V x V →V)
  - VSUM, VMAX (V→S)
  - VSQR, VSIN (V→V)
  - SADD, SDIV (V x S→V)
- Unidades lógicas e aritméticas são replicadas e implementadas com *pipelines* aritméticos para acelerar a execução destas instruções vetoriais (uma operação de multiplicação ponto flutuante, por exemplo, pode ser quebrada em aproximadamente 12 estágios)
- Grandes bancos de registradores são utilizados para alimentar estes *pipelines* de forma eficiente (acabam sendo usados como *caches*)
- Exemplo de código

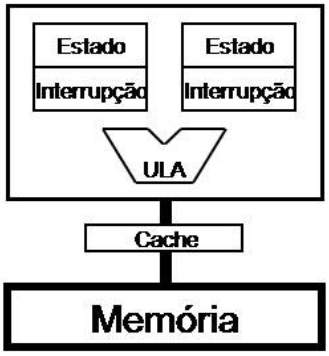
Escalar	Vetorial
`DO 100 I = 1,N`	`A(1:N) = B(1:N) + C(1:N)`
`100 A(I) = B(I) + C(I)`	

- A figura abaixo apresenta a arquitetura da máquina Cray-1 (Hwang, 1985) que tem as seguintes características:
  - 12 unidades de *pipeline* aritmético
  - 8 registradores vetoriais cada um com 64 elementos de 64 bits
  - 10 instruções vetoriais



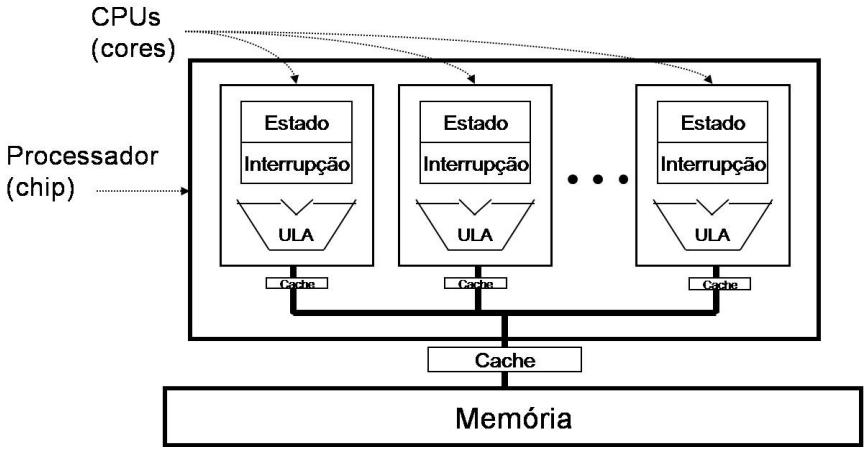
2.4 *Hyperthreading*

- Proposta pela Intel
  - atualmente disponível em processadores Core2Duo, Core2Quad, Xeon
- Abordagem multi-thread
  - 2 threads podem executar simultaneamente no mesmo processador
  - não existe troca de contexto para execução das threads!
- Processador é virtualmente duplicado (2 procs. lógicos)
- Objetivo: melhor utilização de recursos
- Componentes adicionados a um processador tradicional (< 5% da área do chip)
  - lógica de controle e duplicação do módulo referente ao contexto do processo em execução (pilha, regs de controle, etc) → **permitir concorrência na execução dos processos**
  - lógica de controle e duplicação do controlador de interrupções → **permitir a gerência concorrente de interrupções**
- Recursos compartilhados entre processos
  - cache e unidades de execução
- Arquitetura de processadores hyperthreading



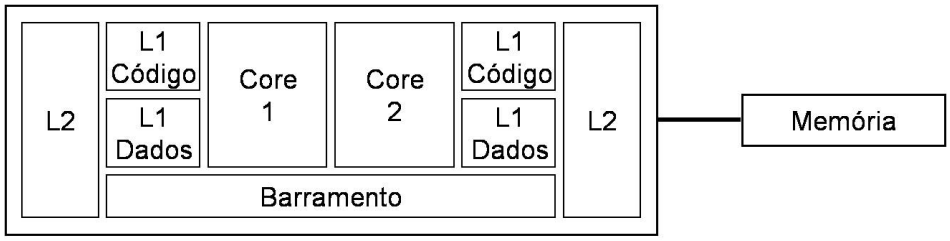
2.5 *Multicore* (material Professores Rafael Santos e Gerson Cavalheiro)

- Tecnologia em hardware na qual múltiplos cores são integrados em um único chip
- Multiplicação total dos recursos de processamento

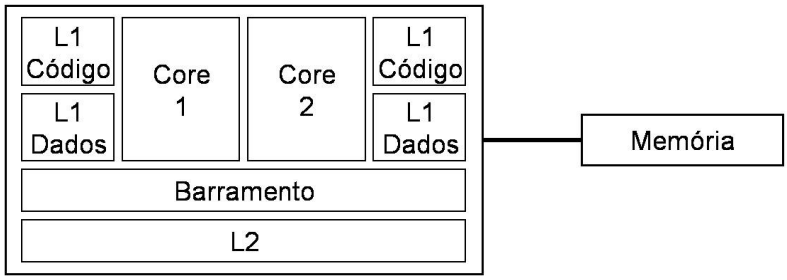


2.5.1 Compartilhamento das *Caches*

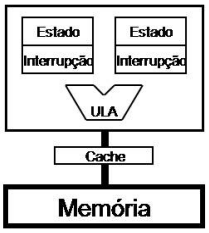
- Com cache L2 privada por core
  - sem disputa no acesso, mas problemas de aproveitamento da área das *caches* por cada processo e de afinidade de memória no escalonamento (processo estava rodando em uma *cache* e retorna do escalonamento em outra)



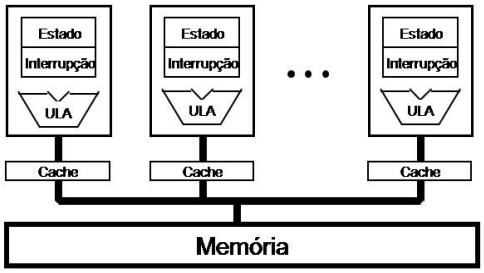
- Com cache L2 compartilhada
  - maior disputa no acesso, mas sem problemas de aproveitamento da área das *caches* pelos processos e também sem problemas de afinidade de memória no escalonamento



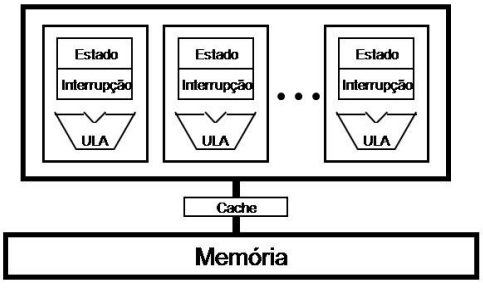
2.5.2 Comparativo



Processador HyperThreading



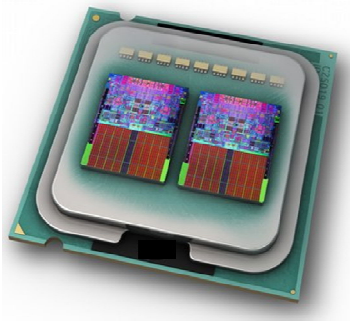
Multiprocessador SMP



Processador multi-core

2.5.3 Multicores comerciais

- AMD
  - Barcelona – Quad-Core Opteron™ (64 bits, 4 cores, L2 privada, L3 compartilhada)
  - Kuma, Rana - AMD AthlonTM X2 Dual-Core (32/64 bits, 2 cores, L2 privada)
  - FASN8 – 8 cores
- Intel
  - Montecito – Itanium 2 (64 bits, 2 cores, L2 e L3 privada)
  - Dempsey – Xeon 5000 (64 bits, 2 cores, L2 privada, HT)
  - Yonah – CoreTM Duo / CoreTM 2 / Core 2TM Extreme (32 bits, 2/4 cores, L2 compartilhada)
- SUN
  - Niagara – UltraSPARC T1 (64 bits, 4/6/8 cores, L2 privada)
- Questão de licença
  - Depende da opção do fornecedor:
    Microsoft: por processador, não por core
    Oracle: por core, com fator de redução por core em processador multi-core
    VMWare: por processador, com limite de 4 cores/processador



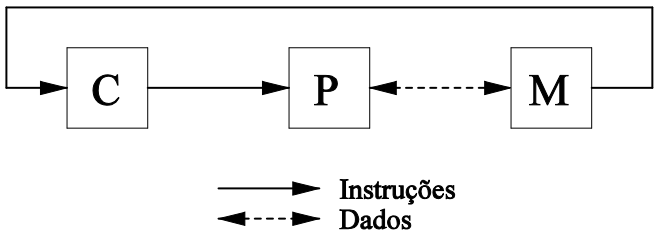
3 PARALELISMO EM ARQUITETURAS COM MÚLTIPLOS PROCESSADORES

3.1 Classificação de Flynn (fluxo de instruções / fluxo de dados)

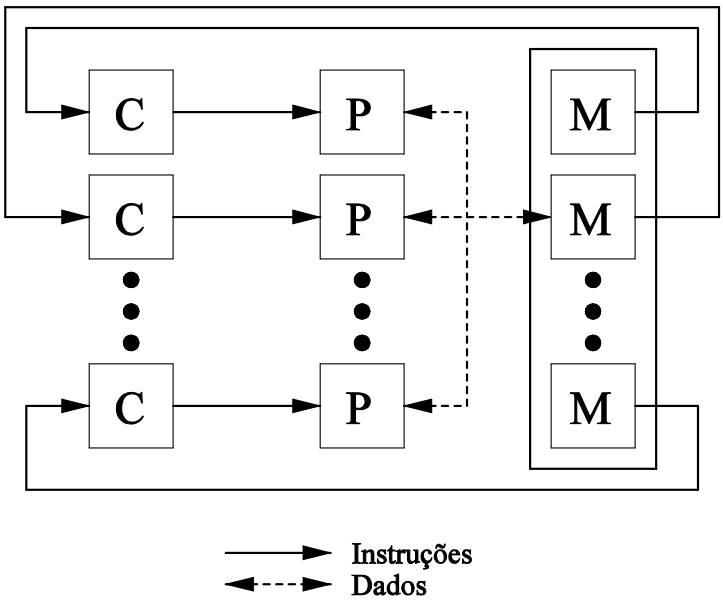
- Para uma classificação inicial de arquiteturas paralelas, pode ser usada a classificação genérica de Flynn
- Apesar de ter sua origem em meados dos anos 70, é ainda válida e muito difundida.
- Baseando-se no fato de um computador executar uma seqüência de instruções sobre uma seqüência de dados, diferenciam-se o fluxo de instruções (*instruction stream*) e o fluxo de dados (*data stream*).
- Dependendo de esses fluxos serem múltiplos ou não, e através da combinação das possibilidades, Flynn propôs quatro classes:

	SD (*Single Data*)	**MD** (*Multiple Data*)
SI (*Single Instruction*)	SISD Máquinas von Neumann convencionais	SIMD Máquinas *Array* (CM-2, MasPar)
MI (*Multiple Instruction*)	MISD Sem representante (até agora)	MIMD Multiprocessadores e multicomputadores (nCUBE, Intel Paragon, Cray T3D)

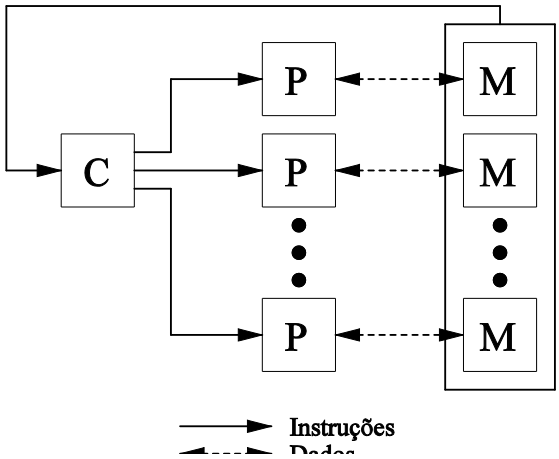
- Na classe **SISD** (*Single Instruction Single Data*), um único fluxo de instruções atua sobre um único fluxo de dados
  - O fluxo de instruções (linha contínua) alimenta uma unidade de controle (C) que ativa a unidade central de processamento (P)
  - A unidade P, por sua vez, atua sobre um único fluxo de dados (linha tracejada), que é lido, processado e reescrito na memória (M)
  - Nessa classe, são enquadradas as máquinas *von Neumann* tradicionais com apenas um processador, como microcomputadores pessoais e estações de trabalho



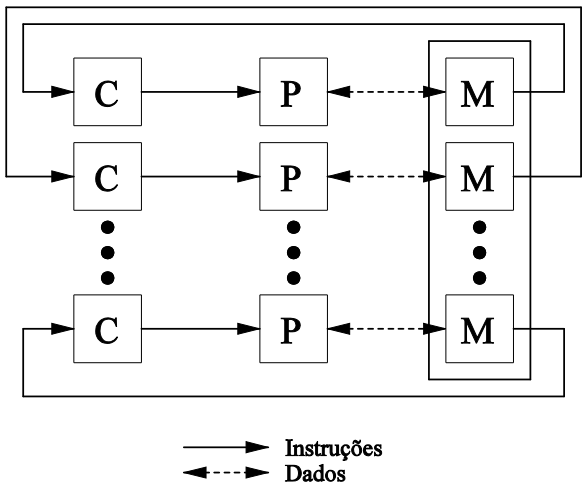
- Na classe MISD (*Multiple Instruction Single Data*), múltiplos fluxos de instruções atuariam sobre um único fluxo de dados
  - Múltiplas unidades de processamento P, cada uma com sua unidade de controle própria C, recebendo um fluxo diferente de instruções
  - Essas unidades de processamento executam suas diferentes instruções sobre o mesmo fluxo de dados
  - Na prática, diferentes instruções operam a mesma posição de memória ao mesmo tempo, executando instruções diferentes
  - Como isso, até os dias de hoje, não faz qualquer sentido, além de ser tecnicamente impraticável, essa classe é considerada vazia



- As **máquinas paralelas** concentram-se nas duas classes restantes, **SIMD** e **MIMD**
- No caso **SIMD** *(Single Instruction Multiple Data),* uma única instrução é executada ao mesmo tempo sobre múltiplos dados.
  - O processamento é controlado por uma única unidade de controle C, alimentada por um único fluxo de instruções
  - A mesma instrução é enviada para os diversos processadores P envolvidos na execução.
  - Todos os processadores executam suas instruções em paralelo de forma **síncrona** sobre diferentes fluxos de dados
  - Na prática, pode-se dizer que o mesmo programa está sendo executado sobre diferentes dados, o que faz com que o princípio de execução SIMD assemelhe-se bastante ao paradigma de execução seqüencial
  - É importante ressaltar que, para que o processamento das diferentes posições de memória possa ocorrer em paralelo, a unidade de memória M não pode ser implementada como um único módulo de memória, o que permitiria só uma operação por vez
  - Nessa classe, são enquadradas as máquinas *Array* como CM-2 e MasPar.



- Enquanto, em uma máquina **SIMD**, só um fluxo de instruções, ou seja, só um programa, está sendo executado, em uma máquina **MIMD** (*Multiple Instruction Multiple Data*), cada unidade de controle C recebe um fluxo de instruções próprio e repassa-o para sua unidade processadora P para que seja executado sobre um fluxo de dados próprio
  - Dessa forma, cada processador executa o seu próprio programa sobre seus próprios dados de forma **assíncrona**
  - Sendo assim, o princípio MIMD é bastante genérico, pois qualquer grupo de máquinas, se analisado como uma unidade (executando, por exemplo, um sistema distribuído), pode ser considerado uma máquina MIMD
  - Nesse caso, como na classe SIMD, para que o processamento das diferentes posições de memória possa ocorrer em paralelo, a unidade de memória M não pode ser implementada como um único módulo de memória, o que permitiria só uma operação por vez
  - Nessa classe, enquadram-se servidores com múltiplos processadores (*dual*, *quad*), as redes de estações e máquinas como CM-5, nCUBE, Intel Paragon e Cray T3D



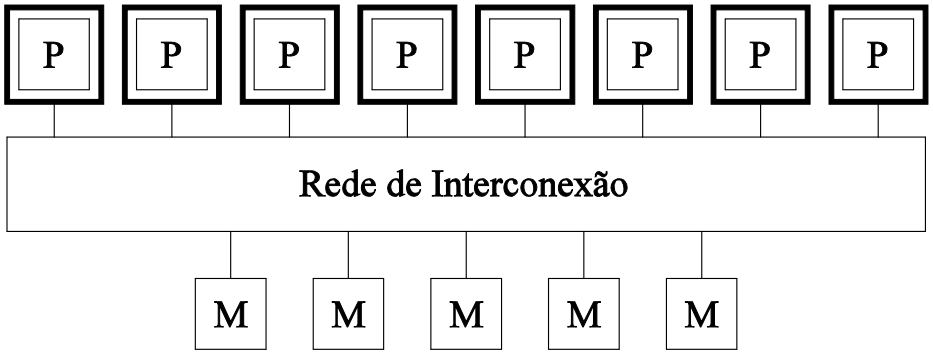
3.2 Classificação segundo o compartilhamento de memória

- Um outro critério para a classificação de máquinas paralelas é o compartilhamento da memória

- Quando se fala em **memória compartilhada** (*shared memory*), existe um único espaço de endereçamento que será usado de forma implícita para comunicação entre processadores, com operações de `load` e `store`
- Quando a memória não é compartilhada, existem **múltiplos espaços de endereçamento privados** (*multiple private address spaces*), um para cada processador. Isso implica comunicação explícita através de troca de mensagens com operações `send` e `receive`
- **Memória distribuída** (*distributed memory*), por sua vez, refere-se à localização física da memória. Se a memória é implementada com vários módulos, e cada módulo foi colocado próximo de um processador, então a memória é considerada distribuída.
- Outra alternativa é o uso de uma **memória centralizada** (*centralized memory*), ou seja, a memória encontra-se à mesma distância de todos os processadores, independentemente de ter sido implementada com um ou vários módulos
- Dependendo de uma máquina paralela utilizar-se ou não de uma memória compartilhada por todos os processadores, pode-se diferenciar:
  - Multiprocessadores
  - Multicomputadores

3.2.1 Multiprocessadores

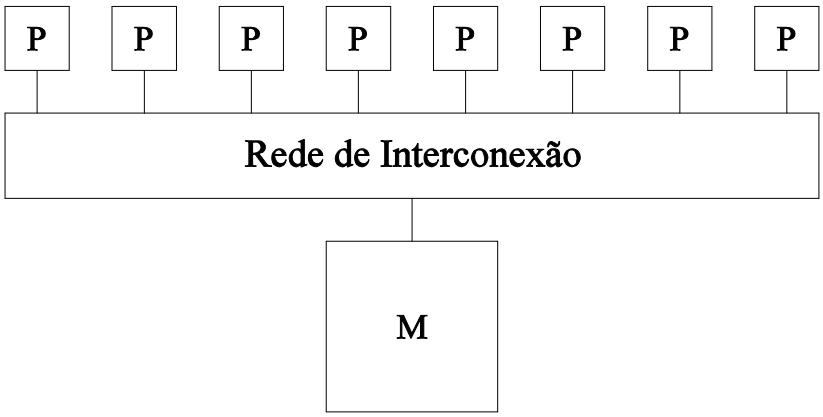
- Todos os processadores *P* acessam, através de uma rede de interconexão, uma memória compartilhada *M*
- Esse tipo de máquina possui apenas um espaço de endereçamento, de forma que todos os processadores *P* são capazes de endereçar todas as memórias *M*



- A comunicação entre processos é feita através da memória compartilhada de forma bastante eficiente com operações do tipo `load` e `store`
- Essas características resultam do fato de esse tipo de máquina paralela ser construída a partir da replicação apenas do componente processador de uma arquitetura convencional (destacados com uma moldura mais escura na figura). Daí o nome **múltiplos processadores**
- Em relação ao tipo de acesso às memórias do sistema, multiprocessadores podem ser classificados como:
  - UMA
  - NUMA
  - COMA

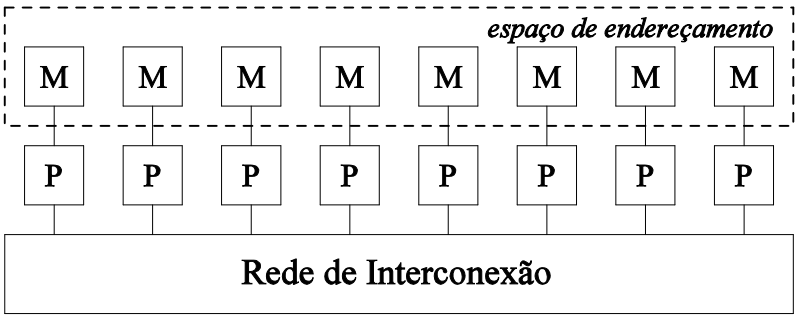
Acesso uniforme à memória (*uniform memory access*, UMA)

- A memória usada nessas máquinas é centralizada e encontra-se à mesma distância de todos os processadores, fazendo com que a latência de acesso à memória seja igual para todos os processadores do sistema (uniforme)
- Como o barramento é a rede de interconexão mais usada nessas máquinas e suporta apenas uma transação por vez, a memória principal é normalmente implementada com um único bloco.
- É importante ressaltar que máquinas com outras redes de interconexão e com memórias entrelaçadas (implementadas com múltiplos módulos e, dessa forma, permitindo acesso paralelo a diferentes) também se enquadram nessa categoria se mantiverem o tempo de acesso à memória uniforme para todos os processadores do sistema



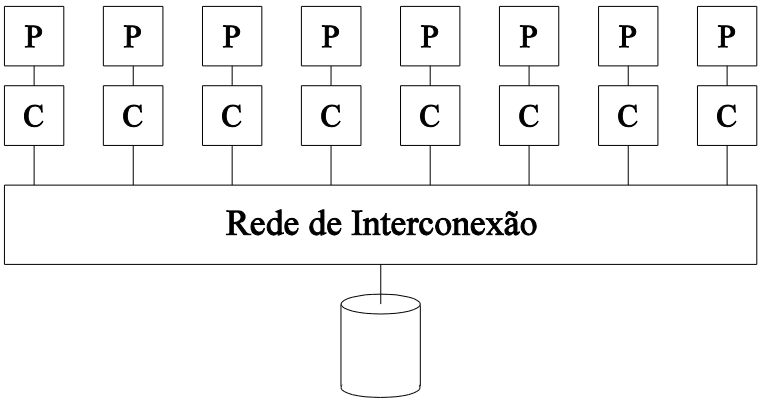
Acesso não uniforme à memória (*non-uniform memory access*, NUMA)

- A memória usada nessas máquinas é distribuída, implementada com múltiplos módulos que são associados um a cada processador
- O espaço de endereçamento é único, e cada processador pode endereçar toda a memória do sistema
- Se o endereço gerado pelo processador encontrar-se no módulo de memória diretamente ligado a ele, dito local, o tempo de acesso a ele será menor que o tempo de acesso a um módulo que está diretamente ligado a outro processador, dito remoto, que só pode ser acessado através da rede de interconexão
- Por esse motivo, essas máquinas possuem um acesso não uniforme à memória (a distância à memória não é sempre a mesma e depende do endereço desejado)



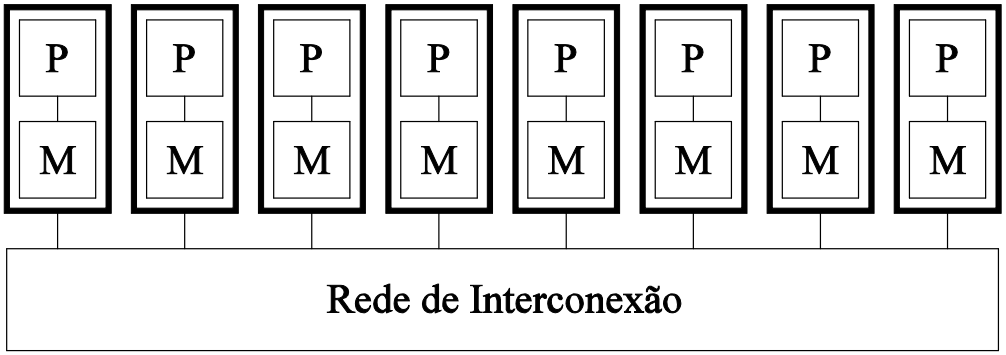
Arquiteturas de memória somente com *cache* (*cache-only memory architecture*, COMA)

- Em uma máquina COMA, todas as memórias locais estão estruturadas como memórias *cache* e são chamadas de COMA *caches*. Essas *caches* têm muito mais capacidade que uma *cache* tradicional
- Arquiteturas coma têm suporte de hardware para a replicação efetiva do mesmo bloco de *cache* em múltiplos nós fazendo com que essas arquiteturas sejam mais caras de implementar que as máquinas NUMA



3.2.2 Multicomputadores

- Cada processador *P* possui uma memória local *M*, à qual só ele tem acesso
- As memórias dos outros processadores são consideradas memórias remotas e possuem espaços de endereçamento distintos (um endereço gerado por P_1 só é capaz de endereçar M_1)
- Como não é possível o uso de variáveis compartilhadas nesse ambiente, a troca de informações com outros processos é feita por envio de mensagens pela rede de interconexão
- Por essa razão, essas máquinas também são chamadas de sistemas de **troca de mensagens** (*message passing systems*)
- Essas características resultam do fato de esse tipo de máquina paralela ser construído a partir da replicação de toda a arquitetura convencional (destacadas com uma moldura mais escura na figura abaixo), e não apenas do componente processador como nos multiprocessadores (daí o nome **múltiplos computadores)**



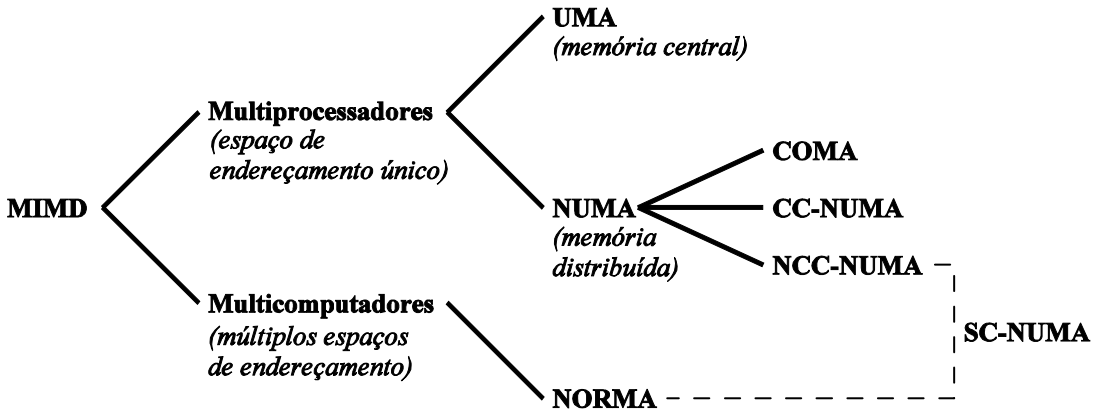
- Em relação ao tipo de acesso às memórias do sistema, multicomputadores podem ser classificados como:
  - NORMA

Sem acesso a variáveis remotas (*non-remote memory access*, NORMA)

- Como uma arquitetura tradicional inteira foi replicada na construção dessas máquinas, os registradores de endereçamento de cada nó só conseguem endereçar a sua memória local

3.2.3 Visão geral da classificação segundo o compartilhamento de memória

- A figura abaixo apresenta uma visão geral da classificação segundo o compartilhamento de memória:



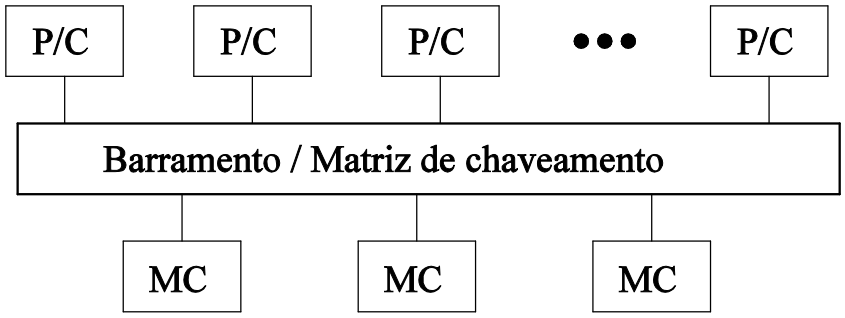
- A linha tracejada indica que as máquinas das classes NCC-NUMA e norma podem ser transformadas em máquinas SC-NUMA através da inclusão de uma camada de software que implemente coerência de *cachê*

3.3 Tendências na Construção de Máquinas Paralelas

- A seguir serão apresentadas as principais tendências para a construção de máquinas paralelas [De Rose, 2003]
- Pode ser considerada uma **classificação mais comercial** que as classificações apresentadas anteriormente
- Segundo a classificação de Flynn, todos os modelos aqui apresentados pertencem à classe MIMD

3.3.1 Multiprocessadores simétricos (SMP)

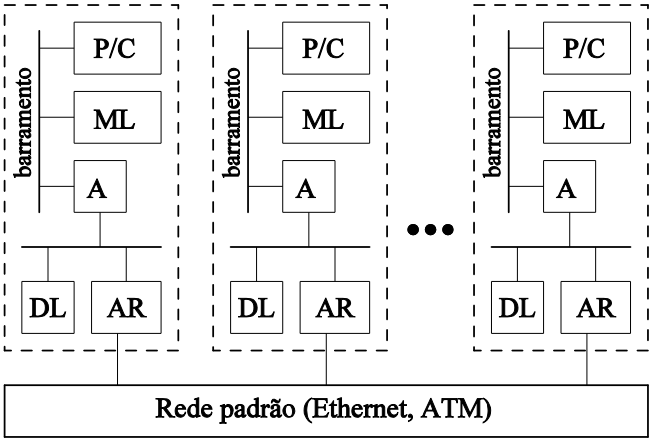
- Multiprocessadores simétricos (SMP - *Symmetric Multiprocessors*) são sistemas constituídos de processadores comerciais, também denominados “de prateleira” (*of the shelf*), conectados a uma memória compartilhada (MC), na maioria dos casos através de um barramento
- Como a maioria dos processadores comerciais encontrados no mercado utiliza-se amplamente de memórias *cache*, tanto no chip quanto fora dele, os processadores foram representados na figura por P/C
- Como conseqüência, o barramento utilizado nessas máquinas implementa coerência de *cache* através do protocolo *snoopy*



- O adjetivo **simétrico** refere-se ao fato de que todos os processadores têm acesso igual ao barramento e à memória, não ocorrendo privilégios por parte do sistema operacional a nenhum dos processadores no atendimento de requisições
- Devido à existência de uma memória compartilhada por todos os processadores, o que caracteriza essas máquinas como multiprocessadores, o paradigma natural de comunicação nesses sistemas é o de memória compartilhada
- Por se tratar de um paradigma mais próximo da programação feita em sistemas convencionas, a programação desses sistemas é considerada mais fácil do que em máquinas que se comunicam por troca de mensagens
- Programas que definem múltiplos fluxos de execução, por exemplo, aproveitam-se automaticamente dos múltiplos processadores desse tipo de sistema (por exemplo, programas Java que possuam várias *threads*).
- Um fator que compromete a escalabilidade dessas máquinas é o uso de um barramento como rede de interconexão
- O barramento impede a construção de multiprocessadores simétricos com um grande número de processadores por se tratar de um canal compartilhado que só permite uma transação por vez. Isso faz com que o seu desempenho e, conseqüentemente, o desempenho global das máquinas caia, à medida que aumenta a disputa por seu acesso (ou seja, que aumenta o número de processadores ligados a ele). As maiores máquinas SMP encontradas hoje no mercado possuem em torno de 100 processadores
- São exemplos de SMP: IBM R50, SGI Power Challenge, SUN Ultra Enterprise 10000, HP/Convex Exemplar X-Class e DEC Alpha Server 8400

3.3.2 Redes de estações de trabalho (NOW)

- Redes de estações de trabalho (NOW – *Network of Workstations*) são sistemas constituídos por várias estações de trabalho interligadas por tecnologia tradicional de rede como Ethernet
- Na prática, uma rede local de estações que já existe é utilizada na execução de aplicações paralelas
- Sob o prisma das arquiteturas paralelas, a rede local pode ser vista como uma máquina paralela em que vários processadores, com suas memórias locais (estações de trabalho), são interligados por uma rede, constituindo assim uma máquina NORMA de baixo custo (ou sem qualquer custo, caso a rede já exista)
- A figura abaixo apresenta a arquitetura dessas máquinas



- Redes tradicionais como Ethernet não são otimizadas para as operações de comunicação de uma aplicação paralela. O resultado é uma alta latência nessas operações, a qual compromete o desempenho da máquina como um todo
- Na prática, devido ao baixo desempenho da rede de interconexão, máquinas NOW são utilizadas como ambientes de ensino de processamento paralelo e distribuído ou na execução de aplicações em que a comunicação entre os nós não é muito intensa
- O exemplo mais comum de NOW são estações de trabalho interligadas por tecnologia Ethernet

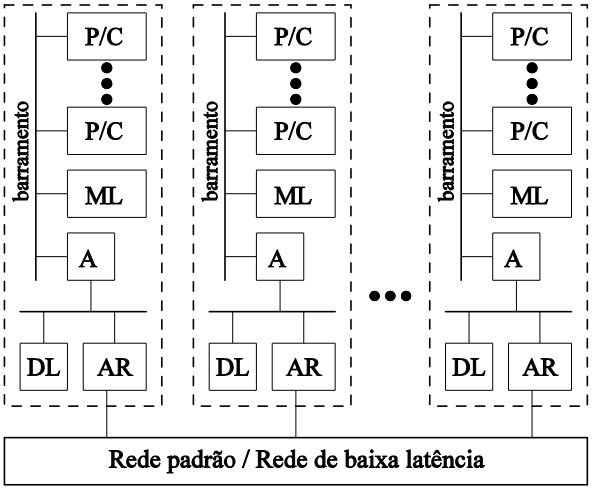
3.3.3 Máquinas agregadas (COW)

- Máquinas agregadas (COW – *Cluster of Workstations*) podem ser vistas como uma evolução das redes de estações de trabalho (NOW)
- Como as NOW, também são constituídas por várias estações de trabalho interligadas, mas com a diferença de terem sido projetadas com o objetivo de executar aplicações paralelas
- Dessa forma, a máquina pode ser otimizada para esse fim e, na maioria dos casos, as estações que servem de nó não possuem monitor, teclado e mouse, sendo denominadas “estação de trabalho sem cabeça“ (*headless wokstation*)
- A máquina resultante também é chamada de “pilha de computadores pessoais” (*Pile-of-PC´s*). Mas as principais otimizações são feitas no software. Como essas estações não serão usadas localmente, o sistema operacional pode ser “enxugado”, e vários servidores, desabilitados
- Várias camadas de rede podem ser simplificadas, ou até mesmo totalmente eliminadas, pois as necessidades de comunicação em máquinas paralelas são diferentes das necessidades em redes locais
- Uma COW pode ser vista como uma NOW **dedicada** ao processamento paralelo e distribuído [De Rose, 2003]
- Já que o sistema está sendo concebido para se dedicar à execução de aplicações paralelas, a rede de interconexão também pode ser otimizada para esse fim. Aqui encontramos atualmente duas tendências:
  - Agregados interligados por redes padrão
  - Esta tendência é impulsionada pelos grandes fabricantes como HP, IBM e Dell, que estão interessados na construção de máquinas paralelas poderosas, agregando

milhares de estações de trabalho de baixo custo (*low end*). Para a interligação de tantas máquinas, fica muito caro investir em uma rede especial, sendo usada, para esse fim, uma rede padrão como a rede Ethernet. O enfoque aqui é a obtenção de desempenho com muitos nós de pequeno poder computacional e, de preferência, com aplicações que não tenham muita necessidade de comunicação

  - Agregados interligados por redes de baixa latência
  - Esta tendência é impulsionada por pequenas empresas que fabricam placas de interconexão especificamente para máquinas agregadas. Essas placas implementam protocolos de rede de baixa latência otimizados para as características de comunicação de aplicações paralelas. Como o custo dessas placas é mais alto do que o de placas de rede padrão, fica muito caro construir máquinas com muitos nós (mais do que algumas centenas de nós). Para compensar o menor número de nós, são usados nós mais poderosos, muitas vezes servidores de médio porte com vários processadores (multiprocessadores SMP)

- A figura abaixo apresenta a arquitetura dessas máquinas. A diferença para a arquitetura de uma NOW é muito pequena, já que se trata de uma evolução do mesmo princípio. A principal diferença é que a rede de interconexão utilizada pode ser tanto padrão como uma rede de baixa latência:



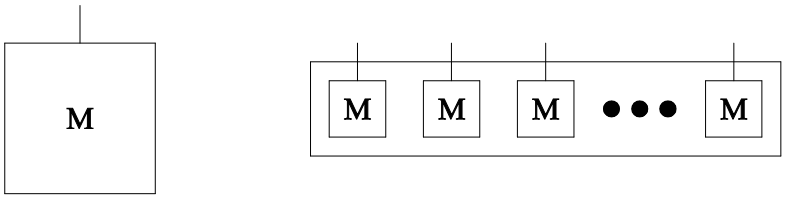
- Entre as principais vantagens na utilização de agregados, temos: ótima relação custo × benefício, alta configurabilidade e baixo custo de manutenção
- São exemplos de COW o iCluster do HP Labs de Grenoble com rede Fast-Ethernet, o Primergy Server do PC2 em Paderborn com rede rápida SCI e a máquina Amazônia do CPAD com rede rápida Myrinet

3.4 Organização da Memória Principal

- Em uma máquina paralela, a memória principal tem um papel fundamental, como vimos na seção de classificações, especialmente em multiprocessadores onde é compartilhada por todos os processadores
- É necessário que se tenha cuidado na escolha do tipo de organização de memória para que se evite uma drástica degradação de desempenho causada por dois ou mais processadores tentando acessar os mesmos módulos do sistema de memória

3.4.1 Memórias entrelaçadas

- Não é desejável que a memória principal seja implementada por um único módulo de forma monolítica mas sim particionada em vários módulos independentes com um espaço de endereçamento único distribuído entre eles
- Essa forma de implementação é chamada de **entrelaçamento** (*interleaving*) e atenua a interferência entre processadores no acesso à memória, permitindo acessos concorrentes a diferentes módulos
- O entrelaçamento de endereços entre *M* módulos de memória é chamado de entrelaçamento de *M*-vias



(a) Bloco único (b) Entrelacada

- É importante destacar que de nada adianta a quebra da memória principal em vários módulos se a rede de interconexão utilizada na máquina não suporta múltiplas transações
- Uma máquina UMA, por exemplo, que se utilize de um barramento para ligar os processadores à memória principal, não se beneficia dos múltiplos canais de uma memória entrelaçada, já que o próprio barramento é o gargalo

3.5 Redes de Interconexão

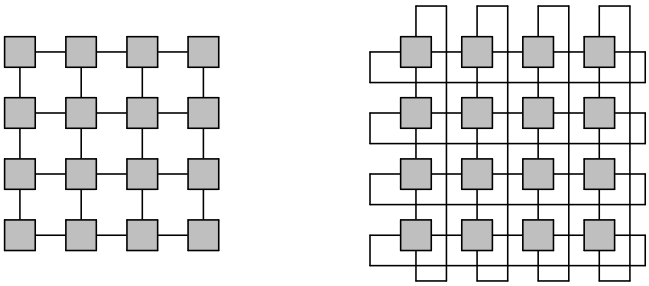
- A forma como os processadores de uma arquitetura são ligados entre si e com outros componentes do sistema (normalmente com a(s) memória(s)) é dada pela rede de interconexão
- Nas duas principais classes de arquiteturas vistas acima, a rede de interconexão desempenha um papel muito importante: nos multiprocessadores, ela pode ajudar a amenizar o problema dos conflitos de acesso, e em multicomputadores, ela influencia diretamente a eficiência da troca de informações
- As redes de interconexão se dividem em dois grandes grupos:
  - Estáticas
  - Dinâmicas

3.5.1 Redes estáticas

- Se os componentes da máquina (processadores, memórias) estão interligados através de ligações fixas, de forma que, entre dois componentes, exista uma ligação direta dedicada, a rede de interconexão é denominada **estática** (ponto-a-ponto)
- Redes estáticas são utilizadas, na maioria dos casos, em multicomputadores. Nesse caso, a **topologia** (estrutura da interligação) determina as características da rede
- Máquinas paralelas possuem quase sempre estruturas regulares com ligações homogêneas, enquanto sistemas distribuídos, por causa da posição geográfica e de

sua integração gradual, possuem estruturas irregulares com ligações heterogêneas (ligações heterogêneas entre redes locais)

- Os principais critérios para a avaliação de uma topologia são o número total de ligações entre componentes, quantas ligações diretas cada componente possui (**grau do nó**) e a maior distância entre dois componentes quaisquer da rede (**diâmetro**)
- O grau do nó pode ser constante ou variar de acordo com o número total de nós da rede (por exemplo, em uma rede em forma de estrela)
- Para cada ligação que um componente possui, é necessária uma interface física correspondente (porta), o que aumenta consideravelmente o custo do componente para muitas ligações
- Em relação ao custo, o melhor caso seria uma rede com um grau de nó baixo e constante
- A estrutura com a menor conectividade (relação entre o número ligações e o número de nós) é o **anel**:
  - Com um grau de nó constante igual a 2, o seu custo é baixo, mas seu diâmetro cresce de forma linear em relação ao número total de nós
  - Se todos os processadores necessitarem trocar dados entre si, pode ocorrer uma sobrecarga do anel, o que acarretaria atraso na transferência dos dados
  - Outra desvantagem do anel é a falta de caminhos alternativos entre os nós, o que resulta em uma baixa confiabilidade
- O outro extremo, em relação ao anel, é a topologia **totalmente conectada**, com um grau de nó igual ao número de nós e um crescimento quadrático do número de conexões em relação ao número de nós, mas com o diâmetro ideal de 1
- Outro fator que pode ser decisivo na escolha de uma topologia, além da relação custo/desempenho, é sua adequação a uma classe específica de algoritmos
- No caso ideal, o padrão de interconexão da topologia corresponde exatamente ao padrão de comunicação da aplicação paralela que executa na máquina
- A **árvore binária**, por exemplo, favorece a execução de algoritmos de divisão e conquista (*divide and conquer*)
  - O seu diâmetro cresce de forma linear em relação à altura *h* da árvore *A(h)* e de forma logarítmica em relação ao número de nós
  - Outras características das árvores binárias são o seu grau de nó máximo de 3 (a raiz tem grau igual a 2) e sua baixa confiabilidade, já que a falha de um nó resulta na perda da ligação com toda a subárvore abaixo dele (particionamento da estrutura)
  - Mesmo que não ocorram falhas, o uso de árvores pode-se tornar problemático, pois todo o fluxo de dados entre a subárvore esquerda e a direita tem que passar pela raiz, a qual se torna rapidamente o gargalo da rede
- Uma topologia muito utilizada em máquinas paralelas é a **malha bidimensional** *M(x,y)*, podendo ter as bordas não conectadas (a) ou ter bordas conectadas de forma cíclica, formando um *torus* bidimensional (b):



(a) Malha (b) Torus

- O grau do nó é constante e igual a 4 (se não forem consideradas as bordas da malha), possibilitando facilmente um aumento do número de processadores em qualquer uma das dimensões (inclusão de linhas ou de colunas) e resultando em uma boa escalabilidade
- Malhas não precisam necessariamente ser quadradas como na figura, podendo uma das dimensões ser menor do que a outra, resultando em diferentes retângulos (o que pode ser vantajoso no caso da escalabilidade, pois permite o aumento do número de processadores em múltiplos da menor dimensão)
- O diâmetro da malha cresce proporcionalmente à raiz quadrada do número de nós, e a existência de caminhos alternativos entre nós aumenta a confiabilidade da rede e diminui o risco de gargalos.
- Malhas são adequadas aos problemas nos quais uma estrutura de dados bidimensional tem que ser processada de forma particionada
- Muitos problemas atuais, que necessitam de grande poder de processamento, possuem essas características, como, por exemplo, operações com matrizes, processamento de imagens e equações diferenciais
- Malhas também podem ter mais de duas dimensões ($d > 2$). O grau do nó é, nesses casos, $2d$
- Uma atenção especial vem sendo dada às estruturas tridimensionais, pois cada vez mais aplicações que necessitam de alto desempenho modelam aspectos físicos do nosso mundo tridimensional. Alguns exemplos são: previsão do tempo, simulação de partículas e aerodinâmica

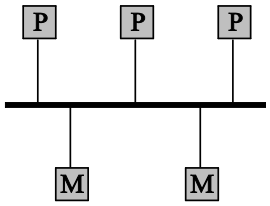
3.5.2 Redes dinâmicas

- Na interconexão de componentes com redes **dinâmicas,** não existe uma topologia fixa que defina o padrão de comunicação da rede
- Quando uma conexão entre dois pontos faz-se necessária, a rede de interconexão adapta-se dinamicamente para permitir a transferência dos dados
- Uma rede dinâmica é dita **bloqueante** quando uma conexão estabelecida entre dois pontos P_1 e P_2 impede o estabelecimento de outra conexão entre componentes quaisquer que não P_1 e P_2
- Em uma rede dinâmica **unilateral,** cada componente possui uma ligação bidirecional com a rede. No caso de uma rede dinâmica **bilateral,** cada componente possui uma ligação de envio e outra de recebimento

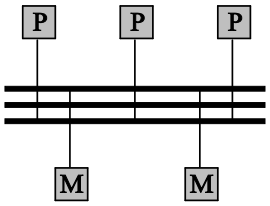
- Enquanto as redes estáticas vistas até agora são utilizadas na maioria dos casos na interconexão de nós de multicomputadores, redes dinâmicas, por sua vez, são tipicamente responsáveis pela interligação de processadores com memórias em multiprocessadores. Algumas redes dinâmicas são também empregadas em multicomputadores
- Para facilitar sua análise, as redes dinâmicas foram divididas em três grupos de acordo com suas características:
  - Barramento
  - Matriz de chaveamento
  - Redes multinível
  -

Barramento

- Dentre as redes dinâmicas, o **barramento** é a alternativa de menor custo
- Porém, por tratar-se de um canal compartilhado por todas as conexões possíveis, tem baixa tolerância a falhas (baixa confiabilidade) e é altamente bloqueante
- Sendo assim, acaba tendo sua escalabilidade comprometida, sendo utilizado em multiprocessadores com um número moderado de processadores (em torno de 100)
- As duas deficiências podem ser amenizadas com o uso de vários barramentos em paralelo



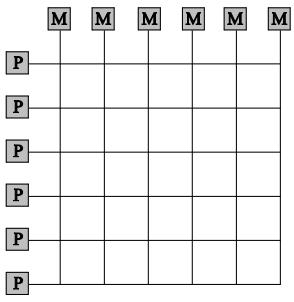
(a) Barramento único



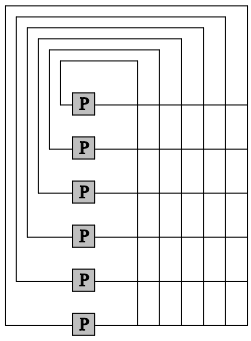
(b) Barramento múltiplo

Matriz de Chaveamento

- A rede dinâmica de maior custo é a **matriz de chaveamento** (*crossbar switch*), que permite o chaveamento entre dois componentes quaisquer, desde que estes não se encontrem já ocupados
- Uma matriz de chaveamento pode ser usada como rede unilateral para ligar processadores a memórias em um multiprocessador (a) ou como rede bilateral para interligar processadores de um multicomputador (b):



(a) Unilateral

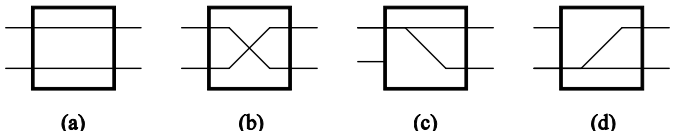


(b) Bilateral

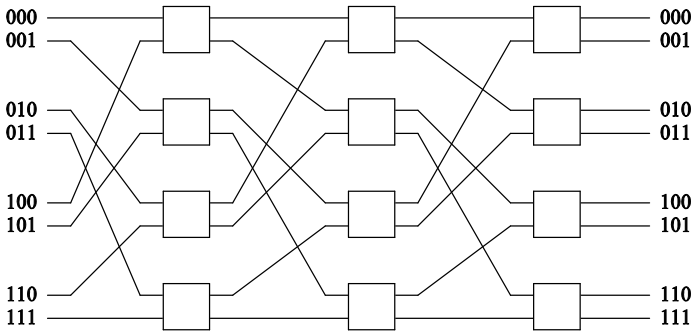
- A matriz de chaveamento não é bloqueante e tem uma escalabilidade boa, permitindo o acréscimo de componentes aos pares
- O alto custo, que cresce de forma quadrática em relação ao número de componentes interligados, inviabiliza, por razões econômicas, a sua utilização para a interconexão de muitos processadores

Redes Multinível

- Uma outra aplicação para redes hierárquicas de matrizes de chaveamento é a construção de **redes de permutação multinível**
- A idéia básica é a ligação de pequenas matrizes de chaveamento (normalmente de tamanho 2×2) em vários níveis consecutivos e conectá-las de forma a reduzir a probabilidade de conflitos entre conexões de diferentes pares
- Diferentemente das redes estáticas, a latência, nesse tipo de rede, é igual para qualquer par comunicante, crescendo, porém, de forma logarítmica de acordo com o número de possíveis conexões
- As matrizes chaveadoras presentes na maioria das redes multinível têm tamanho 2×2 e permitem no mínimo 2 e, na maioria das vezes, 4 posições de chaveamento:



- A figura abaixo mostra um exemplo de rede multinível denominada ***Omega***:
  - O número de linhas é dado pela metade do número de possíveis componentes *n*, o número de níveis de $\log_2 n$, e, no total, $(n/2)$
  - $\log_2 n$ matrizes de chaveamento são utilizadas



  - Nessa rede, existe apenas um caminho possível entre uma entrada e uma saída
  - Sendo assim, a escolha do caminho é muito eficiente e pode ser feita de forma descentralizada
  - Porém, por causa dessa falta de redundância, a rede é bloqueante

3.5.3 Roteamento de Mensagens

- É muito comum que na construção de máquinas paralelas, por motivos de custo, sejam utilizadas redes de interconexão que não possuem ligações diretas entre todos os componentes de um sistema

- Sendo assim, uma mensagem, para chegar ao seu destino, pode precisar trafegar por nós intermediários
- É dado o nome de **roteamento** ao procedimento de condução de uma mensagem, através de nós intermediários, até seu destino
- Todos os nós envolvidos nessa condução participam do roteamento, identificando se a mensagem já chegou ao seu destino e, se não for o caso, reenviando-a para um próximo nó
- Fora o nó destino, todos os outros nós envolvidos nesse procedimento decidem o caminho seguido pela mensagem através da rede de interconexão e são chamados de nós roteadores
- Existem duas formas básicas de conduzir uma mensagem ao seu nó destino:
  - Chaveamento de circuito
  - Chaveamento de pacotes
- Nas redes de telecomunicações, é tradicionalmente usado o **chaveamento de circuito** (*circuit switching*), pelo qual, inicialmente, é estabelecido um caminho fixo da origem ao destino, e só depois são enviadas todas as mensagens
  - Esse estabelecimento de conexão tem naturalmente um custo associado, que, no caso das telecomunicações, é pequeno em relação à duração da chamada
  - Essa forma de roteamento é usada por poucas máquinas paralelas, pois a comunicação entre dois nós, nesses casos, tem pouca duração (mensagens pequenas)
  - Sendo assim, o estabelecimento da conexão teria uma representatividade significativa no tempo total de comunicação, e a reserva de circuitos na rede para poucas mensagens subutilizaria os canais reservados e poderia ainda atrasar o estabelecimento de outras conexões
- Mais comum em máquinas paralelas é o **chaveamento de pacotes** (*packet switching*) onde não existe caminho pré-definido, mas cada mensagem decide, a cada nó, qual a direção que irá seguir na rede
  - Isso elimina o custo inicial de estabelecimento de circuito, mas embute um custo adicional para o roteamento de cada mensagem em cada um dos nós visitados
  - Duas grandes vantagens do chaveamento de pacotes que tornam esse tipo de roteamento atrativo para máquinas paralelas são a **inexistência de uma reserva de canais** da rede para uma única operação de comunicação e o **estabelecimento dinâmico do caminho** a ser seguido por uma mensagem
  - O estabelecimento dinâmico do caminho, por sua vez, pode permitir que os algoritmos de roteamento reajam mais rapidamente a congestionamentos e falhas na rede de interconexão, optando por caminhos alternativos

3.6 Coerência de Cache (*cache coherence*)

- É muito comum que as máquinas paralelas atuais sejam construídas com processadores produzidos em larga escala com o objetivo de reduzir os custos de projeto
- Dessa fora as máquinas paralelas com múltiplos processadores acabam por incorporar *caches* em suas arquiteturas

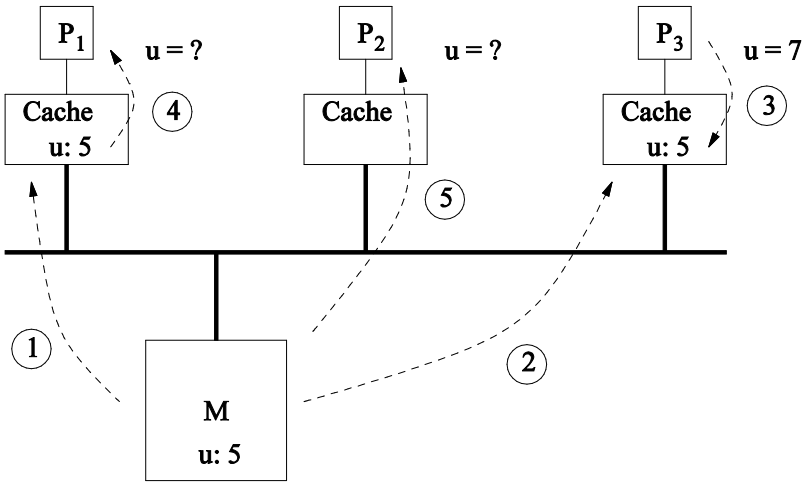
- Porém a presença de *caches* privadas em multiprocessadores necessariamente introduz problemas de **coerência de *cache*** (*cache coherence* [Hwang 93,98])
- Como múltiplas cópias de uma mesma posição de memória podem vir a existir em *caches* diferentes ao mesmo tempo, cada processador pode atualizar sua cópia local, não se preocupando com a existência de outras cópias da mesma posição em outros processadores
- Se essas cópias existirem, cópias de um mesmo endereço de memória poderão possuir valores diferentes, o que caracteriza uma situação de inconsistência de dados
- Outra alternativa para eliminar completamente o problema seria não permitir que dados compartilhados para operações de escrita sejam colocados nas *caches* do sistema
  - Nesse caso, somente instruções e dados privados são *cacheable* (passíveis de serem colocados na *cache*), e dados compartilhados são considerados *noncacheable*
  - O compilador fica responsável pela colocação da respectiva etiqueta (*tag*) nos dados para que os mecanismos de cópia de dados entre os níveis da hierarquia de memória estejam cientes de quais os dados que devem ser repassados diretamente ao processador sem passar pelas *caches* privadas
- Em multicomputadores, esse problema não ocorre, já que cada nó possui uma hierarquia de memória inteira associada à sua memória local, e não existe um espaço de endereçamento global compartilhado por todos os nós
- DEFINIÇÃO:
  - Sendo assim, uma arquitetura multiprocessada com *caches* privadas em cada processador **é coerente** se e somente se uma leitura de uma posição *x* de memória efetuada por qualquer processador *i* retorne o valor mais recente desse endereço
  - Ou seja, toda vez que uma escrita for efetuada por um processador *i* em um endereço de memória *x*, tem que ser garantido que todas as leituras subseqüentes de *x*, independentemente do processador, forneçam o novo conteúdo de *x*

3.6.1 O problema da inconsistência de dados

- É imprescindível que o resultado de um programa composto por múltiplos processos não seja diferente quando o programa roda em vários processadores, do que quando roda em apenas um, utilizando-se de multiprogramação
- Porém, quando dois processos compartilham a mesma memória através de *caches* diferentes, existe o risco de que uma posição de memória, em um determinado momento, não possua mais o valor mais recente desse dado
- As causas mais comuns desse tipo de inconsistência de dados serão detalhadas abaixo e são:
  - Inconsistência no compartilhamento de dados
  - Inconsistência na migração de processos
  - Inconsistência de E/S

Inconsistência no compartilhamento de dados:

- Os processadores P_1, P_2 e P_3 possuem *caches* privadas e estão interconectados a uma memória principal compartilhada através de um barramento. Eles efetuam uma seqüência de acessos à posição de memória *u*
- Inicialmente, P_1 tenta ler *u* de sua *cache*. Como *u* não está presente, o dado é lido da memória principal e copiado para sua *cache*. Na seqüência, P_3 faz o mesmo, gerando também uma cópia de *u* em sua *cache*. Então P_3 efetua uma escrita em *u* e altera o conteúdo da posição de 5 para 7
- Quando P_1 efetuar uma leitura de *u* novamente, ele recebe o conteúdo da posição *u* que se encontra em sua *cache*, ou seja, 5, e não o valor mais recente de *u,* que é 7



- Vale destacar que esse problema ocorre independentemente da política de atualização da memória principal utilizada pela *cachê*
- No caso de uma política de ***write-through*** (escrever através), em que, a cada escrita na *cache*, a posição da memória principal é atualizada também, a alteração de P_3 teria sido repassada também à memória principal, o que não impediria P_1 de ler o valor menos recente de *u* de sua *cachê*
- Em se tratando da política ***write-back*** (escrever de volta), a situação seria ainda pior, pois a alteração de P_3 teria apenas marcado o bloco de *u* na *cache* como sujo (através de seu *dirty-bit*), e a memória principal não seria atualizada imediatamente. Somente em uma eventual substituição desse bloco na *cache* é que a memória principal seria atualizada. Se P_2 efetuasse uma leitura em *u* nesse meio-tempo, copiaria para sua *cache* o valor menos recente de *u* (valor 5)

Inconsistência na migração de processos:

- Quando um processo perde o processador por causa de uma operação de E/S, não existe qualquer garantia de que ele vá retornar sua execução no mesmo processador de uma máquina multiprocessada

- O escalonador do sistema associará esse processo a um dos processadores livres segundo sua política de escalonamento quando a operação de E/S estiver concluída
- Isso faz com que, muitas vezes, o processo volte a executar em outro processador perdendo as informações de sua antiga *cache*

Inconsistência de E/S

- Problemas de inconsistência de dados também podem ocorrer durante operações de E/S que façam acesso direto à memória principal (*DMA - Direct Memory Acces* [Weber 2000])
- Operações de E/S desse tipo têm como origem ou destino à memória principal, e não se preocupam se os dados em questão estão sendo compartilhados por vários processadores, com possíveis cópias em diferentes *caches* privadas
- Sendo assim, quando uma controladora de E/S carrega um dado *x'* (atualizando o valor *x* que tinha sido escrito na memória principal por *write-through*) em uma posição da memória principal, ocorre inconsistência de dados entre *x'* e as cópias dessa posição de memória nas *caches* privadas dos processadores P_1 e P_2 que possuem o antigo valor *x*
- Quando um valor é lido da memória principal em uma operação de E/S, e as *caches* atualizam essa memória com a política de *write-back*, também pode ocorrer inconsistência de dados. Basta, para isso, que o valor atual das cópias dessa posição de memória nas *caches* privadas dos processadores P_1 e P_2 ainda não tenha sido substituído e, portanto, ainda não tenha sido copiado para a memória principal

3.6.2 Estratégias de coerência de *cache*

- Como vimos até agora, o problema da coerência de *cache* resume-se ao fato de que, em um determinado momento, possam existir simultaneamente múltiplas cópias de um mesmo dado da memória principal **o qual pode ser alterado localmente sem que se faça algo em relação às outras cópias** (o que geraria inconsistência de dados)
- Ou seja, o problema de coerência de *cache* seria resolvido se essa inconsistência de dados fosse eliminada. Isso pode ser obtido através de duas estratégias básicas:
  - Uma operação de escrita em uma posição da *cache* resulta na **atualização** de outras cópias desse mesmo dado em outras *caches* do sistema (*write-update*)
  - Uma operação de escrita em uma posição da *cache* resulta na **invalidação** de outras cópias desse mesmo dado em outras *caches* do sistema (*write-invalidate*)
- As duas estratégias resolvem o problema, impedindo que sejam geradas múltiplas cópias diferentes da mesma posição de memória
- A estratégia de **invalidação** tem um custo menor, mas resulta em uma maior latência de acesso caso as cópias invalidadas (eliminadas da *cache*) sejam novamente acessadas, o que resultaria em uma busca na memória principal
- A **atualização** das cópias tem naturalmente um custo mais alto, especialmente em máquinas com muitos processadores (e muitas cópias potenciais de um mesmo

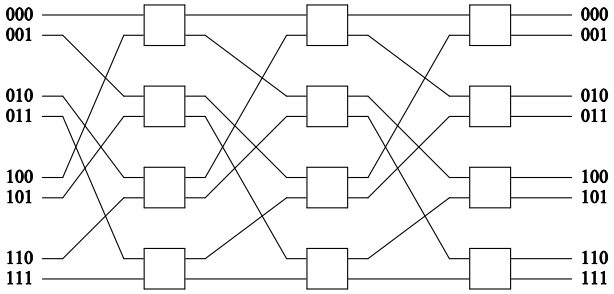
endereço), mas faz com que um novo acesso a essas cópias seja resolvido em nível de *cache* (menor latência)

- A questão de qual das estratégias acima resulta em melhor desempenho para o sistema como um todo está diretamente ligada ao padrão de compartilhamento da carga de trabalho, ou seja, dos programas que executam na máquina:
  - Se os processadores que estavam usando as cópias antes de serem atualizadas fizerem novos acessos a esses dados, o custo das **atualizações** vai ter valido a pena
  - Se os processadores não utilizarem esses dados novamente, o tráfego gerado pelas atualizações só onerou a rede de interconexão e não teve utilidade. Nesse segundo caso, a **invalidação** eliminaria as cópias antigas e acabaria com uma situação de compartilhamento aparente
- Um fenômeno que onera bastante a estratégia de atualizações é denominado *"pack rat"*. Ele é uma conseqüência da migração de processos executada pelo sistema operacional durante a multiprogramação. Quando um processo perde o processador por causa de uma operação de E/S, ele pode voltar a executar em outro processador, dependendo da disponibilidade dos recursos do sistema naquele momento e da política empregada pelo escalonador do sistema operacional. Nesse caso, além de não poder acessar mais os dados de sua antiga *cache* e ter que buscá-los novamente da memória principal, os dados antigos continuam sendo atualizados em vão até que sejam eliminados da *cache* antiga por alguma política de substituição de blocos (como, por exemplo, *LRU - Least Recent Used* [Stallings 1998], que elimina da *cache* os blocos menos recentemente usados).
- Como é muito fácil construir casos nos quais uma determinada estratégia vai desempenhar melhor que a outra e vice-versa, alguns autores propõem como alternativa que as duas estratégias sejam implementadas em hardware e que o sistema permita a troca entre atualização e invalidação dinamicamente em tempo de execução
- O momento da troca pode ser:
  - Determinado pelo programador através de uma chamada de sistema
  - Seguir uma certa probabilidade alterável na configuração do sistema
  - Depender do padrão de acesso observado em tempo de execução

4 EXERCÍCIOS

1. Defina processamento paralelo.
2. Como a multiprogramação acelera a execução de um grupo de programas mesmo em uma máquina com apenas um processador?
3. Desenhe o comportamento do fator de aceleração (*Speed-Up*) real e do fator de aceleração ideal para uma aplicação paralela. Por que existe diferença entre estas linhas? Por que esta diferença com o aumento do número de processadores?
4. Explique o conceito de pipeline superescalar. Qual o seu CPI teórico. Indique dois problemas ligados a este conceito.
5. Quais são as duas possibilidades para se construir máquinas com vários processadores? Que nome é dado a cada uma delas?

6. Classifique um servidor com 2 processadores Pentium III (máquina dual) nas três classificações vistas em aula.
7. Por que alguns autores consideram uma arquitetura COMA como sub-classificação de arquiteturas NUMA?
8. Desenhe as arquiteturas NORMA e NUMA. Os desenhos são muito parecidos. Onde esta a diferença significativa entre elas?
9. De um exemplo de máquina para a classe MISD de Flynn.
10. O que significa bloqueante, no contexto de redes de interconexão?
11. Mostre que a rede abaixo é bloqueante fixando um caminho e mostrando outro que não podem ser utilizado ao mesmo tempo (considerando chaveamento de circuito).



12. Como uma rede *crossbar* pode ser usada para a construção de um multicomputador (desenhe).
13. Desenhe uma rede estática para interligar 7 processadores que possuam no máximo grau 4.
14. O que acontece se 2 processadores de um multiprocessador escreverem ao mesmo tempo na mesma posição de memória?