Arquiteturas Paralelas: memória compartilhada e distribuída em máquinas MIMD

Paulo Sérgio Lopes de Souza
pssouza@icmc.usp.br

Universidade de São Paulo / ICMC / SSC – São Carlos
Laboratório de Sistemas Distribuídos e Programação Concorrente

Semântica de Memórias Compartilhadas

- Diferentes módulos de memória são usuais com um mesmo endereçamento
- Redes de conexão mais complexas permitem concorrência de R/W
- Entrega de requisições podem ocorrer em ordens diferentes da esperada
- Cópias de blocos de memória podem estar em diferentes caches
- Modelos de consistência de memória
  - Ditam regras entre hardware e software para ordem de R/W nos módulos
  - Determinam quando *writes* “alcançam” R/W de outras variáveis
- Alguns modelos de consistência de memória:
  - Estrita
    - escritas são visíveis instantaneamente a todos os processos
  - Sequencial
    - processos veem a mesma ordem de requisições de acesso à memória
  - (de) Processador
    - escritas de um processo são observadas por ele na mesma ordem
  - Fraca
    - sincronizações explícitas são consistentes sequencialmente
  - (de) Liberação
    - sincronizações explícitas são consistentes ao processador em relação aos demais

Semântica de Memórias Compartilhadas

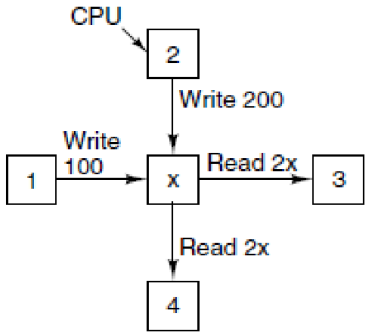
- **Consistência Estrita (ou Restrita)**
  - Garante que a **leitura sempre obtém última escrita instantaneamente**
    - Assume *clock* global e que não há atraso em mensagens no hardware
      - O que é impossível
    - Futuros R/W só ocorrerão depois que um *write* prévio finalizar

  - Possível quando:
    - Só um módulo de memória
    - Critérios FIFO
    - Sem caches
    - Nem duplicações

  - Gera um grande gargalo para o desempenho em sistemas reais
    - Impede otimizações no hardware e no compilador

Semântica de Memórias Compartilhadas

- **Consistência Sequencial**
  - Hardware intercala requisições de R/W em uma ordem não determinística
  - Todas as CPUs veem a mesma ordem para manter consistência
    - Ainda há a visão de um clock global, mas agora a ordem pode variar
    - Assume que a memória é atômica (uma única e grande posição)
  - No exemplo:
    - CPUs 1 e 2 fazem, cada uma, uma escrita em x
    - CPUs 3 e 4 fazem, cada uma, duas leituras de x, uma imediatamente após a outra
    - Diferentes ordens de execução são possíveis



(b)	(c)	(d)
W100	W100	W200
W200	R3 = 100	R4 = 200
R3 = 200	W200	W100
R3 = 200	R4 = 200	R3 = 100
R4 = 200	R3 = 200	R4 = 100
R4 = 200	R4 = 200	R3 = 100

(a)

Figure 8-24. (a) Two CPUs writing and two CPUs reading a common memory word. (b)–(d) Three possible ways the two writes and four reads might be interleaved in time.

Tanenbaum (2013)

Semântica de Memórias Compartilhadas

- **Consistência de Processador (*processor consistency*)**
  - Menos rigorosa e mais fácil de implementar em mais CPUs
  - Não garante que todas as CPUs veem a mesma ordem
    - Diferentemente da consistência sequencial

  - Propriedades
    - Escritas de uma CPU são vistas nas outras na ordem da emissão
      - writes 1A,1B,1C são vistos em outros processadores nesta ordem
    - Para cada palavra de memória, todas as CPUs veem todas as escritas para essa palavra na mesma ordem
      - CPUs concordam qual é a última escrita

  - Exemplo:
    - duas CPUs (1 e 2) emitem as escritas - 1A, 1B, 1C, 2A, 2B e 2C para a mesma palavra de memória
      - 1B não poderá vir antes de 1A, mas estas ordens são possíveis:
        - 1A, 1B, 2A, 2B, 1C, 2C ou 2A, 1A, 2B, 1B, 1C

Semântica de Memórias Compartilhadas

- **Consistência Fraca (*weak consistency*)**
  - Não garante nem a ordem de escrita de uma CPU
    - Uma CPU vê 1A antes de 1B e outra CPU pode ver 1B antes de 1A
  - Variáveis de sincronização + operação de sincronização explícita
    - Permitem que escritas pendentes finalizem e nenhuma inicie até todas as escritas antigas terminem, incluindo a operação de sincronização
  - Sincronização descarrega pipeline
    - Coloca memória em um estado estável, sem operações pendentes
    - Operações de sincronização são consistentes entre si
      - CPUs enxergam sincronizações na mesma ordem

Semântica de Memórias Compartilhadas

- **Consistência Fraca (*weak consistency*)**
  - Insere uma sincronização explícita entre acessos
  - No exemplo abaixo, a ordem das escritas 1A e 1B não é garantida; mas 1C ocorre depois
- Software impõe uma ordem na sequência de eventos,
  - Há um atraso no processamento pelo esvaziamento do pipeline
    - Sincronizações não são frequentes por questões de desempenho

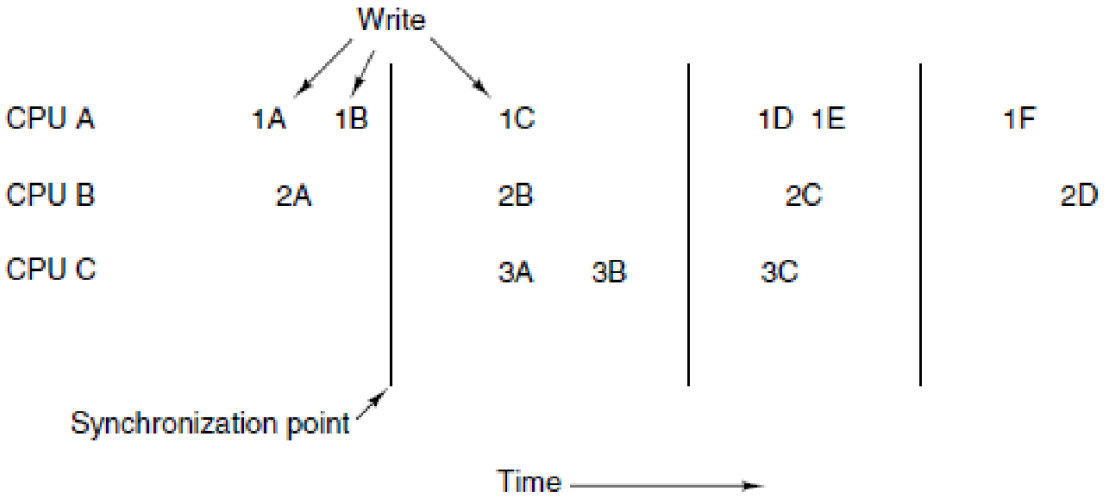


Figure 8-25. Weakly consistent memory uses synchronization operations to divide time into sequential epochs.

Tanenbaum (2013)

Semântica de Memórias Compartilhadas

- **Consistência de Liberação (*release consistency*)**
  - Tenta melhorar ineficiência da Consistência Fraca
    - Operações de sincronização são divididas em *acquire* e *release*
    - *Semelhantes a um lock/unlock. Garante Consistência de Processador*
  - Foco da proposta:
    - Quando um processo deixa a Região Crítica, não é necessário forçar que todas as escritas finalizem, basta garantir que elas terminem antes de outros processos entrarem na RC
  - *Acquire*
    - Obtém acesso exclusivo aos dados compartilhados para poder escrever
    - Só obtém acesso garantindo exclusão mútua
  - *Release*
    - Marca a saída da RC
      - Não força fim das escritas, mas só finaliza de fato quando elas terminarem
    - Novas operações na memória podem iniciar imediatamente
      - Novo *acquire* só prossegue se *release* prévio terminou

Coerência de Cache com Memória Compartilhada

- Caches são muito usadas
  - Reduzem demanda sobre rede de interconexão e às memórias mais lentas

- Replicam dados da memória em diferentes níveis de cache
- Diferentes CPUs/Núcleos replicam os níveis de cache

- Dados replicados nas caches podem ficar desatualizados em Writes

- Coerência de cache determina:
  - regras para a propagação da atualização dos dados
  - requisitos esperados de R/W para uma posição de memória

- Determinam a propagação de *Writes*; mas não quando os *Writes* ocorrerão
  - O “quando” é determinado pelos modelos de consistência de memória

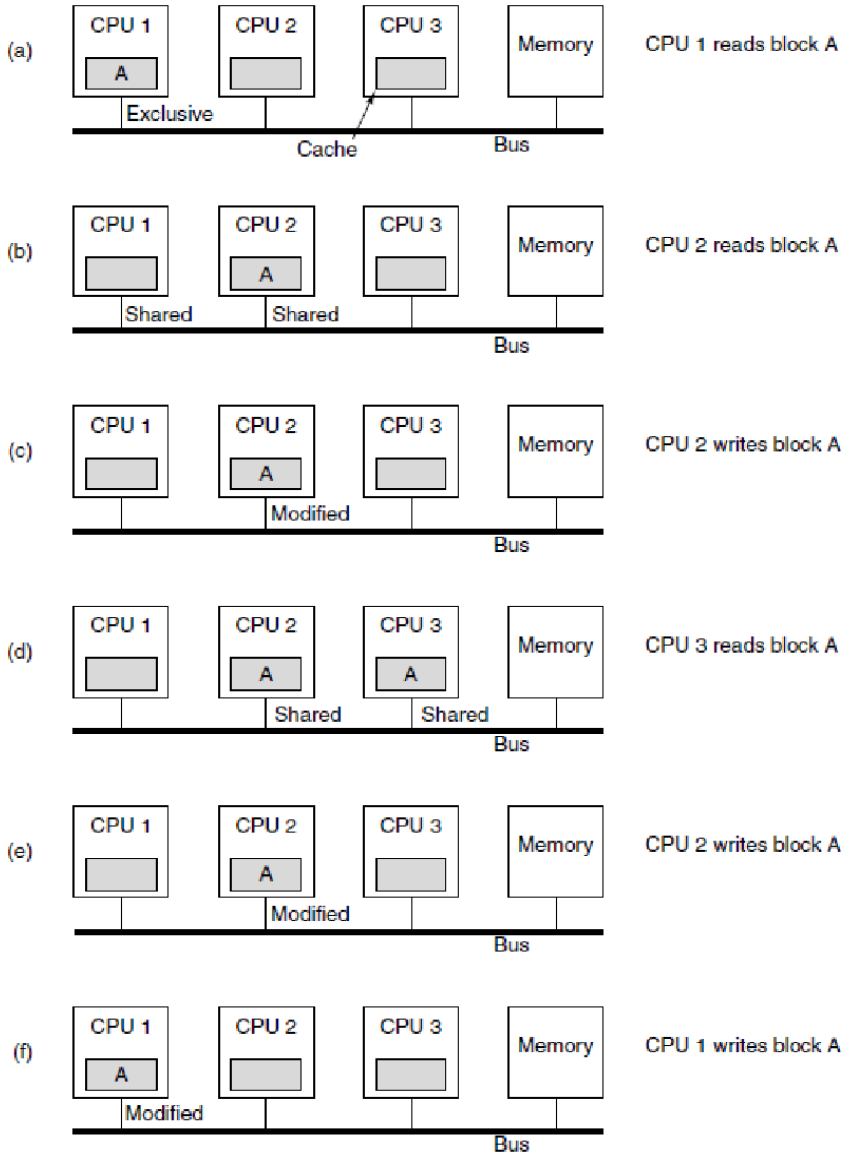
Coerência de Cache com Memória Compartilhada

- Soluções por software e hardware
  - Software: deixam hardware mais simples e são mais conservativas (lentas), como p.ex., evitando o uso de cache
  - Hardware: protocolos de coerência de cache
    - Protocolos *update, invalidate* ou *adaptive* (que usa ambos)

- Protocolo *update*
  - Escritas no bloco de uma cache atualizam este bloco em outras caches
  - Pode gerar sobrecarrega na memória e nos barramentos
- Protocolo *invalidate*
  - Escritas no bloco de uma cache invalidam este bloco em outras caches
  - Protocolo MESI (*Modified, Exclusive, Shared, Invalid*)

Coerência de Cache com Memória Compartilhada

- **Protocolo MESI**
  Considerando um barramento simples



Tanenbaum (2013)

Figure 8-28. The MESI cache coherence protocol.

Coerência de Cache com Memória Compartilhada

- Protocolo MESI

Table 17.1 MESI Cache Line States

	M Modified	E Exclusive	S Shared	I Invalid
This cache line valid?	Yes	Yes	Yes	No
The memory copy is ...	out of date	valid	valid	—
Copies exist in other caches?	No	No	Maybe	Maybe
A write to this line ...	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus

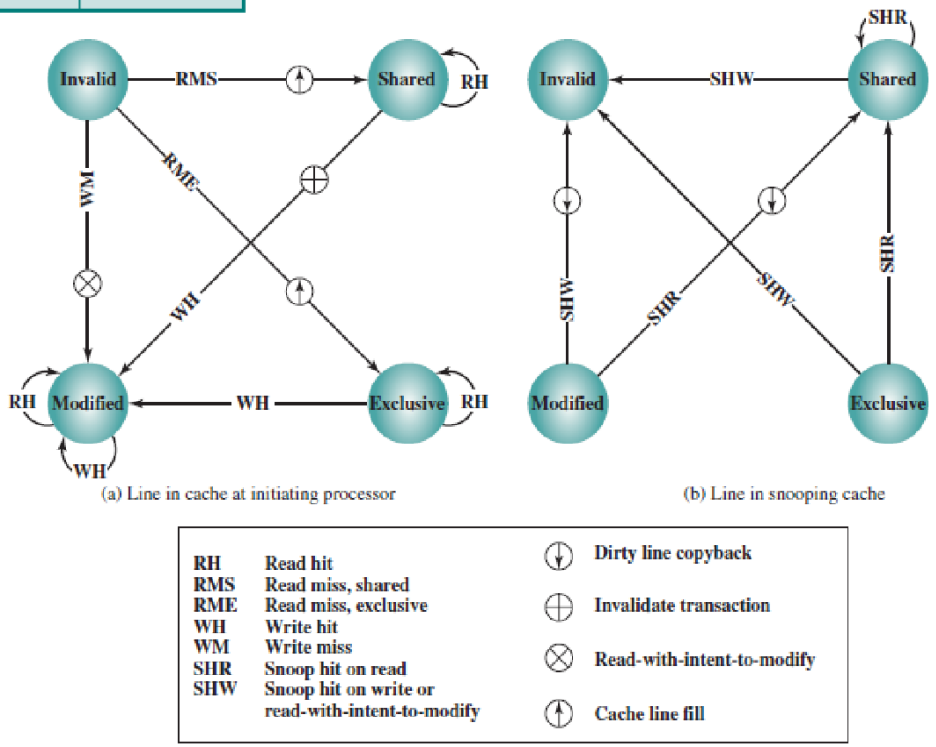


Figure 17.6 MESI State Transition Diagram

Stallings (2013)

Coerência de Cache com Memória Compartilhada

- Duas categorias de soluções por hardware
  - *Snoopy* e Diretório

  - **Protocolo *Snoopy*** (monitoração)
    - Voltados às redes de conexão em barramento, máquinas simétricas
      - Veja o slide sobre pro
    - Controladores monitoram o barramento
      - Permitem na gestão dos estados dos protocolos de coerência

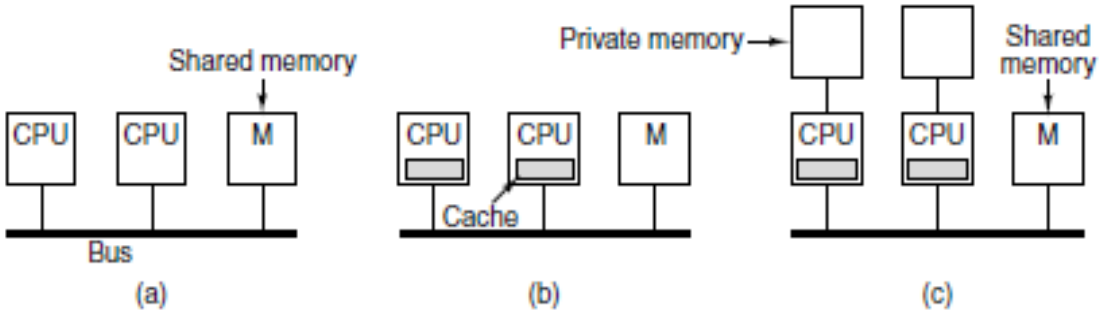
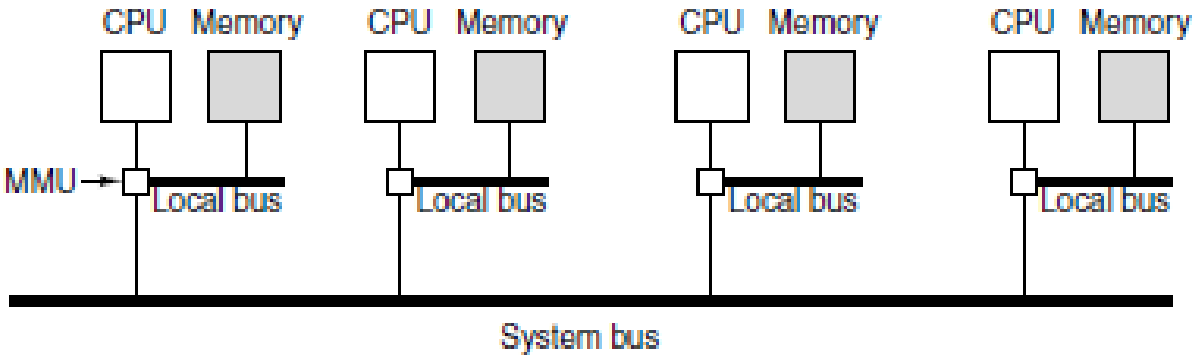


Figure 8-26. Three bus-based multiprocessors. (a) Without caching. (b) With caching. (c) With caching and private memories.

Tanenbaum (2013)

Coerência de Cache com Memória Compartilhada

- **Protocolo de Diretório** (máquinas assimétricas)
  - Coleta e mantém informações sobre as cópias dos blocos nas CPUs
    - Estruturas em hardware chamadas de diretórios
      - Centralizadas ou distribuídas
  - Quando controladores das caches requisitam operações
    - Diretórios verificam e emitem sinais para a transferência de dados
      - Entre memória e cache e entre caches se necessário
    - Propagam ops de coerência às CPUs com blocos compartilhados
  - Comunicações extra representam um gargalo ao desempenho
    - Comuns em redes complexas, não apenas em um barramento

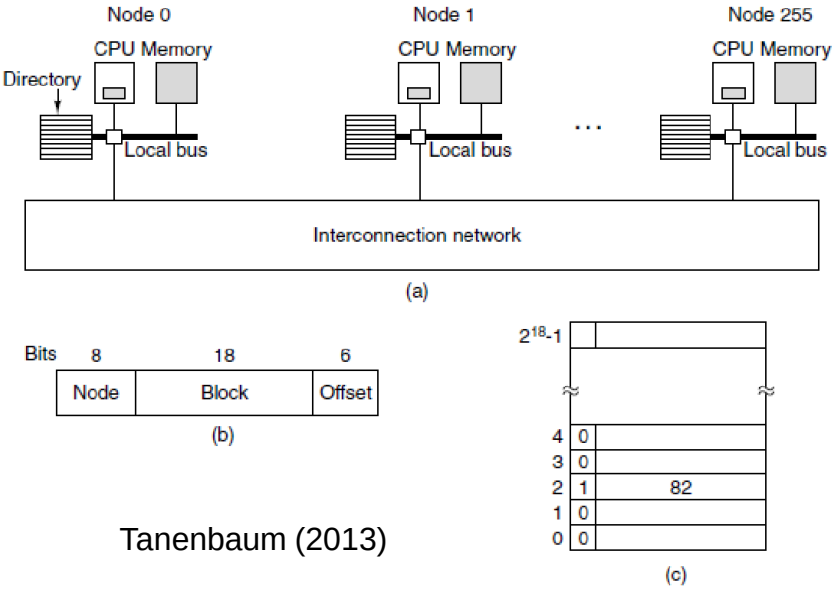


Tanenbaum (2013)

Coerência de Cache com Memória Compartilhada

- Exemplo Protocolo de Diretório
  - 256 nós (2^8 nós), cada um com 16MB (2^24B) de RAM
  - RAM tem ao todo 2^32 Bytes, organizada em blocos de 64Bytes (2^6 Bytes)
    - Há 2^26 blocos (ou linhas) na RAM
  - Nós têm as posições de memória de acordo com sua numeração
    - Nó 0 tem 0-16MB, nó 1 tem 16-32MB, ...
  - Caches têm 2^24 Bytes (2^18 blocos, cada um com 2^6 Bytes)
    - Nós têm as entradas de diretório para os 2^18 blocos de cache
  - Um bloco só pode estar em uma cache

  - Operação sobre o nó 20, end:
    - 0x24000108
      - nó 36, bloco 04, offset 08
    - MMU nó 20 requisita ao 36 a op
    - Nó 36 procura bloco 04 na cache
      - não encontra e acessa RAM
    - Bloco 04 é enviado ao nó 20
      - Dir do nó 36 registra end 20

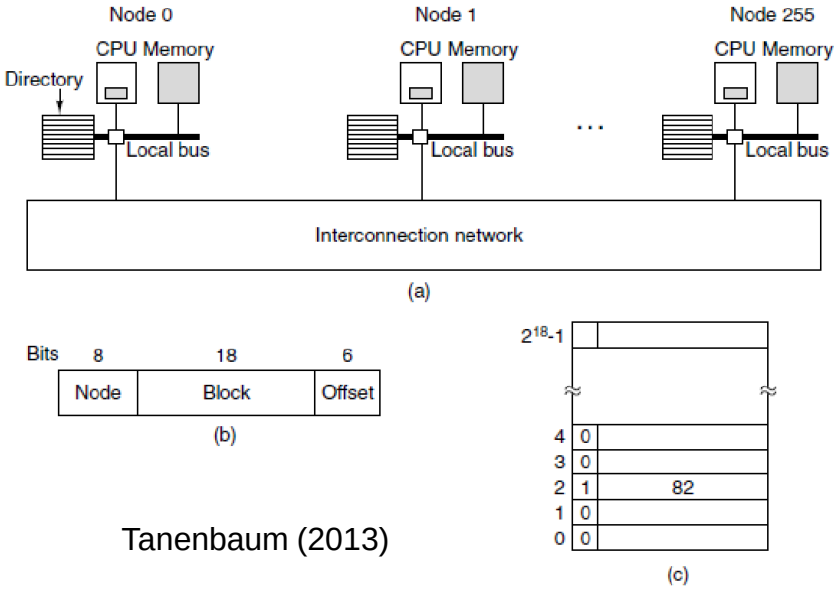


Tanenbaum (2013)

Figure 8-33. (a) A 256-node directory-based multiprocessor. (b) Division of a 32-bit memory address into fields. (c) The directory at node 36.

Coerência de Cache com Memória Compartilhada

- Exemplo Protocolo de Diretório (continuação)
  - Nova requisição do nó 20 solicita outro bloco ao nó 36
    - Esta deveria estar na posição 02 do diretório
  - Diretório indica que o bloco deve ser obtido no nó 82
    - Bloco é enviado ao nó 20. Nr 20 é registrado no diretório do nó 36
      - Bloco do nó 82 é invalidado
  - Sobrecarga para manter o diretório:
    - RAM de cada nó tem 2^27 bits (16 MBytes)
    - Dir tem 2^18 entradas com 9 bits cada
      - 8 bits para ident 256 nós
      - 1 bit para (in)validar
    - (9 * 2^18 bits) / (2^27 bits) ∴
    - 9 * 2^-9 ∴
    - 9 * ((1/2)^9) ∴
    - 9 / 512 ∴
    - 0,01757 ∴
    - 1,76%

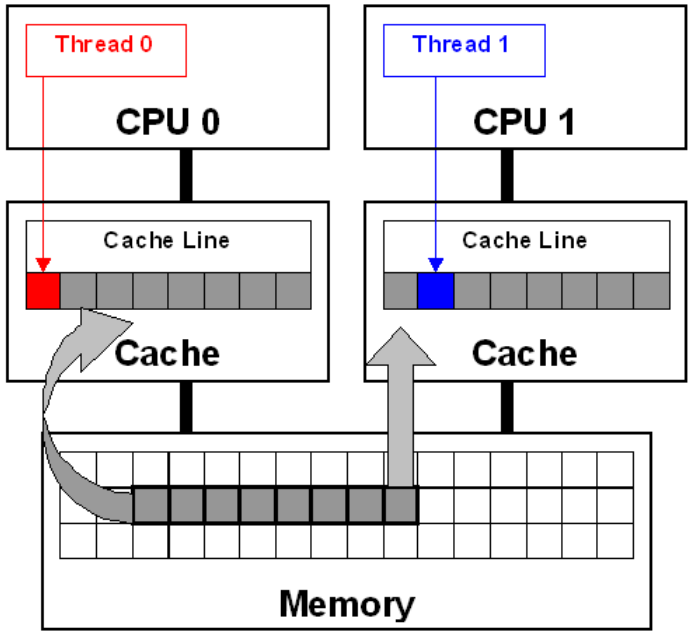


Tanenbaum (2013)

Figure 8-33. (a) A 256-node directory-based multiprocessor. (b) Division of a 32-bit memory address into fields. (c) The directory at node 36.

Coerência de Cache com Memória Compartilhada

- **Falso Compartilhamento**
  - Ocorre quando dados diferentes de um bloco compartilhado são atualizados pelos processadores em suas caches
  - O uso de dados dispostos sequencialmente na memória RAM, por processadores distintos, acarreta em falso compartilhamento quando há escritas e leituras em tais blocos.
  - Bloco fica inválido, mas de fato não precisaria ficar
    - Os dados modificados são diferentes dentro do bloco
  - Unidade de transferência por bloco acaba por invalidar o bloco todo
    - Ocorrendo um perda de desempenho
  - Solução
    - manter maior distância entre dados espalhados nas caches



Fonte WEB

Memória Distribuída

- Máquinas (ou nós) têm suas memórias locais
  - Não há compartilhamento de um espaço de endereçamento entre nós
  - Dentro de um nó pode existir compartilhamento de memória

- Processos podem ter seus próprios espaços de endereçamento
  - Tanto em máquinas distintas quanto no mesmo nó

- Possuem uma escalabilidade bem maior em função da arquitetura
  - Memórias locais funcionam em paralelo e independentemente
  - Trocas de mensagens permitem a comunicação e sincronização

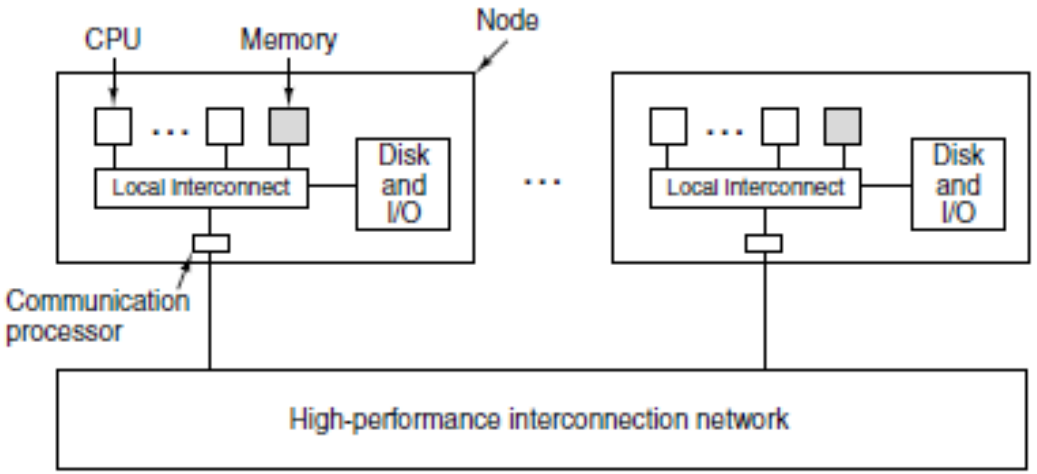


Figure 8-36. A generic multicomputer.

Tanenbaum (2013)

Referências

Stallings, W.; Computer Organization and Architecture: Designing for Performance. Ninth Edition. Pearson. 2013.

Tanenbaum, A. S.; Austin, T.; Structured Computer Organization. Sixth Edition. Pearson. 2013.

Patterson, D. A.; Hennessy, J. L.; Computer Organization and Design: the hardware / software interface. Fith Edition. Elsevier, 2014.

Rauber, T.; Rünger, G.; Prallel Programming for Multicore and Cluster Systems. Second Edition. Springer. 2013.

Material Complementar

Material sobre Consistência de Memória e Coerência de Cache

https://www.cs.utexas.edu/~bornholt/post/memory-models.html

https://scs.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=694972dd-dac1-4635-8939-b6abe3f99749

https://www.cs.colostate.edu/~cs551/CourseNotes/Consistency/TypesConsistency.html

Arquiteturas Paralelas: memória compartilhada e distribuída em máquinas MIMD

Paulo Sérgio Lopes de Souza
pssouza@icmc.usp.br

Universidade de São Paulo / ICMC / SSC – São Carlos
Laboratório de Sistemas Distribuídos e Programação Concorrente

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico
CAPES
FAPESP