DLE – Um Protocolo Eficiente Para Coerência de Cache Baseado em Diretórios Limitados

Antônio Carlos Fontes Atta* (atta@dcc.unicamp.br)
Célio Cardoso Guimarães† (celio@dcc.unicamp.br)

DCC - IMECC - UNICAMP
CxP 6065 - CEP 13081-970 - Campinas - SP

Resumo

Protocolos baseados em diretórios têm sido propostos em resposta à busca por soluções eficientes para o problema da coerência de cache em grandes sistemas multiprocessados de memória compartilhada. Uma classe desses protocolos emprega os chamados diretórios limitados. Apesar de serem razoavelmente econômicos em termos de espaço ocupado e demandarem lógica de controle simples, os protocolos baseados em diretórios limitados exibem um desempenho global inferior ao de outras soluções. Isto se deve, principalmente, ao comportamento diversificado que as aplicações paralelas apresentam com relação ao número de processadores que compartilham simultaneamente um mesmo bloco.

Neste artigo, propomos um protocolo baseado em diretórios limitados com invalidação por difusão (*broadcasting*) que filtra os efeitos negativos causados por graus elevados de compartilhamento, tornando os diretórios limitados mais atrativos e eficientes. As características e requisitos do protocolo são discutidos, na tentativa de demonstrar sua viabilidade prática. Adicionalmente, a utilização de barramentos é revista, com indicações de que, se usados de forma especializada, esses dispositivos podem ser bastante úteis nas futuras gerações de sistemas multiprocessados.

Abstract

Directory-based protocols have been proposed as an efficient means of implementing cache consistency in large-scale shared-memory multiprocessors. One class of these protocols is known as "limited directories". Despite of their space economy and simple control logic, limited directories based protocols show inferior performance when compared to others schemes. This fact is mainly due to the diversified behavior of parallel applications with regard to the number of processors that simultaneously share a given block of data.

In this paper, we propose a limited directory based protocol with invalidation broadcasting that filters the negative effects caused by data shared by a large number of processors, turning the limited directories more attractive and efficient. The properties and requisites of the protocol are discussed in order to indicate its viability. Additionaly, the use of the bus is reviewed, indicating that, if applied in specialized ways, these devices can be quite useful in the next generations of multiprocessors.

*Bacharel em Ciência da Computação pela UFBa; Mestrando em Ciência da Computação na UNICAMP; Área de Interesse: Processamento Paralelo com ênfase em Sistemas Multiprocessados de Memória Compartilhada.

†Professor Adjunto do Depto. de Ciência da Computação da UNICAMP; Ph.D. em Ciência da Computação pela Case Western Reserve University; Áreas de Interesse: Processamento Paralelo e Sistemas Operacionais Distribuídos.

1 Introdução

A exemplo dos sistemas monoprocessados, memórias cache têm sido usadas nos sistemas multiprocessados de memória compartilhada como forma de se obter a redução da latência no acesso à memória principal. Essa capacidade, aliada à possibilidade de reduzir-se também os atrasos impostos pela rede de interconexão — decorrentes das características físicas e das disputas pelo seu uso — têm motivado a adoção, por parte dos projetistas de sistemas multiprocessados,[1] de configurações onde cada processador é dotado de um cache privado conforme o modelo da Figura 1. Essa decisão de projeto, entretanto, dá origem ao problema da consistência das informações distribuídas pelos caches, ou simplesmente, problema da coerência de cache [19]. Um dado está consistente se operações de leitura nesse dado retornam sempre o valor da última operação de escrita sobre ele [7].

A solução eficiente do problema da coerência de cache é hoje considerada um dos principais desafios para as futuras gerações de sistemas multiprocessados. Basicamente, existem soluções a nível de *software* e a nível de *hardware*. Neste trabalho apenas soluções em *hardware* serão consideradas.

A manutenção da coerência, a nível de *hardware*, é realizada em tempo de execução pelos controladores de memória e de cache, sendo diretamente influenciada pelas características da rede de interconexão usada no sistema.

Os protocolos para redes tipo barramento [15], em sua maioria, exploram a capacidade de difusão do barramento para detectar, através da monitoração constante de suas vias, operações que, por ventura, causem inconsistências. No caso de uma operação de escrita a um dado compartilhado, por exemplo, duas ações possíveis para se evitar a inconsistência são: enviar o dado modificado pelo barramento para que os outros caches possam atualizar suas respectivas cópias ou enviar um comando de consistência, também pelo barramento, requisitando que cada controlador de cache invalide a cópia local do bloco que contém o dado, caso ela exista. Como cada controlador de cache permanece eternamente "escutando" os sinais que trafegam no barramento, protocolos baseados nessa forma de operação são conhecidos como **snoop cache protocols**. A grande quantidade de protocolos eficientes para barramentos já propostas [13, 17, 11, 10], estudadas [3], e aplicadas [20] indica quão devidamente definida e entendida encontra-se essa classe de soluções. Redes tipo barramento, entretanto, apresentam uma grave limitação à expansão do sistema: sendo a via única de comunicação, ela atinge rapidamente o ponto de saturação à medida que cresce o número de processadores a ela conectados. Para sanar essa limitação, as redes de interconexão dos sistemas multiprocessados da próxima geração são constituídas de modo que o número de vias de comunicação cresce juntamente com o número de processadores instalados. Um exemplo desse tipo de rede é a rede de chaves multiestágios [12].

Nas redes multiestágios, a existência de diversas vias de comunicação inviabiliza o processo de difusão/detecção dos sinais que trafegam por ela, impedindo, por conseguinte, a aplicação dos protocolos *snoopy*. Outras alternativas de protocolos, portanto, têm sido propostas. A principal característica dessa nova classe de soluções é a existência de uma estrutura de dados que informa ao protocolo de coerência, de que forma as cópias dos blocos estão distribuídas

[1] Por questões de simplicidade, o termo "sistema multiprocessado" será sempre empregado em lugar de sistemas multiprocessados de memória compartilhada (sistemas com dois ou mais processadores independentes que compartilham um mesmo espaço global de endereçamento).

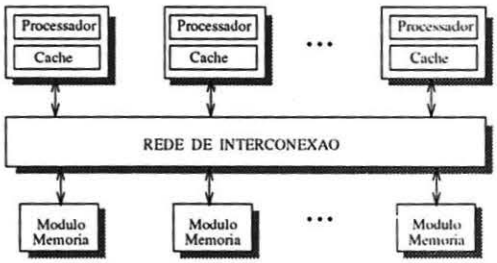

Figura 1: Exemplo de configuração com caches privados.

pelo sistema, de modo que, os comandos de consistência (invalidações, por exemplo) possam ser enviados diretamente através das ligações ponto a ponto disponíveis nas redes multiestágios. Essa estrutura de dados é conhecida como **diretório** [7]. Diretórios, por sua vez, também apresentam uma série de características que restringem a sua utilização.

Neste artigo, apresentamos uma proposta de protocolo baseada em diretórios. Inicialmente, na seção 2, faremos uma rápida revisão de algumas propostas de solução baseadas em diretórios; uma análise aprofundada dessas e de outras soluções baseadas em diretórios, além de soluções tipo *snoopy* pode ser vista em [5]. Em seguida, na seção 3, apresentamos o nosso protocolo, suas principais características e requisitos, juntamente com um discussão que mostra de que forma ele tenta sanar algumas das restrições das propostas anteriores. Finalmente, na seção 4, são feitas algumas considerações sobre o desempenho esperado do protocolo proposto e o estágio atual do trabalho é descrito.

2 Protocolos Baseados em Diretórios

2.1 Diretório Mapa de Bits

Nesse esquema, proposto por Censier e Feautrier [7], o diretório é armazenado na memória principal e é composto por entradas contendo vetores de *bits*. Existe uma entrada para cada bloco de memória. A dimensão dos vetores é igual ao número de caches do sistema, sendo que um elemento (*bit*) ligado no vetor indica a presença de uma cópia do bloco no cache indicado pela posição do elemento. Além do vetor, cada entrada possui ainda um *bit* de modificação. Entradas com vetores contendo diversos *bits* ligados indicam compartilhamento dos blocos associados; entradas com vetores contendo apenas um *bit* ligado e com o *bit* de modificação também ligado indicam que a versão mais recente do bloco encontra-se no cache correspondente ao *bit* ligado do vetor. Antes de permitir a execução de modificações a blocos compartilhados, o protocolo de coerência envia, a partir das informações fornecidas pelo diretório, sinais de invalidação do bloco a todos os caches envolvidos.

Uma vez que se tem acesso direto a uma entrada do diretório — através do endereço do bloco — e, a partir daí, aos caches com cópia do bloco — através das posições ligadas no vetor — é possível implementar-se protocolos de coerência bastante eficientes e o diretório mapa

de *bits* tem sido considerado como a melhor solução em termos de desempenho. A estrutura necessária à manutenção do mapa de *bits* (conjunto formado por todos os vetores de *bits*), contudo, torna essa solução cara em termos de espaço ocupado. Posto formalmente, devido ao fato do tamanho de uma entrada associada a cada bloco no diretório ser proporcional ao número de caches ($\Theta(N)$) e o tamanho do próprio diretório ser proporcional ao número de blocos na memória ($\Theta(M)$) e considerando, ainda, que a quantidade de memória global do sistema cresce juntamente com o número de processadores, o *overhead* de espaço ocupado pelo diretório é proporcional ao produto dessas duas grandezas, ou seja, $\Theta(MN)$ uma função quadrática. Além disso, o tamanho do vetor de *bits*, definido na fase de projeto, é um fator limitante do grau de expansão do sistema, já que, ele determina o número máximo de processadores que o sistema pode comportar.

2.2 Diretório Limitado

Uma solução proposta por Agarwal *et al.* [1] para reduzir o *overhead* de espaço imposto pela solução mapa de *bits* consiste em limitar à uma constante, o número máximo de cópias simultâneas em caches dos blocos de memória. Cada entrada do diretório continua associada a um bloco de memória, mas, no lugar do vetor de *bits*, tem-se um número limitado de apontadores que indicam os caches que compartilham o bloco. Diretórios estruturados dessa forma são conhecidos como diretórios limitados.

Dado que cada apontador na estrutura limitada requer $\log_2 N$ *bits* para armazenar o endereço de um processador em um sistema com N caches e considerando, mais uma vez, que a quantidade de memória cresce linearmente com o número de processadores, o tamanho de um diretório limitado cresce como $\Theta(N \log N)$, já que, é alocado um número pequeno e fixo de apontadores por entrada no diretório. Esse *overhead*, aproximadamente linear, é muito menos severo que o observado no esquema mapa de *bits* ($\Theta(N^2)$), tornando os diretórios limitados bastante atrativos para a utilização em protocolos de coerência de sistemas sujeitos a grandes expansões.

Além do *overhead* de espaço reduzido, o principal fator que motiva a utilização dos diretórios limitados em protocolos de coerência está associado ao comportamento peculiar que as aplicações paralelas apresentam com relação a dados compartilhados. Estatísticas obtidas a partir da sequência de referências feitas por algumas aplicações na fase de execução indicam que, normalmente, apenas uns poucos processadores fazem referência à determinadas posições antes delas serem modificadas [1], [14], [18]. É de se esperar, portanto, que entradas nos diretórios constituídas de poucos apontadores sejam capazes de, na maioria dos casos, comportar o endereço dos caches que possuem cópia de um bloco, antes do envio de sinais de invalidação.

As ações efetuadas quando um determinado cache deseja compartilhar um bloco cuja entrada já não possui mais apontadores disponíveis definem e diferenciam as diversas variações desse modelo básico de diretório limitado. Uma forma imediata de tratar essa falta de apontadores consiste em liberar um dos apontadores da entrada — invalidando-se a cópia do cache correspondente — para, em seguida, armazenar o endereço do novo cache, antes de permitir o acesso desse cache ao bloco. Uma segunda estratégia, consiste em permitir que tantos caches quantos queiram compartilhem um bloco, sendo que no caso de haver falta de apontadores, a indicação dessa ocorrência é armazenada no diretório e, no momento em que for necessário

invalidar o bloco, a invalidação é enviada a todos os caches do sistema, independentemente do fato deles possuírem ou não cópia do bloco. Essas duas estratégias são conhecidas como *não difusão* e *difusão* de invalidações respectivamente. Outros esquemas são propostos em [21] e [8].

3 Uma Proposta de Diretório Limitado Eficiente

A maior restrição para a utilização dos diretórios limitados está relacionada com o correto dimensionamento do número de apontadores. Os efeitos da má escolha do número de apontadores podem ser melhor observados através da análise das duas formas básicas de tratamento da falta de apontadores apresentadas na seção anterior. Nos protocolos que não usam a difusão de sinais de invalidação, o subdimensionamento provoca uma espécie de "dança" dos blocos, fenômeno que ocorre quando um bloco é frequentemente requisitado por um número de processadores maior que o número de apontadores disponíveis, o que ocasiona a execução de invalidações e transferências desnecessárias do bloco – um exemplo claro desse fenômeno está presente em boa parte das aplicações paralelas, onde, durante a fase de inicialização, um mesmo bloco pode vir a ser requisitado por todos os processadores do sistema. Nos protocolos que utilizam a difusão de invalidações, todos os processadores podem requisitar um bloco sem a ocorrência da "dança" do bloco. Em caso de necessidade da invalidação desse bloco, contudo, a rede de interconexão é quem "paga o preço"; por ser o único meio de comunicação entre os módulos do sistema, ela sofre um grande aumento em seu tráfego. Nos dois tipos de solução o superdimensionamento acarreta um desperdício de espaço, já que, os apontadores em excesso podem ficar sem utilização a maior parte do tempo.

Chaiken [9] mostra que modificando o código da aplicação paralela utilizando técnicas que eliminam a existência de variáveis com alto grau de compartilhamento ou que são muito referenciadas (usando árvores de combinação [16], por exemplo, na confecção de primitivas de sincronização), é possível obter-se um desempenho dos diretórios limitados compatível com os diretórios *mapa de bits*. Essa estratégia, todavia, requer um grande esforço por parte do programador na identificação dessas variáveis, na escolha da técnica correta para cada variável e na verificação de que a aplicação se tornou realmente mais eficiente após todas as modificações. Além disso, esse enfoque obriga a revisão, quando não reprogramação das aplicações já existentes, o que vai contra uma das maiores motivações para o desenvolvimento de sistemas multiprocessados de memória compartilhada: a possibilidade de utilização quase que imediata de aplicações inicialmente desenvolvidas para sistemas monoprocessados.

Com base nessas observações, é possível visualizar a necessidade, nos diretórios limitados, de mecanismos eficientes para a gerência da falta de apontadores que não imponham *overheads* de espaço no armazenamento do diretório nem *overheads* de comunicação no envio de sinais de invalidação, e que não obriguem os programadores a se envolverem com detalhes de arquitetura da máquina para desenvolverem aplicações paralelas eficientes. Neste trabalho, apresentamos uma proposta de protocolo para diretório limitado com invalidação por difusão (*broadcasting*) que adota um mecanismo eficiente para o tratamento da falta de apontadores.

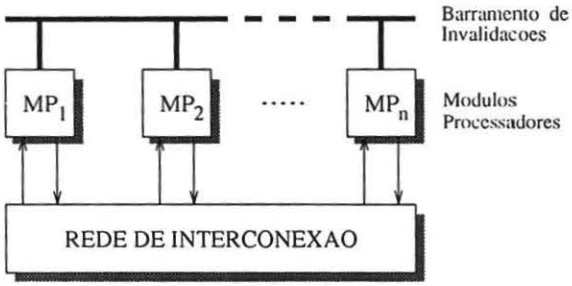

Figura 2: Localização do barramento de invalidações e suas respectivas conexões em um sistema multiprocessado.

3.1 Revisão dos Diretórios Limitados com Invalidação por Difusão

Em um diretório limitado com invalidação por difusão a falta de apontadores para um bloco é registrada através de um *bit* na entrada do diretório correspondente ao bloco. O bloco é suprido para tantos quantos forem os processadores que o requisitarem para leitura. No momento em que ocorre uma requisição para escrita ou é recebido um pedido de invalidação das cópias de um bloco, o diretório é verificado e, se não houve falta de apontadores, um sinal de invalidação é enviado a cada um dos processadores indicados pelos apontadores; caso tenha ocorrido a falta, o sinal é enviado pela rede a todos os processadores. Havendo ou não a falta de apontadores, todos os processadores que receberam um comando de invalidação devem enviar para o emissor do comando um sinal de reconhecimento da invalidação para que, com isso, se tenha a garantia de que todas as cópias foram invalidadas.

Em uma análise imediata dos protocolos de diretório limitado com invalidação por difusão percebe-se que eles são extremamente eficientes no atendimento às requisições para leitura de blocos com falta de apontadores (não há *overheads* de comunicação com sinais de invalidação e há um *overhead* de espaço insignificante decorrente do *bit* de difusão). Em contrapartida, a explosão de sinais que é sentida na rede no momento da difusão das invalidações revela o lado extremamente ineficiente dessa solução. Os sinais enviados a processadores que realmente possuem cópias do bloco são justos e inevitáveis, fazendo parte de qualquer solução com diretório existente; já os sinais de invalidação enviados a processadores sem cópia do bloco, além de desnecessários, competem com os outros sinais válidos do sistema na disputa por um recurso extremamente caro como é a rede de interconexão.

O verdadeiro responsável pela ineficiência do processo de difusão, entretanto, é a própria rede de interconexão que não oferece um meio adequado a este tipo de operação. Redes tipo barramento, por outro lado, são bastante eficazes no tratamento de difusões.

Propomos, portanto, um protocolo baseado em diretórios limitados dotado de um barramento especial com funções específicas para o tratamento dos sinais de difusão. O detalhamento deste barramento especial ou barramento de invalidações, assim como os argumentos que justificam a sua adoção são apresentados e discutidos mais adiante nesta seção. Suporemos, por hora, que o barramento de invalidações é um dispositivo perfeitamente adequado às necessidades do nosso protocolo. O protocolo proposto é descrito a seguir.

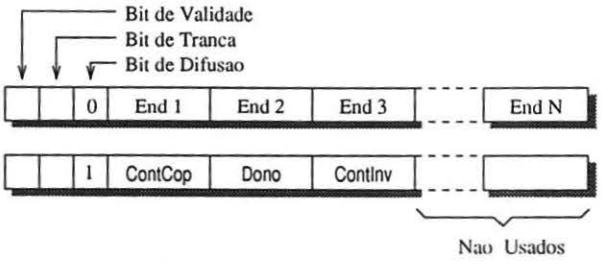

Figura 3: Redefinição dos campos no diretório DLE

Para efeito de clareza da apresentação, consideraremos que o sistema multiprocessado que usamos como referência é composto por módulos contendo processador, cache e parte da memória global compartilhada dotada de controlador de memória e diretório com entradas referentes aos blocos de memória do módulo. De fato, esse enfoque reflete a tendência observada na organização das atuais arquiteturas multiprocessadas — nada impede, todavia, a utilização do protocolo em arquiteturas onde a memória é organizada em bancos separados dos módulos processadores. A Figura 2 localiza o barramento de invalidações nesse modelo de sistema multiprocessado.

3.2 Descrição do Protocolo DLE — Diretório Limitado Eficiente

3.2.1 Organização do Diretório

Em cada banco de memória existe um diretório contendo informações referentes aos blocos de memória do banco. Para cada bloco de memória existe uma entrada no diretório constituída, basicamente, de 1 *bit* de validade, 1 *bit* de difusão, 1 *bit* de "tranca" e 3 ou mais campos com *bits* suficientes para endereçar qualquer um dos caches do sistema. O significado do conteúdo dos campos varia de acordo com o valor do *bit* de difusão: se o *bit* de difusão estiver desligado, os campos endereçam os únicos caches com cópia do bloco (End1, End2, End3, ..., EndN); se o *bit* de difusão estiver ligado, o primeiro campo é um contador que indica quantas cópias do bloco foram requisitadas pelos caches do sistema (CntCop), o segundo campo fica reservado para armazenar, no caso de uma requisição de alteração do bloco, o endereço do cache requisitante (Dono) e o terceiro campo é também um contador que indica o número de vezes que um mesmo sinal de invalidação foi enviado pelo barramento de invalidações (CntInv). Outros campos na entrada (End4, ..., EndN), se existirem, não são usados quando o *bit* de difusão estiver ligado. A redefinição dos campos em uma entrada do diretório é mostrada graficamente na Figura 3.

3.2.2 Algoritmo DLE

Localmente, o bloco no cache pode assumir um dos seguintes estados:

- *Inválido* - inconsistente, a cópia da memória ou a de outro cache é a mais atual;
- *Válido* - consistente, podem existir outras cópias do bloco em outros caches;

□ *Modificado* - inconsistente, o cache contém a versão mais atual do bloco.

A mudança de estados pode ocorrer em função de uma operação do processador local (leitura ou escrita) ou induzida por comandos enviados pelo controlador do diretório em resposta a requisições feitas por outros processadores. A Figura 4 apresenta o diagrama de estados de um bloco no cache segundo o protocolo DLE.

As ações a seguir são efetuadas a depender da operação que está sendo executada sobre o bloco no cache (leitura, escrita ou substituição) e seu respectivo estado.

1. **LEITURA** (bloco *válido* ou *modificado* no cache)

   A operação de leitura é atendida localmente.

2. **LEITURA** (bloco *ausente* ou *inválido* no cache)

   Uma requisição do bloco para leitura é feita ao banco de memória correspondente. Duas situações podem ocorrer:

   (a) *Bit* de modificação desligado

   O bloco de memória está consistente. Mais uma vez, duas situações podem ocorrer:

   i. *Bit* de difusão desligado
   Possivelmente existe apontador disponível na entrada do diretório correspondente ao bloco. O controlador de memória envia uma cópia do bloco, que assume o estado *válido* no cache requisitante, e aloca um dos apontadores para apontar para esse cache. Caso não exista mais apontador disponível, o *bit* de difusão é ligado e CntCop é inicializado com o número de apontadores que a entrada pode suportar adicionado de 1.

   ii. *Bit* de difusão ligado
   O controlador de memória envia uma cópia do bloco e incrementa CntCop na entrada do diretório correspondente ao bloco. A cópia assume o estado *válido* no cache requisitante.

   (b) *Bit* de modificação ligado

   O bloco de memória está inconsistente. Na entrada do diretório correspondente ao bloco, Dono indica qual o último cache que requisitou autorização para modificação (algoritmo de escrita a seguir) que, por conseguinte, contém a versão mais atual do bloco. O controlador de memória envia então um sinal para Dono atualizar o bloco na memória e enviar uma cópia do bloco para o cache requisitante. Tanto a cópia do bloco em Dono quanto no cache requisitante assumem o estado *válido*. No diretório, o *bit* de modificação é desligado e o primeiro apontador é atualizado com o endereço do cache requisitante (o segundo apontador já contém o endereço de Dono); os apontadores restantes ficam liberados.

3. **ESCRITA** (bloco *modificado* no cache)

   A operação de escrita é efetuada imediatamente na cópia local do bloco.

4. **ESCRITA** (bloco *válido*, *ausente* ou *inválido* no cache)

Antes de modificar o bloco, é necessário obter uma autorização. Um sinal é enviado ao banco de memória correspondente indicando o desejo de alteração do bloco por parte do cache; caso o bloco no cache esteja *ausente* ou *inválido*, um pedido de envio de cópia é associado ao pedido de autorização. Ao final da operação, o bloco no cache assume o estado *modificado*. Quando a requisição da autorização chega no controlador do diretório, duas situações podem ocorrer:

(a) *Bit* de modificação desligado

O bloco de memória está consistente. Mais uma vez, duas situações podem ocorrer:

i. *Bit* de difusão desligado

Um sinal de invalidação é enviado pela rede de interconexão para cada um dos caches endereçados pelos apontadores da entrada do diretório correspondente ao bloco.CntCop é atualizado com o número de cache para os quais foram enviados os sinais de invalidação e Dono passa a conter o endereço do cache que requisitou a autorização. A partir desse momento, o controlador diretório é liberado para atender a pedidos referentes a outros blocos. Nos caches que receberam o sinal de invalidação, a cópia do bloco envolvido assume o estado *inválido* e um sinal de reconhecimento/execução da invalidação é remetido para o banco de memória que solicitou a invalidação. Para cada sinal remetido, CntCop é decrementado e, ao alcançar o valor zero, o *bit* de modificação é ligado sendo o sinal de autorização enviado para Dono; caso uma cópia do bloco tenha sido requerida, ela é enviada juntamente com a autorização.

ii. *Bit* de difusão ligado

O diretório não tem controle de quais caches possuem cópias do bloco, apenas de quantos eles são através de CntCop. O endereço do bloco é colocado em um *buffer* de invalidações para ser difundido juntamente com o endereço do banco de memória pelo barramento de invalidações. Dono passa a conter o endereço do cache que requisitou a autorização. A partir desse momento, o controlador do diretório é liberado para atender a pedidos referentes a outros blocos. O pacote enviado pelo barramento de invalidações alcança todos os processadores e é armazenado em um *buffer* de chegada (a seção Barramento de Invalidações a seguir, detalha como são tratadas as exceções, ou seja, quando nem todos os módulos do sistema recebem o pacote de invalidação devido a situações como *buffer* de chegada cheio, por exemplo). Os caches que efetivamente possuem cópias do bloco, efetuam a invalidação e remetem um sinal para o banco de memória que solicitou a invalidação (seu endereço fora enviado juntamente com o pacote de invalidação) através da rede de interconexão indicando o seu reconhecimento/execução. Para cada sinal de reconhecimento/execução recebido, CntCop é decrementado e, ao alcançar o valor zero, o *bit* de modificação é ligado sendo o sinal de autorização enviado para Dono; caso uma cópia do bloco tenha sido requerida, ela é enviada juntamente com a autorização. O *bit* de difusão é desligado.

(b) *Bit* de modificação ligado

Na entrada do diretório correspondente ao bloco, Dono indica qual o último cache que requisitou autorização para modificação. Um sinal é então enviado a Dono para

que ele invalide sua cópia e envie o bloco para o cache requisitante; ao receber a cópia, o cache requisitante avisa o controlador que atualiza Dono para o novo possuidor da versão mais atual.

5. **SUBSTITUIÇÃO** (bloco *inválido* no cache)

   O bloco no cache está inconsistente e pode ser substituído dispensando qualquer operação extra.

6. **SUBSTITUIÇÃO** (bloco *válido* ou *modificado* no cache)

   Como o bloco substituído deixará de fazer parte do elenco de blocos no cache, o diretório deve ser atualizado. Duas situações podem ocorrer:

   (a) *Bit* de modificação desligado

   Não há necessidade de atualizar a memória. Mais uma vez, duas situações podem ocorrer:

   i. *Bit* de difusão desligado
   O apontador correspondente ao cache que está efetuando a substituição é liberado.

   ii. *Bit* de difusão ligado
   CntCop é decrementado e, caso tenha alcançado zero, o *bit* de difusão é desligado.

   (b) *Bit* de modificação ligado

   O bloco no cache é enviado para a memória a fim de atualizar o bloco de memória (*copy-back*). Em seguida, o *bit* de modificação é desligado e os apontadores liberados.

   OBS.: Como a operação de substituição ocorre em função de uma operação de leitura ou escrita, o sinal de substituição pode ser combinado com esses outros, com o objetivo de reduzir o número de sinais que trafegam pela rede de interconexão.

3.2.3 Sincronização

O protocolo DLE foi especificado de forma que, em nenhum instante, o controlador do diretório fica impedido de atender a outras requisições enquanto está aguardando a execução de operações que, apesar de estarem relacionadas com o atendimento de uma requisição em andamento, não dependem do controlador; é o caso, por exemplo, das operações envolvidas no processo de invalidação das cópias de um bloco em uma requisição de escrita. Por um lado, essa decisão de projeto torna possível a obtenção do desempenho máximo por parte do controlador do diretório, uma vez que, ele fica habilitado a atender a diversas requisições de maneira sobreposta; além disso, ela se adequa perfeitamente à forma de operação, também assíncrona, das redes multiestágios — onde normalmente não existe o estabelecimento de conexões, mas sim a troca de mensagens na forma de pacotes. Por outro lado, essa forma de operação requer mecanismos eficientes de sincronização que garantam a execução completa, consistente e sem conflitos de cada uma das requisições recebidas. No protocolo DLE, esses mecanismos estão representados basicamente por dois objetos: o *bit* de tranca e o contador de invalidações CntInv.

O *bit* de tranca tem como função evitar que duas requisições distintas atuem simultaneamente sobre um mesmo bloco. Ele é ligado antes do controlador do diretório efetuar qualquer

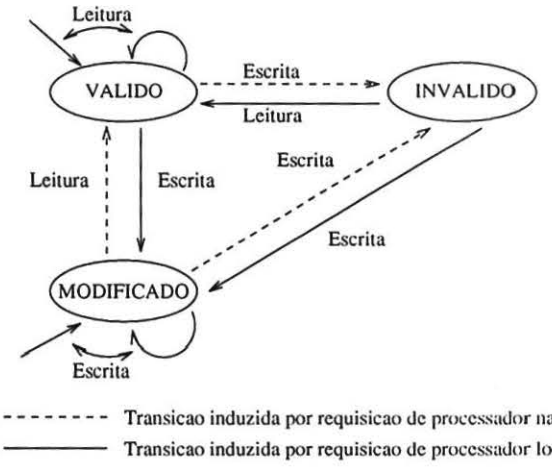

Figura 4: Diagrama de estados de um bloco no cache segundo o protocolo DLE

operação sobre um bloco (leitura, escrita, tratamento de pedidos de invalidação, etc.) para indicar que uma requisição sobre aquele bloco está pendente e que outras requisições sobre o mesmo bloco devem ser inibidas até que esse *bit* seja desligado. O *bit* de tranca resolve, portanto, a maioria dos problemas resultantes de requisições conflitantes como é o caso de requisições de leitura de um bloco que está sofrendo invalidações decorrentes de uma requisição para escrita anterior. As requisições conflitantes podem ser tratadas de duas formas:

1. Toda requisição direcionada a blocos com o *bit* de tranca ligado é rejeitada, obrigando o cache requisitante a refazer a requisição;
2. O controlador armazena localmente as requisições que podem ser atendidas assim que o *bit* de tranca seja liberado, como é o caso do exemplo acima[2] e rejeita as que são realmente inviáveis, como é o caso de duas requisições simultâneas para escrita — nesse caso, uma é atendida e a outra é rejeitada.

O contador de invalidações (ContInv) por sua vez, tem por objetivo auxiliar na detecção e correção de possíveis falhas na operação do barramento de invalidações. Considere, por exemplo, a situação na qual uma difusão pelo barramento de invalidações não ocorreu de maneira perfeita e nem todos os processadores registraram o comando de invalidação. Isso pode acontecer por razões que vão desde a falta de espaço no *buffer* de chegada de um processador que, coincidentemente, possui uma cópia do bloco a ser invalidado — uma situação pouco provável, porém possível — até uma falha de operação do próprio barramento. Nesses casos, o aviso de reconhecimento/execução da invalidação não é remetido, o contador de cópias (ContCop) do bloco nunca alcança o valor zero e o processador que efetuou a requisição para escrita do bloco nunca recebe a autorização; além disso, o controlador fica impossibilitado de atender a qualquer outra requisição direcionada a esse bloco.

[2]Um *bit* auxiliar na entrada do bloco pode indicar a existencia de operações aguardando a liberação.

Para evitar esse tipo de *deadlock*, o processador que enviou a requisição de alteração deve, decorrido um certo período de tempo sem o retorno da autorização por parte do controlador, ressubmeter a requisição. Dessa forma, na tentativa de atender a esse novo pedido, o controlador enviará novamente o comando de invalidação incrementando ContInv. O processo se repete até que a requisição seja atendida, atestando o sucesso das invalidações, ou que ContInv alcance um valor predeterminado, forçando o controlador a executar ações de emergência que vão desde a difusão de comandos de invalidação pela própria rede de interconexão, até a indicação de uma possível falha de *hardware* do barramento de invalidações. Nesse último caso, é possível obter-se um bom nível de tolerância a falhas, se, a partir do momento da detecção da falha, o controlador do diretório iniciar a execução de um protocolo alternativo do tipo *não difusão* até que o problema no barramento de invalidações seja sanado.

3.3 Barramento de Invalidações

Trata-se de um barramento que segue os mesmos princípios de funcionamento de um barramento padrão, com as vias de sinais e circuitos lógicos de controle habituais, mas que é voltado exclusivamente para a difusão de invalidações. A princípio, o comportamento serializado com que o acesso a um barramento é feito, pode levar à conclusão de que ele rapidamente alcança o ponto de saturação com um número reduzido de dispositivos conectados, como aliás já fora afirmado neste trabalho. Entretanto, diversas razões nos levam a crer que, se utilizado da forma especializada como está sendo proposta, o ponto de saturação do barramento, apesar de existir, está muito além dos registrados quando o barramento é utilizado como rede de interconexão, onde ele é obrigado a suportar uma gama variada de formas de operação. Algumas dessas razões estão relacionadas a seguir:

- Diferentemente de um barramento padrão, o barramento de invalidações, como o nome indica, é usado exclusivamente para o envio de sinais específicos de invalidação, operação que pode ser efetuada em um único ciclo. Em um barramento padrão, por outro lado, um acesso para uma leitura ou *copy-back* de um bloco, por exemplo, requer um ciclo de endereçamento seguido de tantos ciclos quantos forem necessários para a transferência do bloco (durante esse período o barramento fica inabilitado para o atendimento às requisições de outros componentes do sistema).

- A necessidade do envio de sinais de invalidação pelo barramento está restrita aos casos em que houve falta de apontadores no diretório e não a todos os casos de escritas a blocos compartilhados. Como já discutido nesse trabalho, espera-se que essas sejam situações infrequentes.

- A difusão do sinal de invalidação é feita de forma unidirecional, do controlador do diretório para os caches do sistema; a acusação de recebimento e execução da invalidação é feita através da rede de interconexão. Em um barramento padrão, o processo de comunicação é normalmente feito de forma bidirecional, com uma requisição sendo enviada em um sentido e respondida em outro.

- Como o barramento de invalidação é usado de forma específica – para a difusão de um pacote de *bits* de tamanho predefinido —, ele pode ser projetado para otimizar esse tipo de operação fazendo uso, por exemplo, de *buffers* nos *drivers*, operação *pipeline*, etc.

Apesar da validade das razões acima, desenvolvemos a seguir uma expressão que pode ser usada na previsão do ponto de saturação do barramento de invalidações a partir da sua taxa de transferência e da velocidade dos processadores a ele conectados.

3.3.1 Estimativa de Saturação

A taxa de transferência (*bandwidth*) fixa transforma o barramento em um recurso limitante do desempenho em sistemas que o utilizam de forma compartilhada. Na subseção anterior afirmamos que o ponto onde um barramento especializado começa a limitar o desempenho está muito além dos observados em barramentos não especializados. A questão que surge, por conseguinte, é qual seria o ganho obtido com a utilização especializada? Especificamente no caso do barramento de invalidações, qual a estimativa de expansão do sistema, em número de processadores, antes que se estabeleça o congestionamento dessa via? O desenvolvimento de uma expressão a partir do resultado de estudos recentes dos padrões de invalidação observados em aplicações paralelas reais pode nos dar uma idéia da resposta a essas questões.

Primeiramente, vamos desenvolver uma expressão que nos informe o número médio de requisições à memória que ocasionam a necessidade do uso do barramento de invalidações. Usaremos como referência um sistema com N processadores, dotado de diretórios limitados com 4 apontadores por entrada.[3]

Seja ω a probabilidade de um processador qualquer efetuar uma referência à memória para escrita a um bloco compartilhado no início de um ciclo de memória e seja β a probabilidade de que uma dessas escritas envolve um bloco em cuja entrada o *bit* de difusão encontra-se ligado, ou seja, o número de cópias do bloco em caches do sistema é maior que 4. O número esperado de sinais enviados pelo barramento é dado, portanto, por $\rho \times \omega \times \beta$, onde ρ é o desempenho máximo do sistema, dado por N multiplicado pelo número máximo de instruções que cada processador pode executar por unidade de tempo. Logo, a seguinte relação, entre a taxa de transferência do barramento de invalidações $\mathcal{TT}$ e a expressão acima, deve ser válida para obter-se o máximo desempenho:

$$\mathcal{TT} \geq \rho \times \omega \times \beta \qquad (1)$$

A Tabela 1 traz os valores típicos de ω e β obtidos por Gupta e Weber [14] a partir das referências à memória geradas em simulações de 4 aplicações paralelas reais. Apesar dos dados dessa tabela estarem relacionados a um sistema com 32 processadores, os autores afirmam que o número de invalidações manteve-se constante quando a simulação foi repetida para 8 e 16 processadores; adicionalmente, eles não acreditam que esses dados sofram mudanças significativas em sistemas com um número elevado de processadores.

Para calcularmos ρ na inequação 1 usaremos o maior produto $\omega \times \beta$ da Tabela 1, dado pela aplicação *LocusRoute* ($\omega = 0,05$ e $\beta = 0,06$) e tomaremos como referência para o barramento de invalidações as características do FutureBus+, um dos barramentos de alto desempenho da atualidade.[2]. O FutureBus+ é capaz de fornecer uma taxa de transferência da ordem

[3]Esse parece ser um número razoável que atende bem ao compromisso "espaço ocupado pelo diretório × probabilidade de falta de apontadores" discutido no início deste capítulo e que vem sendo indicado como adequado por estudos como [9], [18] e [14].

Tabela 1: Valores típicos de ω e β obtidos por [14] a partir de aplicações paralelas reais.

Aplicação	Num. CPU	Data Ref. milhões	Escrita Comp. milhões	Escrita Comp. %(ω)	Invalidações (Apt > 4) %(β)
MaxFlow	32	25,5	2,2	8	3
Water	32	48,0	1,0	2	1
PTHOR	32	16,6	0,9	5	4
LocusRoute	32	18,5	1,0	5	6

de 100 MTransf./seg. (64 *bits*/transf.) em seu modo pacote; nesse modo, a necessidade do estabelecimento de conexão (*handshaking*) antes do envio dos dados é eliminada, obtendo-se, com isso, altas taxas de transferência[4].

Temos portanto:

$$100M \geq \rho \times 0,05 \times 0,06$$

$$\rho \leq 33.333M \quad (2)$$

Para calcular o número máximo de processadores que obedece a relação 2, vamos tomar o desempenho máximo, da ordem de 2,5 MIPS [20], dos processadores usados atuais sistemas multiprocessados com rede de interconexão multiestágios como é o caso do BBN Butterfly (68020) e do IBM RP3 (IBM ROMP - um processador próprio). Além disso, vamos presumir que a execução de cada instrução gera uma referência à área de dados. Assim,

$$2,5M \times N \leq 33.333M$$

$$N \leq 13.333$$

Esse valor máximo de N ultrapassa, em mais de uma ordem de grandeza, o número máximo de processadores projetados para aqueles sistemas, 256 para o BBN Butterfly e 512 para o IBM RP3. Mesmo quando consideramos os processadores mais modernos (10 vezes mais rápidos, por exemplo), ainda assim esse limite é bem superior.

Apesar do resultado obtido com os cálculos anteriores não ser conclusivo, ele nos dá uma boa indicação da viabilidade da utilização do barramento de invalidações nas futuras gerações de sistemas multiprocessados.

3.3.2 Restrições de Comprimento, Alimentação e Arbitragem

A conexão de um número muito grande de dispositivos a um único barramento encontra dois fatores de limitação: a extensão ou comprimento desse barramento e o nível de corrente do

[4]Esse modo de operação se adequa perfeitamente à forma de operação do barramento de invalidações, onde um pacote contendo o endereço do bloco a ser invalidado é difundido pelo barramento para todos os outros processadores do sistema; considerando-se pacotes de 64 *bits* tem-se uma capacidade de endereçamento de 2^{64} blocos.

sinal que irá circular por ele. Por um lado, um barramento muito extenso, operando a uma frequência muito alta, está sujeito à captação de ruído e a outros efeitos eletromagnéticos que causam distorções e atrasos no sinal que está sendo propagado; por outro lado, o nível de corrente necessário para que o sinal seja percebido por todos os pontos de conexão pode chegar a valores que inviabilizam a construção prática de barramentos densamente compartilhados [15]. Além desses dois fatores, um outro que deve igualmente ser considerado é a complexidade dos circuitos de arbitragem — responsáveis por decidir quem tem o direito de colocar o sinal no barramento a cada instante — que aumenta linearmente com o número de conexões.

A tecnologia na área de barramentos tem evoluído na direção de soluções eficazes para os dois primeiros problemas. Uma descrição aprofundada do enfoque adotado no FutureBus na solução desses problemas pode ser encontrada em [6].

Já o problema da arbitragem não parece ter uma solução viável para um número elevado de conexões. A maioria dos barramentos recentes, incluindo o FutureBus, adota um mecanismo de decisão que baseia-se em prioridades mas que restringe o número máximo de dispositivos que podem ser conectados. Felizmente, graças à forma assíncrona com que o algoritmo DLE trata o envio de invalidações pelo barramento, é possível projetar uma estrutura de barramento com sub-barramentos que viabiliza a existência de um grande número de pontos de conexão. Um exemplo dessa estrutura poderia ser obtido através da divisão dos processadores em grupos, com cada grupo conectado a um barramento particular e cada barramento particular sendo conectado a um barramento principal conforme a Figura 5. Todo sinal de invalidação enviado pelo barramento particular alcançaria todos os outros processadores a ele conectados e também o ponto de conexão com o barramento principal; a partir do barramento principal o sinal seria então distribuído para todos os outros processadores. Essa solução além de sanar os 3 problemas discutidos acima, permite que o sistema parta de uma configuração inicial e possa ser expandido gradativamente.

3.3.3 Utilizações Anteriores

A idéia de utilizar um barramento com funções específicas de invalidação não é nova. De fato, ela fora originalmente usada na entitulada **solução clássica** para o problema da coerência de cache, mencionada no capítulo 2. De acordo com Censier e Feautrier [7], essa solução é encontrada nos primeiros biprocessadores que adotavam política *write-through* para a atualização da memória.[5] O envio de sinais de invalidação à cada operação de escrita de um processador em consequência dessa política, contudo, confina a aplicação da solução clássica a sistemas com um número reduzido de processadores. Além disso, existe uma pequena probabilidade de ocorrência de inconsistências, caso operações de leitura sejam atendidas pelo cache entre o momento em que uma modificação é feita e a invalidação é efetuada no bloco envolvido.

Apesar da semelhança funcional, diversos fatores diferenciam os barramentos de invalidações do protocolo DLE e da solução clássica. A restrição do uso do barramento apenas nos casos de blocos com falta de apontadores reduz, como visto, enormemente o tráfego induzido por cada processador nessa via. O emprego de tecnologia recente na área de barramentos

[5] Entre os sistemas comerciais que adotaram a solução clássica estão os biprocessadores IBM 370/168 e o 3033, além dos sistemas monoprocessados, como o VAX11/780, onde ela é usada para garantir a coerência nos acessos à memória entre o processador central e os processadores de I/O (cf. [1]).

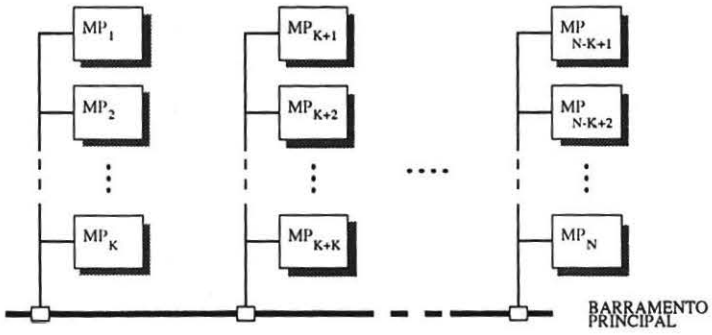

Figura 5: Organização sugerida para o barramento de invalidações como forma de solução para as restrições de Comprimento, Alimentação e Arbitragem.

também contribui para um desempenho esperado acentuadamente melhor do protocolo DLE. Finalmente, a existência do diretório na intermediação de uma operação de escrita a blocos compartilhados elimina a probabilidade de ocorrência de inconsistências; a serialização das operações é garantida, já que, só é dada a autorização a um cache para prosseguir com a modificação de um bloco quando todos os outros confirmarem a execução da invalidação de suas cópias locais.

Por outro lado, algumas características de implementação permanecem comuns às duas soluções. Entre elas, a inevitável necessidade de duplicação dos diretórios dos caches (para que o algoritmo de monitoração do barramento de invalidações não cause interferências no atendimento do cache às requisições dos processadores) e, também, de pequenos *buffers* em cada ponto de interconexão com o barramento para acomodar instantes de pico no tráfego dos sinais de invalidação.

4 Considerações Finais

O uso de trancas (*locks*) em operações de sincronização é um exemplo de situação onde espera-se um bom desempenho do protocolo DLE. Normalmente, esse tipo de operação envolve muitos processadores, o que aumenta a probabilidade da ocorrência de falta de apontadores e, consequentemente, de *overheads*. Nesses casos, a rede de interconexão é a maior beneficiada, já que, no protocolo DLE, um mesmo bloco pode ser compartilhado por até todos os processadores do sistema, sem a necessidade do envio de mensagens "extras" pela rede; semelhantemente, no momento de liberação da tranca, a via auxiliar é usada para o envio da invalida, oes e a rede é mais uma vez utilizada de forma mínima.

Atualmente, encontra-se em fase de desenvolvimento um ambiente de simulação que pretendemos usar para validar o protocolo DLE. Basicamente, esse ambiente será composto de um gerador de *traces* que, a partir das distribuições das referências à memória de aplicações paralelas reais — como as divulgadas em [14] —, produzirá os dados que alimentarão o simulador do protocolo. Espera-se obter resultados diversos; entre eles, comparações de desempenho

com outros protocolos de diretórios limitados e com outras classes de diretório (mapa de *bits*, por exemplo), taxa de utilização da rede de interconexão e comportamento do barramento de invalidações em situações de pico.

Referências

[1] John Hennessy Anant Agarwal, Richard Simoni and Mark Horowitz. An Evaluation of Directory Schemes for Cache Coherence. *Proc. 15th. Int. Symp. on Computer Architecture*, 1988.

[2] Warren Andrew. 32-Bit Buses Contend for Designers' Attention. *Computer Design*, pages 78–96, November 1, 1989.

[3] J. Archibald and Jean-Loup Baer. Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model. *ACM Transactions on Computer Systems*, 4(4):273–298, November 1986.

[4] James Archibald and Jean-Loup Baer. An Economical Solution to the Cache Coherence Problem. *Proc. Int. Symp. on Computer Architecture*, pages 355–362, 1984.

[5] Antonio Carlos Fontes Atta. Análise do Problema da Coerência de Cache em Sistemas Multiprocessados de Memória Compartilhada, 1993. Tese de Mestrado em preparação.

[6] R. V. Balakrishnan. The Proposed IEEE 896 FutureBus — A Solution to the Bus Driving Problem. *IEEE Micro*, pages 23–27, August 1984.

[7] L. M. Censier and P. Feautrier. A New Solution to Coherence Problems in Multicache Systems. *IEEE Transactions on Computers*, C-27(12):1112–1118, December 1978.

[8] John Kubatowicz David Chaiken and Anant Agarwal. LimitLESS Directories: A Scalable Cache Coherence Scheme. *Proc. of 4th ASPLOS*, pages 224–234, 1991.

[9] Kiyoshi Kurihara David Chaiken, Craig Fields and Anant Agarwal. Directory-Based Cached Coherence in Large-Scale Multiprocessors. *IEEE Computer*, 23(6):49–58, 1990.

[10] C. P. Thacker et al. Firefly: A Multiprocessor Workstation. *IEEE Transactions on Computers*, 37(8):909–920, August 1988.

[11] R. H. Katz et al. Implementing a Cache Consistency Protocol. *Proceedings of 12th Int. Symp. on Comp. Architecture*, pages 276–283, June 1985.

[12] Tse-Yun Feng. A Survey of Interconnection Networks. *IEEE Computer*, 14(12):12–27, Dec. 1981.

[13] J. R. Goodman. Using Cache Memory to Reduce Processor-Memory Traffic. *Proceedings of 10th Int. Symp. on Comp. Architecture*, pages 124–131, June 1983.

[14] Anoop Gupta and Wolf-Dietrich Weber. Cache Invalidation Patterns in Shared Memory Multiprocessors. *IEEE Transaction on Computers*, 41(7):794–810, 1992.

[15] David B. Gustavson. Computer Buses — A Tutorial. *IEEE Micro*, pages 7–22, August 1984.

[16] John M. Mellor-Crummey and Michael L. Scott. Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors. *ACM Transactions on Computers*, 9(1):21–65, February 1991.

[17] M. S. Papamarcos and J. H. Patel. A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories. *Proceedings of 11th Int. Symp. on Comp. Architecture*, pages 348–354, June 1984.

[18] Richard Simoni and Mark Horowitz. Modelling the Performance of Limited Pointers Directories for Cache Coherence. *Proc. 18th Int. Symp. on Computer Architecture*, pages 309–318, 1991.

[19] Per Stenstrom. A Survey of Cache Coherence Schemes for Multiprocessors. *IEEE Computer*, 23(6):12–24, June 1990.

[20] Daniel Tabak. *Multiprocesors*. Prentice-Hall, 1990.

[21] Dhiraj K. Pradhan Yeong-Chang Maa and Dominique Thiebaut. Two Economical Directory Schemes for Large-Scale Cache Coherent Multiprocessors. *Computer Architectures News*, 19(5):10–18, 1991.