PROVA DE CORREÇÃO DE MECANISMOS

DE CONTROLE DE COERÊNCIA EM MULTIPROCESSADORES

Adélia Cecília G. Nunes*
Daniel A. Menascé**

RESUMO

Em ambientes multicache, cada processador tem a ele associado uma memória cache privada, utilizada no armazenamento de dados provenientes da memória global. A consistência entre as cópias de um dado, que em um determinado momento pode existir em mais de uma memória cache, e o dado armazenado na memória global é garantida por um mecanismo de controle de coerência, que pode variar de multiprocessador para multiprocessador. Neste artigo nós propomos uma metodologia de prova de correção de mecanismos de controle de coerência para ambientes multicache orientados a barramento.

ABSTRACT

In cache-based multiprocessors, each processor has an associated private cache memory, where data from global memory is stored. In these systems, a coherence control mechanism, that may differ from system to system, is implemented to ensure that all copies of a global memory data remain consistent. In this paper we propose a framework for proving the correctness of coherence control mechanisms for shared-bus multiprocessors.

* Analista de Sistemas da EMBRATEL; Mestre em Ciências da Informática pela PUC-RJ; Áreas de Interesse: Modelagem de Sistemas de Computação, Processamento Paralelo e Redes de Computadores; Endereço: R. Prof. Manoel Ferreira 88/602, Gávea, RJ, Cep.22451; Telefone: (021)216-7796; E-mail: DELI@BRLNCC.BITNET.

** Professor Associado do Departamento de Informática da PUC-RJ; Ph.D. em Ciência da Computação pela UCLA; Áreas de Interesse: Processamento Paralelo, Modelagem de Sistemas de Computação e Arquiteturas de Alto Desempenho; Endereço: Departamento de Informática da PUC-RJ, Cep.22453; Telefone: (021)529-9526; E-mail: MENASCE@BRLNCC.BITNET.

1. Introdução

Mecanismo de Coerência é um mecanismo que garante a coerência do conjunto de memórias do sistema, formado pelos módulos da memória global e pelas memórias cache, não permitindo que existam duas versões diferentes de um mesmo dado, consideradas válidas, em um mesmo instante, neste conjunto de memórias. Em outras palavras, é um mecanismo que faz com que este conjunto de memórias funcione como se fosse uma única memória física que não contém dados duplicados.

Um ambiente multicache sem um mecanismo de coerência pode apresentar problemas de coerência quando existem dados compartilhados entre processos executando em processadores diferentes, ou quando é possível a migração de um processo de um processador para outro.

Vários mecanismos de coerência foram propostos na literatura *[SMIT 82]*, alguns independentes do tipo de rede de interconexão existente no processador, outros com maior aplicação em ambientes orientados a barramento, já que exploram a facilidade de difusão existente nestes ambientes.

Dois tipos de mecanismo de coerência aplicáveis a qualquer tipo de rede de interconexão, são : *Controle por Software* e *Mecanismos Baseados em Diretórios.*

Em um *Mecanismo de Controle por Software [BRAN 85]*, determinadas informações compartilhadas, tais como semáforos, são classificadas, como não podendo ser armazenadas em memória cache, só podendo ser recuperadas diretamente da memória global. Tal procedimento impede a existência de mais de uma cópia da informação no sistema. Por razões de eficiência, algumas variações deste mecanismo permitem que algumas informações compartilhadas sejam armazenadas em memória cache. Neste caso, o sistema possui instruções especiais que permitem ao processador invalidar, quando necessário, a cópia de uma informação compartilhada, existente em sua memória cache. Tais instruções são também necessárias em sistemas que permitem a migração de processos.

Em *Mecanismos Baseados em Diretórios*, o sistema mantém um diretório centralizado ou distribuído, que mantém informações sobre cada linha da memória global. Estas informações são utilizadas para garantir que nenhuma linha exista simultaneamente na memória cache de dois processadores em versões diferentes.

Os mecanismos de coerência com maior aplicação em ambientes multicache orientados a barramento são em geral mecanismos baseados em diretórios distribuídos *[ARCH 86]*. Este diretório é distribuído pelas memórias cache do sistema e é mantido pelo controlador de cada memória cache. Para cada linha da memória cache são mantidos bits de informação que fornecem subsídios ao

controlador da memória cache, na execução das solicitações de leitura e escrita do processador, de forma a manter a coerência entre as cópias do dado.

Os mecanismos de coerência propostos têm sido bastante analisados quanto a sua eficiência *[DUBO 82, DUBO 87, DUBO 88, DUBO 89, VERN 86, VERN 88]*, entretanto, nenhum esforço tem sido feito na análise destes mecanismos quanto a sua correção. Assim, estamos propondo neste artigo uma metodologia para a prova de correção de mecanismos de coerência. Por limitações de espaço, só apresentaremos a prova de correção do mecanismo de coerência Dragon *[ARCH 86]*. Esta metodologia foi aplicada na prova de correção de outros mecanismos de coerência.

Na seção 2, nós descrevemos os conceitos e definições utilizados pela metodologia proposta. Na seção seguinte, é apresentada uma breve descrição do mecanismo Dragon, enquanto na seção 4 apresentamos a prova de correção deste mecanismo. A seção 5 conclue este artigo.

2.Conceitos Básicos e Definições

Definição 1: Um mecanismo de coerência para um ambiente multicache é dito *correto* se e somente se as solicitações de leitura e escrita, originadas por qualquer memória cache do sistema, leva o conjunto de memórias de um estado coerente para outro estado coerente.

A definição de correção introduz o conceito de estado coerente. Definiremos a seguir o que é um estado do conjunto de memórias e depois, mais especificamente, o que é um estado coerente do conjunto de memórias. Para definirmos estado coerente, precisamos classificar os mecanismos de coerência nos dois tipos descritos a seguir :

Tipo A : Mecanismos onde o número de memórias cache com versão mais recente que a da memória global é limitado a um, exemplificado pelo mecanismo Write-Once *[GOOD 83]*.

Tipo B : Mecanismos onde pode existir mais de uma memória cache com versão mais recente que a versão da memória global, exemplificado pelo mecanismo Dragon.

Definição 2: *Estado do conjunto de memórias* é um vetor de N + 1 elementos, onde N é o número de memórias cache do sistema, cujos elementos são vetores que representam o conteúdo da memória.

Temos portanto :

estado = $\vec{\vartheta} = (\vec{\vartheta}g \, , \, \vec{\vartheta}c_1 \, , \, \vec{\vartheta}c_2 \, , \, \ldots \, , \, \vec{\vartheta}c_n)$

$\vec{V}g = (v_1 , v_2 , \ldots , v_m)$
$\vec{V}c_i = (v_{1i} , v_{2i} , \ldots , v_{mi})$

onde :

$\vec{V}g$: conteúdo da memória global
v_j : conteúdo da posição j da memória global

$\vec{V}c_i$: conteúdo da memória cache i

v_{ji}: versão do dado armazenado, na memória cache i, na posição j da memória global. Para efeito de notação, quando não existir, na memória cache i, uma cópia do dado na posição j da memória global, denotaremos $v_{ji} = \not\in$.

m : número de posições de memória na memória global

<u>Definição 3a:</u> *Estado coerente do conjunto de memórias, para um mecanismo de coerência do tipo A,* é um estado do conjunto de memórias no qual, para qualquer dado j (na posição j da memória global) :

(i) se a memória global possui a versão mais recente do dado j, então ou $v_{ji} = \not\in$ ou $v_{ji} = v_j$, $\forall$ memória cache i;

(ii) se a memória global não possui a versão mais recente do dado j e a memória cache k possui a versão mais recente, então $v_{ji} = \not\in$ $\forall$ memória cache i, $i \neq k$, e existe procedimento de *wback*(definição 4).

<u>Definição 3b:</u> *Estado coerente do conjunto de memórias, para um mecanismo do tipo B,* é um estado do conjunto de memórias no qual, para qualquer dado j (na posição j da memória global) :

(i) se a memória global possui a versão mais recente do dado j, então ou $v_{ji} = \not\in$ ou $v_{ji} = v_j$, $\forall$ memória cache i;

(ii) se a memória global não possui a versão mais recente do dado j e a memória cache k foi a última a atualizar este dado, então $v_{ji} = \not\in$ ou $v_{ji} = v_{jk}$ $\forall$ memória cache $i \neq k$, e existe procedimento de *difusão* (definição 5).

<u>Definição 4:</u> Procedimento de *wback* é um procedimento, necessário em um mecanismo de coerência do tipo A, no qual escritas em memória cache podem não ser imediatamente difundidas para a memória global. Neste procedimento, a memória cache responsável pela última atualização do dado, se encarrega de fornecer a versão mais recente do dado à qualquer memória cache solicitante, e de atualizar a versão da memória global, quando o dado for substituído por razões de espaço. Este mecanismo deve garantir ainda que, em cada instante, exista somente uma única memória cache com versão mais recente do dado.

Definição 5: Procedimento de *difusão* é um procedimento, necessário em um mecanismo de coerência do tipo B, onde a memória cache responsável pela última atualização do dado se encarrega de difundir o valor mais recente do dado, à qualquer memória cache solicitante, e de atualizar a memória global quando o dado for substituído por razões de espaço.

3. Breve Descrição do Mecanismo Dragon

Esta descrição consiste na especificação dos possíveis estados de uma dado na memória cache, determinados pelos bits de informação, e na definição das ações que devem ser executadas pelo controlador da memória cache, no recebimento de uma solicitação de leitura/escrita recebida do processador associado, ou detectada no barramento. Existem quatro tipos de solicitação a serem tratadas pelo controlador: leitura de uma dado existente na memória cache (*read hit)*, leitura de um dado não existente na memória cache (*read miss)*, escrita em um dado existente na memória cache (*write hit)* e escrita em um dado não existente na memória cache (*write miss)*. No caso de uma solicitação do tipo read hit, a ação a ser tomada pelo controlador é a de fornecer imediatamente, ao processador, a cópia do dado, sem mudar o estado do dado na memória cache. Portanto esta solicitação será omitida da descrição. No caso das solicitações read/write miss, o controlador da cache redireciona a solicitação para o barramento e espera a cópia do dado solicitado.

O mecanismo Dragon é caracterizado pela difusão de cada escrita feita no dado, para todas as demais memórias cache, e pela atualização da memória global somente em caso de substituição do dado atualizado, na memória cache responsável pela última atualização. Além disto, este mecanismo utiliza sinais recebidos de um barramento especial que interliga as memórias cache do sistema. Chamaremos este barramento de Linha Compartilhada. Ao detectar uma solicitação no barramento, uma memória cache com cópia do dado solicitado envia sinal pela Linha Compartilhada indicando que possui cópia do dado. O controlador da memória cache solicitante utiliza estes sinais, na Linha Compartilhada, para decidir se o dado solicitado existe em outras memórias cache ou não.

Neste mecanismo, um dado em uma memória cache pode estar em um dos seguintes estados: *válido-exclusivo*, *compartilhado*, *sujo-compartilhado* ou *sujo*. Um dado no estado *válido-exclusivo* tem a mesma versão da memória global e não existe em nenhuma outra memória cache, e um dado no estado *compartilhado* pode não ter a mesma versão da memória global e possivelmente existe em outras memórias cache. Caso um dado no estado compartilhado não tenha a mesma versão da memória global, existe necessáriamente no sistema uma cópia do dado no estado sujo-compartilhado. Um dado no estado *sujo-compartilhado* tem a versão mais recente que a da memória global e possivelmente existe em outras memórias cache

(no estado compartilhado); dados neste estado, quando selecionados para serem substituídos, provocam atualização da memória global. Um dado no estado *sujo* tem a versão mais recente que a da memória global e não existe em nenhuma outra memória cache; dados neste estado, quando selecionados para serem substituídos, provocam atualização da memória global.

O comportamento do sistema para cada uma das solicitações de leitura/escrita, sob o mecanismo Dragon, ilustrado na figura 1, é apresentado a seguir:

read miss : Se existe uma versão do dado no estado *sujo* ou *sujo-compartilhado*, o controlador da memória cache com esta versão fornece o dado, envia sinal pela Linha Compartilhada, e muda o estado do dado para *sujo-compartilhado*.
Se não existe nenhuma cópia do dado no estado *sujo* ou *sujo-compartilhado*, o dado é fornecido pela memória global. O controlador das memórias cache, com cópia do dado solicitado no estado *válido-compartilhado*, enviam sinal pela Linha Compartilhada e mudam o estado do dado para *compartilhado*.
O dado, na memória cache solicitante, é armazenado no estado *válido-exclusivo*, se não existe cópia do dado em nenhuma outra memória cache, ou no estado *compartilhado*, caso contrário.

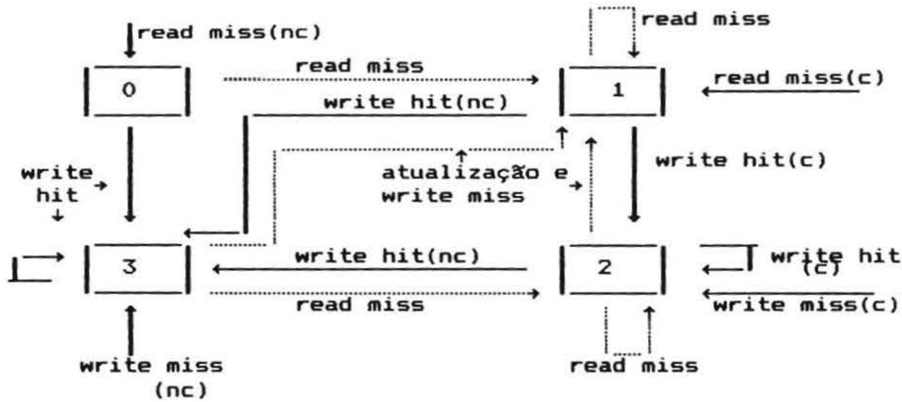
write hit : Se o dado armazenado na memória cache solicitante está no estado *sujo*, a escrita é feita imediatamente.
Se o dado está armazenado no estado *válido-exclusivo*, a escrita é feita imediatamente ao mesmo tempo que o seu estado é mudado para *sujo*.
Se o dado solicitado está armazenado no estado *compartilhado* ou *sujo-compartilhado*, o controlador da memória cache solicitante aguarda até poder enviar uma solicitação de escrita às demais memórias cache, pelo barramento, e realiza a escrita. O controlador de cada memória cache com cópia do dado atualiza a sua versão ao mesmo tempo que envia sinal pela Linha Compartilhada, e muda o estado do seu dado para *compartilhado*.
A memória cache solicitante, observando a Linha Compartilhada, atualiza o estado do dado para *sujo*, se nenhuma memória cache possui cópia do dado, ou para *sujo-compartilhado*, caso contrário.

write miss : Se existe uma versão do dado no estado *sujo* ou *sujo-compartilhado*, o controlador da memória cache com esta versão fornece o dado, envia sinal pela Linha Compartilhada e muda o estado do dado para

compartilhado.

Se não existe nenhuma versão do dado no estado *sujo* ou *sujo-compartilhado*, o dado é fornecido pela memória global. O controlador das memórias cache com cópia do dado no estado *válido-exclusivo* ou *compartilhado*, enviam sinal pela Linha Compartilhada e mudam o estado do dado para *compartilhado.*

Se não existe cópia do dado em nenhuma memória cache, o controlador da memória cache solicitante armazena o dado no estado *sujo* e realiza a escrita. Caso contrário, o controlador da memória cache solicitante armazena o dado no estado *sujo-compartilhado* e realiza a escrita.



nc : nenhum controlador enviou sinal pela Linha Compartilhada

c : algum controlador enviou sinal pela Linha Compartilhada

⟶ : transição devido à solicitação do processador associado

┈┈➝ : transição devido à solicitação detectada no barramento

Estados : 0 - Válido-Exclusivo
1 - Compartilhado
2 - Sujo-Compartilhado
3 - Sujo

Fig 1 Mecanismo Dragon : Diagrama de estados

4. Prova de Correção do Mecanismo Dragon

A prova de correção de um mecanismo de coerência consiste em verificar que cada solicitação de leitura ou escrita, originada

por qualquer memória cache do sistema, mapeia o conjunto de memórias de um estado coerente para outro estado coerente.

Nesta prova, partiremos da premissa de que antes da operação de leitura/escrita, o estado do sistema, denotado por $\vec{\vartheta}a$, é coerente. Além disto, só precisamos verificar as mudanças de estado originadas pelas solicitações *read miss, write hit* e *write miss*, já que a solicitação *read hit* não gera mudança de estado.

Na prova de correção do mecanismo *Dragon*, utilizamos a seguinte notação :

i : memória cache solicitando a operação
j : posição, na memória global, do dado solicitado

$\vec{\vartheta}a$: estado coerente anterior à solicitação

$\vec{\vartheta}p$: estado posterior à solicitação

$\oplus$: operação entre os elementos de um vetor definida por:

$v \oplus \emptyset = v$
$v \oplus w = w$, onde $v \neq \emptyset$

Nota: Podemos pensar nesta operação como a sobreposição de um valor sobre outro, onde o valor $\emptyset$ é o elemento neutro desta operação. Isto é, depois da operação, o valor do elemento do vetor passa a ser igual ao segundo operando, a menos que este segundo operando seja o elemento neutro, quando então o valor do elemento do vetor não é modificado.

Com o objetivo de simplificar a notação definimos esta mesma operação também entre vetores:

$\vec{\vartheta} \oplus \vec{\emptyset} = \vec{\vartheta}$, onde $\vec{\emptyset}$ corresponde a um vetor com todos os elementos iguais a $\emptyset$

Vale observar que para provar que $\vec{\vartheta}p$ é coerente, basta provar que este vetor é coerente com respeito ao dado solicitado, já que por premissa, os demais dados, não alterados pela solicitação, estão coerentes.

Para simplificarmos a notação, suporemos, sem perda de generalidade, que $j \neq 1,m$ e $i \neq 1,n$.

a) *Read Miss* - esta solicitação pode gerar as seguintes mudanças de estado:

a.1) a *memória global possui a versão mais recente do dado j:*

$\vec{\vartheta}p$ pode ser expresso em termos de $\vec{\vartheta}a$ por :

$$\vec{V}p = \vec{V}a \oplus (\vec{\emptyset}, \ldots, \underbrace{(\emptyset, \ldots, v_j, \ldots, \emptyset)}_{Vc_i}, \ldots, \vec{\emptyset})$$

$$v_{ji}$$

Por premissa, $v_{jk} = \not\in$ ou $v_{jk} = v_j \forall k$, $v_{ji} = \not\in$ em $\vec{V}a$.

Em $\vec{V}p$ temos, $v_{jk} = \not\in$ ou $v_{jk} = v_j \; \forall k \neq i$; $v_{ji} = v_j$ e a memória global tem a versão mais recente do dado j. Portanto, $\vec{V}p$ é coerente.

a.2) *a memória cache k, k ≠ i, possui a versão mais recente do dado j :*

$\vec{V}p$ pode ser expresso em termos de $\vec{V}a$ por :

$$\vec{V}p = \vec{V}a \oplus (\vec{\emptyset}, \ldots, \underbrace{(\emptyset, \ldots, v_{jk}, \ldots, \emptyset)}_{Vc_i}, \ldots, \vec{\emptyset})$$

$$v_{ji}$$

Por premissa, $v_{jy} = \not\in$ ou $v_{jy} = v_{jk} \forall y \neq k$, em $\vec{V}a$, a memória cache k possui a versão mais recente do dado, e existe mecanismo de *difusão* (vide Nota 1 a seguir).

Em $\vec{V}p$ temos, $v_{jy} = \not\in$ ou $v_{jy} = v_{jk} \; \forall \; y \neq k,i$; $v_{ji} = v_{jk} = v_j$; a memória cache k tem a versão mais recente do dado j e existe mecanismo de *difusão* (Vide Nota 1 a seguir). Portanto, $\vec{V}p$ é coerente.

Nota 1 : O mecanismo de *difusão* deste mecanismo de coerência consiste em :

O controlador da memória cache responsável pela última atualização do dado, ao detectar uma solicitação de leitura ou escrita no barramento, fornece ele mesmo uma cópia do dado. No caso de solicitação de escrita, esta memória cache deixa de ser responsável pela última atualização do dado. A versão da memória global só é atualizada quando o dado é substituído, por razões de espaço, na cache responsável pela última atualização.

b) *Write Hit* - esta solicitação pode gerar as seguintes mudanças de estado :

b.1) *a memória global possui a versão mais recente do dado j:*

b.1.1) *a memória cache i é a única memória cache com cópia do dado j :*

$\vec{V}p$ pode ser expresso em termos de $\vec{V}a$ por :

$$\vec{\vartheta}p = \vec{\vartheta}a \oplus (\vec{\emptyset}, \ldots, \underbrace{(\emptyset, \ldots, \overset{v_{ji}}{v_{ji}}, \ldots, \emptyset)}_{Vc_i}, \ldots, \vec{\emptyset})$$

Por premissa, $v_{jk} = \notin \forall\ k \neq i$ em $\vec{\vartheta}a$ e a memória global tem a versão mais recente do dado.

Em $\vec{\vartheta}p$ temos, $v_{jk} = \notin \forall\ k \neq i$, e a memória cache i tem a versão mais recente do dado j e existe procedimento de *difusão* (vide Nota 1 acima). Portanto, $\vec{\vartheta}p$ é coerente.

b.1.2) *a memória cache i não é a única memória cache com cópia do dado j :*

$\vec{\vartheta}p$ pode ser expresso em termos de $\vec{\vartheta}a$ por :

$$\vec{\vartheta}p = \vec{\vartheta}a \oplus (\vec{\emptyset}, \underbrace{(\emptyset, \ldots, \overset{v_{ji}}{v_j}, \ldots, \emptyset)}_{Vc_{k1}}, \ldots, \underbrace{(\emptyset, \ldots, \overset{v_{ji}}{v_{ji}}, \ldots, \emptyset)}_{Vc_{kn}}, \ldots, \vec{\emptyset})$$

onde kx : sequência de memórias cache com cópia do dado j

Em $\vec{\vartheta}p$ temos, $v_{jk} = \notin$ ou $v_{jk} = v_{ji}\ \forall\ k$, a memória cache i tem a versão mais recente do dado j e existe procedimento de *difusão* (Vide Nota 1 acima). Portanto, $\vec{\vartheta}p$ é coerente.

b.2) *a memória cache k possui a versão mais recente do dado j:*

b.2.1) *k = 1 e a memória cache i é a única memória cache com cópia do dado j:*

Mesmo do item b.1.1.

b.2.2) *a memória cache i não é a única memória cache com cópia do dado j:*

$\vec{\vartheta}p$ pode ser expresso em termos de $\vec{\vartheta}a$ por :

$$\vec{\vartheta}p = \vec{\vartheta}a \oplus (\vec{\emptyset}, \underbrace{(\emptyset, \ldots, \overset{v_{ji}}{v_j}, \ldots, \emptyset)}_{Vc_{k1}}, \ldots, \underbrace{(\emptyset, \ldots, \overset{v_{ji}}{v_{ji}}, \ldots, \emptyset)}_{Vc_{kn}}, \ldots, \vec{\emptyset})$$

onde kx : sequência de memórias cache com cópia do dado j

Em $\vec{\vartheta}p$ temos, $v_{jk} = \notin$ ou $v_{jk} = v_{ji}\ \forall\ k$, a memória cache i tem a versão mais recente do dado j e existe procedimento de *difusão* (Vide Nota 1 acima). Portanto, $\vec{\vartheta}p$ é coerente.

c) *write-miss* - esta solicitação pode gerar as seguintes mudanças de estado:

$\vec{v}p$ pode ser expresso em termos de $\vec{v}a$ por :

$$\vec{v}p = \vec{v}a \oplus (\vec{\emptyset}, \underbrace{(\emptyset,\ldots,\underset{v_j}{v_{ji}},\ldots,\emptyset)}_{Vc_{k1}},\ldots,\underbrace{(\emptyset,\ldots,\underset{v_{ji}}{v_{ji}},\ldots,\emptyset)}_{Vc_{kn}},\ldots,\vec{\emptyset})$$

onde kx : sequência de memórias cache com cópia do dado j

Em $\vec{v}p$ temos, $v_{jk} = \not\in$ ou $v_{jk} = v_{ji}$ $\forall$ k , a memória cache i tem a versão mais recente do dado j e existe procedimento de *difusão* (Vide Nota 1 acima). Portanto, $\vec{v}p$ é coerente.

5. Considerações Finais

O objetivo deste trabalho foi apresentar uma metodologia para a prova de correção de mecanismos de controle de coerência orientados a barramento. Esta metodologia consiste na verificação da coerência do estado das memórias do sistema após cada solicitação de leitura ou escrita originada por um processador. Esta metodologia foi exemplificada a partir da prova de correção do mecanismo Dragon. Processo semelhante foi adotado para a prova de correção de mecanismos de controle de coerência do tipo A, como por exemplo o mecanismo Write-Once.

Referências

[ARCH 86] Archibald, J.,Baer, J..'Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model',ACM Trans. on Computers,Vol. 4,No. 4,Nov. 86,pp 273-298

[BRAN 85] Brantley, W. C. et all,'RP3 Processor-Memory Element',IEEE Int. Conf. on Parallel Processing,August 20-23,1985,pp 185-221

[DUBO 82] Dubois, M., Briggs, F,'Effects of Cache Coherency in Multiprocessors',IEEE Trans. on Computers,Vol. C-31,No. 11,Nov. 82,pp 1083-1098

[DUBO 87] Dubois, M.,'Effect of Invalidations on Hit Ratio of Cache-Based Multiprocessors',Proc. of the 1987 Int. Conf. on Parallel Processing,August 17-21,1987,pp 225-257

[DUBO 88] Dubois, M.,'Throughput Analysis of Cache-Based Multiprocessors with Multiple Buses',IEEE Trans. on Computers,Vol. 37,No. 1,Jan. 88,pp 58-70

[DUBO 89] Dubois, M., Wang, J.,'Analytical Modelling of Data Sharing in Cache-Based Multiprocessors',Technical Report No. CENG 89-18,University of Southern California,June 89

[GOOD 83] Goodman, J. R.,'Using Cache Memory to Reduce Processor-Memory Traffic',Proc. of the 10th Int. Symp. on Computer Architecture,IEEE,New York,1983,pp 124-131

[SMIT 82] Smith, A. J.,'Cache Memories',Computing Surveys,Vol. 14,No. 3,Sept. 82,pp 473-530

[VERN 86] Vernon, M., Holliday, M.,'Performance Analysis of Multiprocessors Cache Consistency Protocols using Generalized Time Petri Nets',Proc. of Performance 86 and ACM Sigmetrics 1986 Joint Conf. on Computer Performance Modelling, Measurement and Evaluation, Raleigh, N. C.,May 86, pp 9-17

[VERN 88] Vernon, M., Lazowska, E., Zahorjan, J.,'An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols',Technical Report 88-03-04,University of Washington,March 88