UM MECANISMO INTEGRADO DE CONTROLE DE COERÊNCIA E CONCORRÊNCIA

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 introdução de memórias cache privadas leva à necessidade do gerenciamento do acesso a dados compartilhados, que deve ser realizado por dois tipos de mecanismos de controle de acesso: mecanismo de controle de coerência e mecanismo de controle de concorrência. Na maioria dos multiprocessadores, o controle da coerência dos dados é feito por mecanismos baseados em hardware, enquanto o controle de concorrência é geralmente deixado a cargo do programador ou do compilador. Motivados pelo fato de que um melhor desempenho poderia ser obtido caso estes dois mecanismos fossem executados de forma integrada, estamos propondo neste artigo um ambiente multicache com controle integrado de coerência e concorrência.

ABSTRACT

In cache-based multiprocessors, each processor has an associated private cache memory, where data from global memory is stored. The introduction of local cache memories creates the problem of managing the access to shared data. Two types of control mechanisms are required in order to manage access to shared data in multicache multiprocessors: coherence control and concurrency control. Cache coherence is usually maintained by hardware, while concurrency control is usually left to the programmer or to the compiler. Motivated by the fact that a better performance may be achieved if those two controls are implemented together, we propose in this paper a cache-based multiprocessor with an integrated mechanism for cache coherence and concurrency control.

* 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

Multiprocessadores têm recebido considerável atenção nos últimos anos (RP3 *[PFIS 85]*, CEDAR, ULTRACOMPUTER, etc...*[DONG 89]*). Com respeito à organização da memória, existem duas abordagens diferentes em multiprocessadores: memória compartilhada e memória distribuída. Na abordagem com memória compartilhada, todos os processadores têm acesso à uma memória global, onde todos os dados compartilhados são armazenados, enquanto na abordagem com memória distribuída, todos os dados são distribuídos entre as memórias locais do sistema.

Neste trabalho, nós estamos interessados em multiprocessadores com memória compartilhada. Nestas máquinas, um dos maiores problemas de desempenho consiste na concorrência dos acessos à memória global, denominado problema da interferência da memória. Trabalhos recentes mostraram que uma boa abordagem para minimizar este problema consiste no uso de memórias cache *[SMIT 82, HILL 90]*. Estes estudos propõem uma arquitetura onde cada processador tem uma memória cache local usada para armazenar dados da memória global mais recentemente acessados.

Entretanto, a introdução de memórias cache locais cria o problema do gerenciamento do acesso a dados compartilhados por processos executando em processadores diferentes. Neste gerenciamento, são necessários dois tipos de mecanismo: mecanismo de controle de concorrência e mecanismo de controle de coerência.

Mecanismo de Controle de Coerência [STEN 90] 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 com memórias cache locais, 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. O controle de coerência entre memórias cache é em geral mantido por hardware *[ARCH 86, PAPA 82, GOOD 83, STEN 89, FEIT 90]]*, mas existem mecanismos baseados em software *[BRAN 85, CHEO 88]*.

Mecanismo de Controle de Concorrência [BERN 81, DINN 89, GRAU 90] é um mecanismo que garante o acesso concorrente a dados, por processos executando em um mesmo processador ou em processadores diferentes, sem causar anomalias de sincronização. Exemplos de anomalias de sincronização são: acesso a valores intermediários de um processo e perda de atualização. Em outras

palavras, é um mecanismo que garante que a execução paralela de várias processos gere o mesmo resultado que a execução serial destes processos.

Motivados pelo fato de que um melhor desempenho poderia ser alcançado caso estes dois controles fossem feitos de forma integrada, nós propomos neste artigo uma arquitetura de multiprocessador com controle integrado de coerência e concorrência. Este mecanismo integrado é basicamente feito por hardware, mas requer que o compilador seja capaz de gerar instruções especiais, a partir de comandos especiais da linguagem de alto nível. Em *[DIAS 89]*, Dias et al apresenta, para o contexto de armazenamento de dados compartilhados em memória secundária, uma abordagem centralizada de um mecanismo integrado de controle de coerência e concorrência. Naquele artigo, o mecanismo integrado é baseado em um controlador de coerência e concorrência global, que mantém informações atualizadas sobre os dados compartilhados armazenados nos buffers de E/S dos processadores.

Na seção 2 nós descrevemos a arquitetura do processador que estamos propondo. Na seção seguinte, nós descrevemos os comandos especiais necessários, em uma linguagem de alto nível, para o funcionamento do mecanismo de controle integrado. Na seção 4, exemplificamos a geração do código de um programa, a partir de um programa hipotético, e descrevemos o comportamento do mecanismo integrado a partir da execução de uma instrução deste programa, no ambiente proposto. O mecanismo de controle de coerência integrado, proposto para esta arquitetura, é descrito, na seção 5. A seção 6 conclui este artigo.

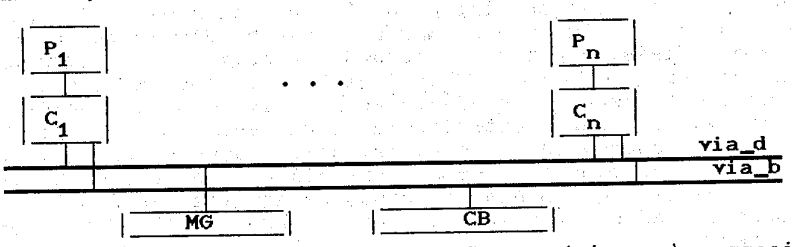
2. Descrição da Arquitetura do Multiprocessador Proposto

A arquitetura do multiprocessador proposto, mostrada na figura 1, é composta de processadores (P), memórias cache privadas(C_i), memória global (MG), controlador de bloqueio (CB), e uma rede de interconexão. A Figura 1 mostra uma arquitetura com uma rede de interconexão específica, composta de dois barramentos, a via_d para dados, e a via_b para informações de bloqueio. Apesar da rede de interconexão não ter que ser orientada a barramento, é necessário que ela tenha capacidade de difusão.

Cada processador tem uma memória cache privada para armazenar os dados compartilhados e não compartilhados, armazenados na memória global.

A memória global armazena dados utilizados por todos os processadores, e o controlador de bloqueio armazena informações de bloqueio referentes a dados compartilhados sendo utilizados por pelo menos um processador. Esta informação de bloqueio é armazenada na *Tabela de Endereços* do controlador, onde em cada

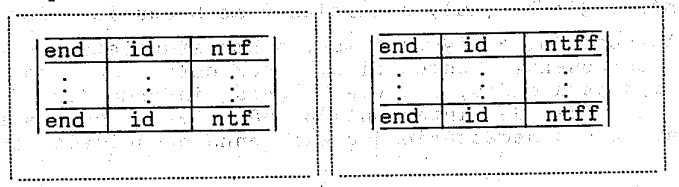
entrada são armazenadas informações que permitem a implementação de uma fila de bloqueio distribuída. Cada entrada na Tabela de Endereços contém o endereço do dado compartilhado, a identificação da tarefa que detem o bloqueio do dado e um contador com o número de tarefas nesta fila, incluindo a tarefa que detem o bloqueio do dado, conforme mostrado na figura 2a. Com esta implementação, o tamanho de cada entrada da Tabela de Endereços é fixo.



P_i: processador i C_i: memória cache associada à P_i
MG: memória global **CB:** controlador de bloqueio
via_d: barramento de dados
via_b: barramento de informações bloqueio

Figura 1. Arquitetura do Multiprocessador Proposto

Como esta Tabela de Endereços não mantém a identificação das tarefas esperando pelo bloqueio do dado, deve ser mantido, em cada memória cache privada, uma *Tabela de Bloqueios Solicitados* que permita ao controlador de cada memória cache detectar, no barramento de informações de bloqueio, que o bloqueio anteriormente solicitado foi concedido. Para isto, cada entrada desta tabela deve conter o endereço do dado solicitado, a identificação da tarefa que solicitou o bloqueio e o número de tarefas executando em outros processadores, esperando pelo bloqueio do dado, que estão na frente da tarefa local, na fila de bloqueio do dado, conforme mostrado na figura 2b.



ntf : número de tarefas na fila de bloqueio

ntff: número de tarefas na fila de bloqueio que estão na frente da tarefa local

Tabela de Endereços Controlador de Bloqueio

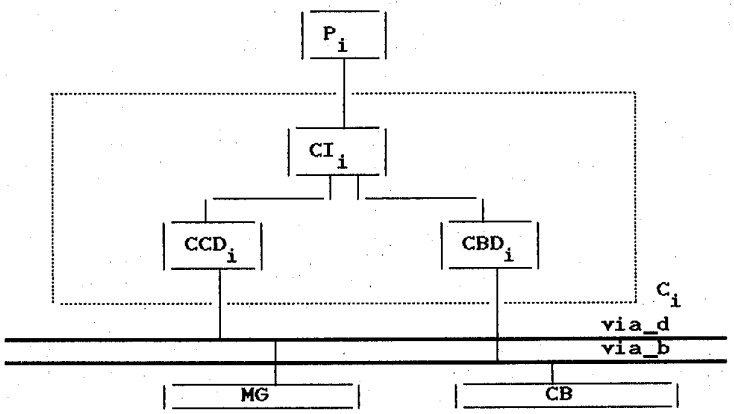
(2a)

Tabela de Bloqueios Solicitados Memória Cache Privada

(2b)

Fig 2. Tabelas Utilizadas no Controle de Concorrência

Cada memória cache privada é composta de três controladores: o Controlador Integrado (CI), o Controlador de Coerência dos Dados (CCD) e o Controlador de Bloqueio dos Dados (CBD), como mostrado na figura 3. Além disto, cada memória cache armazena, para cada linha da cache, uma informação de validade , utilizada pelo mecanismo de controle de coerência, e uma informação de bloqueio , utilizada pelo mecanismo de controle de concorrência. A informação de validade corresponde ao estado do dado na cache que depende do mecanismo de coerência utilizado no sistema, descrito na seção 5. A informação de bloqueio indica se o dado está bloqueado para uma tarefa local (executando no processador associado à cache), para uma tarefa remota ou se está livre. Além disto, se o dado está bloqueado para uma tarefa local, esta informação indica a identificação desta tarefa.



P_i : processador i C_i : memória cache associada processador i

CI_i : controlador integrado

CCD_i : controlador de coerência de dados

CBD_i : controlador de bloqueio de dados

Figura 3. Organização da Memória Cache

O Controlador de Coerência dos Dados é responsável pela manutenção da coerência dos dados armazenados na memória cache, seguindo os procedimentos do mecanismo de controle de coerência descrito na seção 5. O Controlador de Bloqueio dos Dados é responsável pelo controle de concorrência dos acessos a dados compartilhados feitos pelo processador local, conforme exemplificado na seção 4.

O Controlador Integrado é a interface entre as solicitações do processador e os controladores de Coerência e de Bloqueio dos

Dados. Ele é responsável pela integração dos mecanismos de controle de coerência e concorrência, como veremos a seguir.

Quando o Controlador Integrado recebe uma solicitação de leitura/escrita do processador, ele tem condições de determinar o grau de compartilhamento do dado (acesso não compartilhado, acesso exclusivo ou acesso compartilhado), em função do tipo de instrução de acesso, conforme será visto mais adiante. Neste instante, o CI verifica se o dado solicitado está na memória cache associada e:

- se o dado solicitado é um dado privado (acesso não compartilhado), este controlador realiza a leitura/escrita imediatamente;

- se o dado solicitado é um dado compartilhado que só pode ser acessado fora de uma seção crítica (acesso compartilhado), este controlador envia ao Controlador de Coerência de Dados uma solicitação de leitura/escrita seguida da informação da presença do dado na cache;

- se o dado solicitado é um dado compartilhado que só pode ser acessado dentro de uma seção crítica (acesso exclusivo), este controlador verifica se este dado já está bloqueado para a tarefa solicitante, e:
  -- se o dado existe na cache e já está bloqueado para a tarefa em execução, o CI envia ao Controlador de Coerência dos Dados uma solicitação de leitura/escrita seguida da informação de presença do dado na cache;

  -- se o dado existe na cache e não está bloqueado para a tarefa em execução, o CI envia ao Controlador de Bloqueio dos Dados uma solicitação de bloqueio do dado para a tarefa executando e fica esperando a confirmação do bloqueio solicitado;

  -- se o dado não existe na cache, o CI envia ao Controlador de Bloqueio dos Dados uma solicitação de bloqueio do dado para a tarefa em execução e fica esperando a confirmação do bloqueio solicitado;

  -- Ao receber a confirmação do bloqueio solicitado, o CI envia ao Controlador de Coerência de Dados a solicitação de leitura/escrita pendente seguida da informação de existência do dado na cache.

3. Comandos Especiais da Linguagem de Alto Nível

A seguir serão descritos os comandos especiais da linguagem de alto nível, bem como a tradução em linguagem de máquina feita pelo compilador.

a. *Definição de Programa*

Nome Comando : BEGIN_PROGRAM
Sintaxe : BEGIN_PROGRAM <nome do programa>
Semântica : indica o início do programa
Código Gerado : Nenhum

Nome Comando : END_PROGRAM
Sintaxe : END_PROGRAM <nome do programa>
Semântica : indica o final do programa, e que o espaço alocado para as variáveis compartilhadas utilizadas pelo programa, na Tabela de Endereços do Controlador de Bloqueio, deve ser liberado.
Código Gerado : RELLC endereço, para cada endereço correspondente a uma variável com acesso exclusivo.

b. Definição de Precedência

Nome Comando : PRECEDENCE
Sintaxe : PRECEDENCE [<sequência>]
onde <sequência> ::= [<sequência>] |
<sequência> <operador> <sequência> |
identificação da tarefa;
<operador> ::= + | ,;
Semântica : determina a ordem das tarefas definidas no programa. O operador ',' indica que as tarefas podem ser executadas em paralelo, enquanto o operador '+' indica que elas devem ser executadas em série. Por exemplo, o comando PRECEDENCE [t1+[t2,t3]+t4] indica que a tarefa t1 deve ser executada antes das tarefas t2, t3 e t4, que as tarefas t2 e t3 podem ser executadas em paralelo, e que a tarefa t4 deve ser executada após as tarefas t2 e t3.
Código Gerado : Nenhum

c. Definição de Tarefa

Nome Comando : BEGIN_TASK
Sintaxe : BEGIN_TASK <nome da tarefa>
Semântica : indica o início de uma tarefa
Código Gerado : INITTA identificação da tarefa

Nome Comando : END_TASK
Sintaxe : END_TASK <nome da tarefa>
Semântica : indica o final da tarefa, e que todas as variáveis compartilhadas, com acesso exclusivo, referenciadas pela tarefa devem ser liberadas.
Código Gerado : REL endereço, para cada endereço correspondente a uma variável com acesso exclusivo referenciada; e ENDTA nome da tarefa.

d. Definição Variável

Nome Comando : VAR
Sintaxe : VAR <nome das variáveis> [ac,ap] : <tipo das variáveis>
ac : atributo de compartilhamento = {UNSHR,XSHR,NXSHR}. Uma variável do tipo UNSHR não é compartilhada com nenhuma outra tarefa (acesso não compartilhado); uma variável do tipo XSHR é compartilhada por outras tarefas e só pode ser acessada dentro de uma seção crítica (acesso exclusivo), e uma variável do tipo NXSHR é compartilhada mas não precisa ser acessada dentro de uma seção crítica(acesso compartilhado). O valor default deste atributo é o valor UNSHR.

ap : atributo de propriedade = {OWNER,USER}. Este parâmetro só é obrigatório para variáveis compartilhadas. Somente uma tarefa pode ser proprietária de uma variável compartilhada. Este atributo indica o momento no qual deve ser alocado espaço para a variável na Memória Global e no Controlador de Bloqueio, que consiste no início da execução da tarefa proprietária da variável.
Semântica : define o nome da variável, o seu grau de compartilhamento e o momento que ela deve ser criada.
Código Gerado : INITLC endereço, para cada endereço correspondente a variável compartilhada com acesso exclusivo declarada como propriedade da tarefa sendo gerada.

e. Comandos que Necessitam da Leitura de uma Variável

Sintaxe : qualquer comando que necessite da leitura de uma variável. Por exemplo, em um comando do tipo A := B + C, o valor das variáveis B e C precisam ser lidos.
Semântica : depende do comando
Código Gerado : LOAD endereço, se a variável lida é do tipo UNSHR; LOADXSHR endereço, se a variável lida é do tipo XSHR, e LOADNXSHR endereço, se a variável lida é do tipo NXSHR.

f. Comandos que Necessitam da Escrita de uma Variável

Sintaxe : qualquer comando que necessite da escrita e uma variável. Por exemplo, em um comando do tipo A := B + C, o valor da variável A precisa ser atualizado.
Semântica : depende do comando
Código Gerado : STORE endereço, se a variável lida é do tipo UNSHR; STOREXSHR endereço, se a variável lida é do tipo XSHR e, STORENXSHR endereço, se a variável lida é do tipo NXSHR.

4. Exemplo da Execução de um Programa Paralelo

Com o objetivo de exemplificar a tradução em linguagem de máquina feita pelo compilador, mostraremos a seguir o código gerado para um programa paralelo hipotético.

Programa Paralelo

Em linguagem de alto nível	Em linguagem de máquina
BEGIN_PROGRAM exemplo	Corpo do programa:
VAR a:REAL;	
b (XSHR,OWNER):REAL;	INITLC @b
:	:
PRECEDENCE [t1,t2];	RELLC @b
:	
BEGIN_TASK t1;	Tarefa t1
VAR b (XSHR,USER):REAL;	
c (NXSHR,OWNER):REAL;	INITTA t1
a : REAL;	
:	:
a := b + c;	LOADXSHR @b
	LOADNXSHR @c
	STORE (b+c),@a
:	:
END_TASK t1;	
	REL @b
	ENDTA t1
:	
BEGIN_TASK t2;	Tarefa t2
VAR b (XSHR,USER):REAL;	
c (NXSHR,USER):REAL;	INITTA t2
d,e : REAL;	:
:	
b := (d+e)/c;	LOAD @d
	LOAD @e
	LOADNXSHR @c
	STOREXSHR @b
:	:
	REL @b
END_TASK t2	ENDTA t2
:	
END_PROGRAM exemplo.	

Nota : O símbolo @ representa o endereço da variável.

Por razões de espaço, não descreveremos os passos da execução de cada tipo de instrução de máquina gerado pelo compilador. Assim, a título de exemplo, descreveremos a seguir os passos da execução de uma instrução do tipo STOREXSHR endereço, correspondente a uma escrita em um dado com acesso exclusivo.

Ao receber uma instrução STOREXSHR, o Controlador Integrado verifica se o endereço a ser lido existe na cache. Se este endereço existe na cache e está bloqueado para a tarefa em execução, o CI realiza a escrita e muda o estado do dado na cache conforme as regras do mecanismo de coerência integrado descrito no item 5, e passa para a execução da próxima instrução. Se este endereço não existe na cache ou não está bloqueado para a tarefa em execução, o CI solicita ao Controlador de Bloqueio dos Dados a colocação, no barramento de informações de bloqueio, de uma solicitação de bloqueio.

Ao detectar, no barramento de informações de bloqueio, uma solicitação de bloqueio, o Controlador de Bloqueio verifica se a tarefa solicitante já detem o bloqueio do endereço. Caso a tarefa solicitante não detenha este bloqueio, o CB atualiza a fila de bloqueio do endereço solicitado. Se a fila passa a ter somente um elemento ou se a tarefa solicitante já detem o bloqueio do endereço, o CB coloca, no barramento de informações de bloqueio, uma mensagem do tipo LOCKCONF indicando a confirmação do bloqueio solicitado. Caso contrário, o CB coloca, no barramento de informações de bloqueio, uma mensagem do tipo LOCKPOS negando o bloqueio solicitado, e informando o número de tarefas na fila de bloqueio do endereço solicitado.

Ao detectar, no barramento de informações de bloqueio, uma mensagem do tipo LOCKCONF ou LOCKPOS, o Controlador de Bloqueio dos Dados verifica se esta mensagem é relativa a uma tarefa em execução no seu processador local, a partir da identificação da tarefa enviada na mensagem. Caso trate-se de uma tarefa local, se a mensagem é do tipo LOCKPOS, o CBD atualiza a Tabela de Bloqueios Solicitados; se a mensagem é do tipo LOCKCONF, o CBD envia ao Controlador Integrado a confirmação do bloqueio solicitado.

Na Tabela de Bloqueios Solicitados, o campo com o número de tarefas na frente da tarefa em execução é decrementado sempre que o Controlador de Bloqueio dos Dados detecta, no barramento de informações de bloqueio, uma solicitação de liberação do endereço a ser bloqueado. Deste modo, o CBD, ao detectar que não existem mais tarefas na frente da tarefa em execução, na fila de bloqueio, envia ao Controlador Integrado a confirmação do bloqueio solicitado. O CBD deve também, neste instante, enviar ao Controlador de Bloqueio a identificação da tarefa que passou a deter o bloqueio do endereço, já que na Tabela de Endereço do CB, não são armazenadas a identificação das tarefas esperando pelo bloqueio do endereço.

Ao receber uma mensagem de confirmação de bloqueio do Controlador de Bloqueio dos Dados, o Controlador Integrado realiza a escrita e muda o estado do dado na cache conforme as regras do mecanismo de coerência integrado descrito no item 5, e passa para a execução da próxima instrução.

Vale ressaltar que na execução de uma instrução do tipo STORENXSHR ou STORE, o Controlador de Bloqueio não precisa solicitar a confirmação de bloqueio do endereço, já que o endereço a ser atualizado não necessita de acesso exclusivo. Além disto, no caso de uma instrução do tipo STORE, o CI não precisa se preocupar com o controle da coerência do dado, já que o endereço a ser atualizado só é utilizado por uma única tarefa e neste sistema não é possível a migração de processos.

5. Mecanismo de Controle de Coerência Integrado

O mecanismo de controle de coerência integrado proposto, baseado no mecanismo Dragon *[ARCH 86]*, é executado pelo Controlador Integrado em conjunto com o Controlador de Coerência dos Dados. Este mecanismo, baseado na premissa de que dados com acesso exclusivo, dentro de uma seção crítica, só são acessados por uma única tarefa, trata os dados com acesso exclusivo de forma diferente dos dados com acesso compartilhado. As atualizações em dados com acesso exclusivo só são divulgadas às demais memórias cache quando o dado é liberado ou substituído por razões de espaço. Por outro lado, as atualizações em dados com acesso compartilhado são sempre divulgadas. Vale ressaltar que este mecanismo de controle de coerência integrado é muito mais eficiente que um mecanismo de coerência independente do mecanismo de concorrência, já que, com a integração, é possível detectar e eliminar atualizações desnecessárias.

No mecanismo Dragon, existem dois tipos distintos de atualização, uma exclusiva para as memórias cache, onde a memória global não é atualizada, e outra exclusiva para a memória global (em caso de substituição do dado), onde as memórias cache não são atualizadas. De forma a permitir que o tratamento diferenciado entre os dados com acesso exclusivo e os dados com acesso compartilhado seja transparente para o Controlador de Coerência dos Dados, o mecanismo Dragon deve ser modificado no sentido de que a atualização da memória global provoque também a atualização das memórias cache com cópia do dado. Chamaremos este novo mecanismo de Dragon Modificado.

O mecanismo de controle de coerência integrado consiste então na execução do mecanismo Dragon Modificado, descrito a seguir, pelos Controladores de Coerência dos dados do sistema e na execução dos seguintes passos pelos Controladores Integrados do sistema:

- Ao receber uma solicitação de escrita em um dado compartilhado com acesso exclusivo, o CI realiza a escrita na cache e muda o

estado do dado, independente do seu estado anterior, para o estado *Sujo*;
- Ao receber uma solicitação de leitura de um dado com acesso exclusivo, o CI atende esta solicitação imediatamente;
- Ao receber uma solicitação de leitura ou escrita em um dado com acesso compartilhado, o CI envia esta solicitação ao Controlador de Coerência dos Dados;
- Ao receber uma solicitação de liberação de um dado, se o dado existe na cache, o CI envia ao Controlador de Coerência dos Dados uma solicitação de escrita no dado para o valor armazenado na cache, simulando assim uma primeira escrita no dado. Se o dado não existe na cache, o CI não envia nenhuma solicitação ao CCD.

É importante notar que, caso um dado com acesso exclusivo, que já foi atualizado pelo Controlador Integrado, venha a ser substituído antes de ser liberado, não ocorrerá nenhum tipo de inconsistência. Isto porque, como este dado está no estado sujo, a sua substituição provocará a atualização da memória global e consequentemente a atualização das memórias cache.

A descrição do mecanismo Dragon Modificado 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 de Coerência dos Dados, no recebimento de uma solicitação de leitura/escrita recebida do processador associado, ou detectada no barramento de dados. Existem quatro tipos de solicitação a serem tratadas pelo CCD: 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 CCD 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 CCD redireciona a solicitação para o barramento de dados e espera a cópia do dado solicitado.

O mecanismo Dragon Modificado é caracterizado pela difusão de cada escrita feita no dado, para todas as demais memórias cache, e pela atualização da memória global e das memórias cache, 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 de Coerência dos Dados 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 e das demais memórias cache com cópia do dado. 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 e das memórias cache com cópia do dado.

O comportamento do sistema para cada uma das solicitações de leitura/escrita, sob o mecanismo Dragon Modificado, ilustrado na figura 4, é apresentado a seguir:

read miss : Se existe uma versão do dado no estado *sujo* ou *sujo-compartilhado,* o CCD 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 CCD 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 CCD da memória cache solicitante aguarda até poder enviar uma solicitação de escrita às demais memórias cache, pelo barramento de dados, e realiza a escrita. O CCD 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

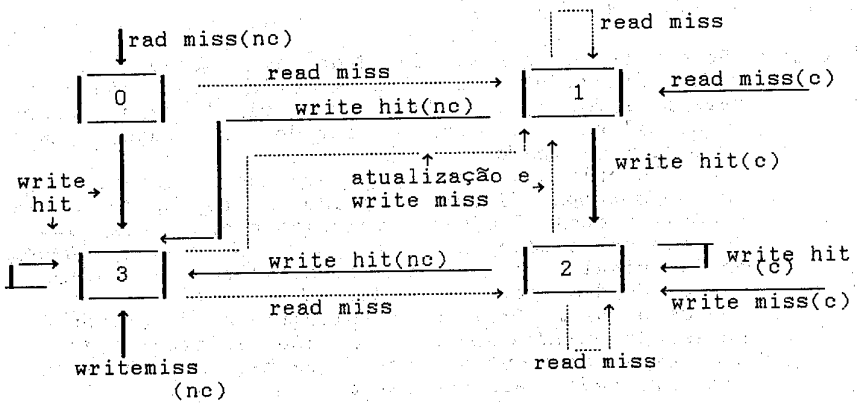
compartilhado.

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 CCD 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 CCD 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 CCD da memória cache solicitante armazena o dado no estado *sujo* e realiza a escrita. Caso contrário, o CCD da memória cache solicitante armazena o dado no estado *sujo-compartilhado* e realiza a escrita.



nc : nenhum CCD enviou sinal pela Linha Compartilhada

c : algum CCD 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 4 Mecanismo Dragon Modificado: Diagrama de estados

6. Considerações Finais

Este artigo propõe uma arquitetura de multiprocessador que contém um controle integrado de coerência e concorrência. Tal mecanismo integrado permite uma maior eficiência do mecanismo de controle de coerência. Além disto, como o controle de concorrência neste mecanismo é baseado em hardware, a tarefa do programador é simplificada, eliminando assim riscos de inconsistências devido ao mau uso de primitivas de sincronização explícitas. Os autores estão neste momento desenvolvendo modelos para quantificar o ganho de eficiência.

A arquitetura proposta é extensível para um grande número de processadores, onde a extensibilidade se dá através de clusters equivalentes à arquitetura proposta neste trabalho. Os diversos clusters se ligam através de uma arquitetura hierarquizada e se comunicam segundo protocolos de controle de concorrência e coerência próprios. A descrição desta arquitetura foge do escopo deste artigo, e será objeto de um próximo trabalho destes mesmos autores.

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

BERN 81] Bernstein, P., Goodman, N.,'Concurrency Control in Database Systems', ACM Computing Surveys, Vol. 13, No. 2, June 81, pp 185-221

[CHEO 88] Cheong, H., Veidenbaum, A. V.,'A Cache Coherence Scheme With Fast Selective Invalidation', Proc. of the 15th Int. Symp. on Computer Architecture, June 88, pp 299-307

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

[DONG 89] Dongarra, J. J., Duff, I. S.,'Advanced Architecture Computers',Technical Report CS-89-90,University of Tenneessee,Nov. 89

[DIAS 89] Dias, D. M. at al,'Integrated Concurrency-Coherence Controls for Multisystem Data Sharing',IEEE Trans. on Software Engineering,Vol. 15,No. 4,April 89,pp 437-447

[DINN 89] Dinning, A.,'A Survey of Synchronization Methods for Parallel Computers', Computer, Vol 22, No. 7, Jul. 89, pp 66-77

[FEIT 90] Feitosa, R. Q.,'O Problema de Coerência de Memórias Cache Privadas em Grandes Multiprocessadores para Aplicações Numéricas: Uma Nova Solução', Anuais do XVII SEMISH, 1990

[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

[GRAU 90] Graunke, G., Thakkar, S.,'Synchronization Algorithms

for Shared-Memory Multiprocessors , IEEE Computer, June 90, pp 60-69

[HILL 88] Hill, M. D.,'A Case for Direct-Mapped Caches', IEEE Computer, Dec. 88, pp 25-40

[PAPA 82] Papamarcos, M. S., Patel, J. J.,'A Low-Overhead Solution for Multiprocessors With Private Ccahe Memories',Proc. of the 11th Int. Symp. on Computer Architecture,IEEE,New York,pp 332-339

[PFIS 85] Pfister, G. at al,'The IBM Research Parallel Prototype(RP3): Introduction and Architecture',IEEE Int. Conf. on Parallel Processing, August 20-23,1985,pp 764-771

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

[STEN 89] Stenstrom,' A Cache Consistency Protocol for Multiprocessors with Multistages Networks', ACM Computer Architecture News, Vol. 17, No. 3, June 89, pp 407-415

[STEN 90] Stenstrom, 'Survey of Cache Coherence Schemes for Multiprocessors', IEEE Computer, June 90, pp 12-24