X Simpósio Brasileiro de Arquitetura de Computadores

Avaliação do Potencial de Técnicas Adaptativas Conjugadas para Software DSMs *

M.C.S. de Castro[‡†] C.L. de Amorim[†]

[‡] Depto de Ciência da Computação/UFJF
[†]COPPE Sistemas/UFRJ
Caixa Postal 68511 - CEP 21945-970 - Rio de Janeiro - RJ

clicia,amorim@cos.ufrj.br

RESUMO

Neste artigo descrevemos um novo modelo de máquina de estados, denominado FSM (*Finite State Machine*) que permite a identificação precisa e detalhada dos padrões de compartilhamento de memória das aplicações quando executadas em sistemas *software DSM*. A FSM é baseada em eventos de coerência registrados em tempo de execução e oferece as vantagens de não aumentar o número de mensagens do protocolo DSM, não ser intrusiva à aplicação e ser transparente ao programador.

Avaliamos a eficácia da FSM em melhorar o desempenho de técnicas adaptativas em TreadMarks utilizando seis aplicações representativas dos *benchmarks* NAS e SPLASH-2. O desempenho dessas aplicações, num sistema simulado com 16 processadores, revelou que os padrões gerados pela FSM permitem que técnicas adaptativas sejam exploradas mais eficazmente por TreadMarks, reduzindo seu overhead em até 58,7%.

ABSTRACT

In this work we describe a new Finite State Machine model (FSM) that enables precise and detailed identification of data sharing patterns for software DSM applications. The FSM is based on coherence events registered at runtime; it is not intrusive and is transparent to the programmer, besides, no additional messages are required for the protocol.

We evaluate the effectiveness of the FSM on improving adaptive techniques for a simulated TreadMarks on six applications of NAS and SPLASH-2 benchmarks. Our performance results, for a cluster of 16 processors, reveal that the FSM allows adaptive techniques to be more effectively explored by TreadMarks, reducing its overheads up to 58.7%.

1 Introdução

Os sistemas *software DSM* (*Distributed Shared Memory*) combinam a facilidade do modelo de programação de memória compartilhada com o baixo custo de sistemas de memória distribuída. Entretando, os problemas principais de sistemas *software DSM* são os *overheads* causados pela comunicação e pelas operações de coerência para manter a memória consistente. O custo de comunicação pode se tornar extremamente alto devido à latência de comunicação da rede de interconexão e da quantidade de mensagens trocadas, que são necessárias para garantir a coerência dos dados compartilhados. Tipicamente, sistemas *software DSM* utilizam a página como unidade de coerência. O grande tamanho da página pode causar dois problemas: falso compartilhamento e fragmentação. O falso compartilhamento ocorre quando dois ou mais processadores realizam acesso a dados não relacionados que estão armazenados numa mesma página, e pelo menos um deles atualiza um dado. A fragmentação ocorre quando há tráfego de dados desnecessário devido à transferência de uma página inteira quando somente seria necessário transferir parte dela.

* Esse trabalho foi parcialmente financiado pela Finep, CNPq e Capes.

Dependendo dos padrões de compartilhamento das aplicações, a fragmentação e o falso compartilhamento podem afetar muito o desempenho de sistemas *software DSM*. Vários estudos foram realizados para minimizar o efeito desses problemas, em particular, avaliação de novos modelos de consistência, diferentes técnicas de busca ou de envio antecipado de dados compartilhados, reestruturação das aplicações, e protocolos adaptativos.

O emprego de modelos relaxados de consistência de memória atrasa a propagação de dados ou ações de coerência até os pontos de sincronização do programa[11, 3, 9], o que alivia os efeitos do falso compartilhamento e da fragmentação. As técnicas de busca ou de envio antecipado, tais como as técnicas de *prefetching*[14] e *forwarding*[15, 16] visam reduzir o tempo de espera pelos dados compartilhados. A modificação ou reestruturação das aplicações permitem melhor localidade e maior compatibilidade da aplicação com o tamanho da unidade de coerência (página), reduzindo a freqüência das ações do protocolo[10]. Protocolos que se adaptam dinamicamente às características das aplicações permitem que sejam implementados diferentes modelos de consistência, diferentes protocolos de coerência (atualização ou invalidação), ou ainda permitem a adaptação entre um ou vários escritores[4, 5, 2, 13].

Neste artigo introduzimos um novo modelo de máquina de estados, denominado FSM (*Finite State Machine*), utilizado para caracterizar o padrão de compartilhamento das aplicações em sistemas *software* DSM. A FSM fornece informações completas sobre os conjuntos de processadores que compartilham cada página ao longo da execução, bem como os tipos de acesso que realizam nas páginas.

A identificação dos padrões de compartilhamento, para uma página ou coleção de páginas, permite a utilização de técnicas adaptativas adeqüadas para cada aplicação em particular. Com o uso da FSM, diferentes técnicas podem ser inseridas nos protocolos e utilizadas em grupos de páginas distintos, tornando o protocolo adaptável ao comportamento dinâmico das aplicações.

Para avaliar o potencial da FSM, ela foi implementada num simulador de TreadMarks [17] e utilizando técnicas adaptativas de acordo com os padrões de compartilhamento produzidos pela FSM, avaliamos o desempenho de um conjunto de seis aplicações. As aplicações utilizadas pertencem aos *benchmarks* SPLASH-2 e NAS. Os resultados são apresentados comparando-se a versão FSM com a versão original.

Nossos resultados mostram que, para as aplicações utilizadas, a FSM é pouco intrusiva, variando em média 2,5% no tempo total de execução em relação ao TreadMarks original. A FSM identifica padrões de compartilhamento simples e repetitivos que permitem classificar as aplicações em regulares, mistas e irregulares, com diferentes ganhos de desempenho. A redução do *overhead* variou entre 10,5% (Water Nsquared) e 58,7% (IS). Para cada aplicação mostramos os resultados das técnicas adaptativas simuladas. Apesar da variedade de técnicas, a adaptação do protocolo foi feita de forma dinâmica, sem a interferência do usuário e sem a necessidade de qualquer modificação nas aplicações.

Este artigo esta organizado da seguinte forma. A próxima seção descreve o sistema DSM TreadMarks e destaca algumas adaptações que podem melhorar seu desempenho. A seção 3 introduz o modelo FSM. A seção 4 discute as técnicas adaptativas que inserimos no TreadMarks. O ambiente de simulação e as aplicações estão descritos na seção 5. A seção 6 analisa os resultados de desempenho obtidos. A seção 7 resume os trabalhos relacionados e apresentamos nossas conclusões na seção 8.

2 Sistema TreadMarks

TreadMarks (TM) é um sistema de memória compartilhada distribuída implementado em *software*. A unidade de coerência é a página, sendo a coerência dos dados mantida com uso do protocolo de invalidações. Este protocolo é implementado através da propagação de notificações de escrita (*write notices*) em operações de sincronização (*locks* e barreiras).

Para reduzir os efeitos de falso compartilhamento, TM fornece suporte a múltiplos escritores através dos mecanismos de *twining* e *diffing*. Inicialmente todas as páginas são protegidas contra escrita. Quando há uma tentativa de atualização numa página, é gerada uma falha de acesso. O TM intercepta esta falha, faz uma cópia da página (*twin*) e a libera para escrita. Quando for necessária a propagação das modificações, TM faz uma comparação entre o *twin* gerado e a versão modificada, e cria um *diff* contendo todas as modificações locais feitas. A cada *diff* existe um *write notice* associado identificando o intervalo e o processador onde o *diff* deve ser criado.

Com o objetivo de reduzir o tráfego de dados pela rede, TM adota o modelo de consistência *lazy release consistency*. A execução de uma aplicação é dividida em intervalos que são iniciados nas sincronizações entre os processsadores. O envio e a aplicação dos *diffs* são atrasados (*lazy*) até os próximos pontos de sincronização.

Nas operações de sincronização as páginas que foram modificadas por outros processadores são invalidadas. Quando ocorre uma falha num acesso à página é necessário buscar um conjunto de *diffs* para torná-la válida. Mais detalhes sobre TreadMarks podem ser obtidos em[12].

TreadMarks fornece suporte a múltiplos escritores visando diminuir os efeitos do falso compartilhamento e da fragmentação. Entretanto, para aplicações onde não há falso compartilhamento o *overhead* associado ao processamento dos *diffs* pode degradar o sistema. Além disso, a busca dos *diffs* pode envolver vários processadores e gerar contenção. Uma das situações mais críticas seria aquela onde todos os processadores simultaneamente requisitam *diffs*.

A identificação de padrões de compartilhamento do tipo um escritor e um ou mais leitores sugere a utilização de uma adaptação onde uma vez identificado um único escritor, é melhor enviar a página do que ter o *overhead* de tratamento de validação da página. Esta adaptação só não surte efeito quando a quantidade de dados atualizados na página é muito pequena. Neste caso pode ser melhor enviar os *diffs* do que a página inteira. Por outro lado, quando são identificados padrões de múltiplos escritores, a técnica de *prefetching* pode ser mais eficaz na antecipação da busca pelos dados, levando a um menor tempo de espera ou no melhor caso, ter os *diffs* disponíveis no momento do acesso. Esta técnica porém, se não tiver uma previsão correta pode degradar o sistema, pois gera interrupções desnecessárias nos processadores remotos, caso os dados trazidos com antecedência sejam invalidados antes de serem utilizados.

3 A Máquina de Estados Finitos (FSM)

O modelo FSM é utilizado para caracterizar o padrão de compartilhamento das aplicações executadas em sistemas *software* DSM. A caracterização do padrão de compartilhamento de cada página é baseada em transições de estado disparadas por eventos utilizados pelo protocolo DSM para manter a memória consistente.

3.1 *Estados e Eventos*

A FSM capta as informações sobre cada página compartilhada durante a execução das aplicações. Ao longo da execução, as páginas mudam o seu estado segundo o tipo de acesso feito à página pelos processadores.

Os acessos a uma página podem ser divididos em acessos dentro e fora de seção crítica e acessos de leitura ou de escrita. A partir desta divisão, definimos os estados de uma página, considerando também a quantidade de processadores. Dessa forma, uma página pode estar em um dentre os seguintes estados para acessos fora de seção crítica: único escritor (SIWR); único leitor (SIRE); único leitor/único escritor (SRSW); único leitor/múltiplos escritores (SRMW); múltiplos leitores/único escritor (MRSW) e múltiplos leitores/múltiplos escritores (WRSH). Em relação aos acessos dentro de seção crítica temos os seguintes estados: único produtor (SIPR); único consumidor (SICO); único produtor/único consumidor (SPSC); único produtor/múltiplos

consumidores (SPMC); múltiplos consumidores (MUCO); múltiplos proprietários (MUOW) e migratória (MIGR). Existem ainda dois estados definidos como *unclassified* (UCLP) e *in-out* (INOUT). Estes estados refletem o estado inicial de uma página e o estado onde ocorrem acessos dentro e fora de seção crítica ao mesmo tempo, respectivamente.

A transição de um estado para outro é provocada pelos eventos utilizados pelo protocolo DSM para manter a memória consistente. Os eventos que disparam as mudanças entre os estados são: falha num acesso de leitura (**F**); violação de proteção de escrita que leva a criação de um *twin* (**T**); invalidação (**I**); aquisição de uma variável de *lock* (**acq**); liberação de uma variável de *lock* (**rel**) e barreira (**B**).

As figuras 1 e 2 mostram, de forma simplificada, alguns dos estados possíveis da FSM e os eventos **T**, **F**, **I** e **acq** que produzem as transições entre esses estados. O evento **B** foi omitido porque as transições provocadas por ele só levam a quatro possíveis estados: UCLP, SIWR, SIRE e MURE. Com relação ao evento **rel**, os estados permanecem os mesmos, exceto o estado INOUT. Este estado, foi também omitido porque vários são os estados que dão origem a ele e uma vez atingido, ele só pode mudar para o estado UCLP, num evento **B** ou num evento **rel**.

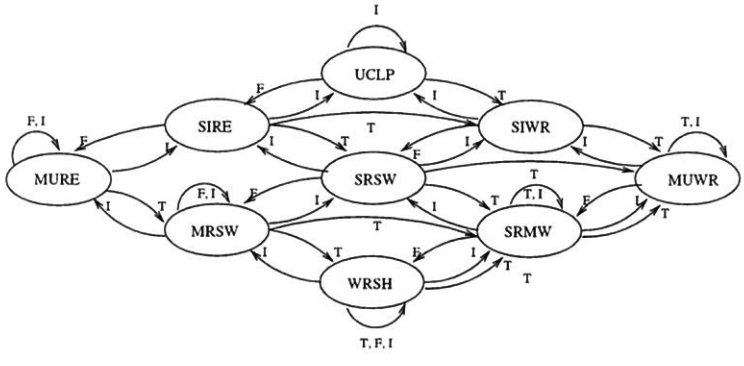

Figura 1: Representação da FSM para estados fora de seção crítica

Várias das possíveis transições de estado relativas aos eventos **T**, **F**, **I** e **acq**, dentro e fora de seção crítica, estão representadas nas figuras 1 e 2 da FSM. Como exemplo, considere uma página no seu estado inicial UCLP. A ocorrência de um evento falha num acesso de leitura (**F**), provoca a transição do seu estado para SIRE. Na ocorrência de um evento **T** (violação da proteção de escrita), fora de seção crítica, pode ocorrer a transição para dois possíveis estados. Transição do estado SIRE para SIWR, se o evento ocorreu no processador leitor ou transição para o estado SRSW se o evento ocorreu num outro processador. Similarmente, para a ocorrência de um evento **F**, dentro de seção crítica numa página UCLP, provoca a transição do seu estado para SICO. Na ocorrência de um evento **T**, pode ocorrer a transição para dois possíveis estados. Transição do estado SICO para SIPR, se o evento ocorreu no processador leitor ou transição para o estado SPSC se o evento ocorreu num outro processador.

Para manter informações sobre os tipos de padrão de compartilhamento que resultam dos acessos dinâmicos às páginas compartilhadas, a FSM possui, associada à cada página, uma estrutura que armazena o estado corrente da página e os conjuntos de processadores relacionados à cada tipo de acesso. Quando ocorre um evento, a identificação do processador é adicionada ao conjunto apropriado.

Cada processador possui um estado individual. As possíveis diferenças de estado entre processadores distintos ocorrem temporariamente e são corrigidas quando há necessidade da

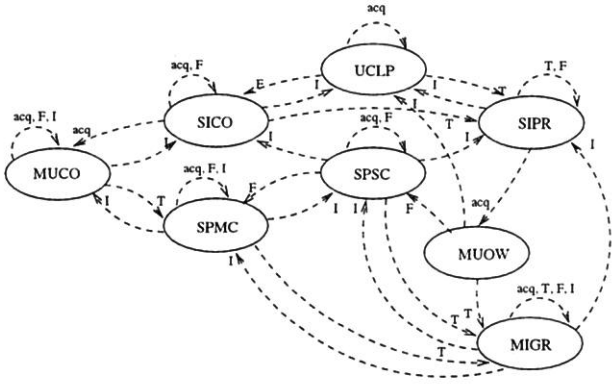

Figura 2: Representação da FSM para estados dentro de seção crítica

realização de operações de coerência.

Uma página pode ter estados diferentes em processadores distintos até que eles se comuniquem e seja computada a combinação de seus estados individuais. Num pedido de página ou de *diffs* os estados individuais são enviados junto com o pedido e a combinação é realizada pelo processador que atende ao pedido. Na barreira, os estados são acoplados à mensagem que informa ao gerente da chegada de cada processador à barreira, e a combinação dos estados individuais é realizada pelo processador gerente. O estado resultante é enviado na mensagem de resposta do processador que atende ao pedido ou na mensagem do gerente na saída da barreira. Desta forma, ao final das operações, os processadores envolvidos possuem o mesmo estado para a página.

A implementação da FSM não necessita de mensagens extras para a captação das informações, porém acrescenta uma quantidade relativamente pequena de informações do estado das páginas. Além disso, pode ser implementada sobre qualquer protocolo DSM.

4 Técnicas de Adaptação

Para demonstrar o potencial da FSM, vamos mostrar como utilizá-la em técnicas de adaptação para o sistema TreadMarks simulado. Para isso, a partir das informações coletadas dinamicamente pela FSM, inferimos o padrão de compartilhamento de cada página e as classificamos em três grupos que denominamos PAGE, HYBRID e DIFF, considerando apenas os acessos de escrita.

O padrão de compartilhamento é inferido monitorando a aplicação. Nas páginas com acessos fora de seção crítica, gravamos seu estado global na barreira até encontrarmos as três primeiras escritas. Nas páginas com acessos dentro de seção crítica, gravamos seu estado parcial na liberação dos *locks* durante os primeiros 16 acessos.

Uma página é classificada como PAGE, se possuir um único escritor, como HYBRID, se possuir mais de um escritor, porém somente um deles acessa à página de cada vez, e como DIFF se possuir vários escritores. Nas três classes podem ou não existir leitores.

Para cada grupo distinto de páginas é possível empregar técnicas de adaptação diferentes. Por exemplo, nas páginas classificadas como PAGE ou HYBRID podemos enviar as páginas em avanço (*forwarding*) quando for conveniente e para páginas classificadas como DIFF podemos

pedir pelos *diffs* antecipadamente (*prefetching*). Nos parágrafos subseqüentes descrevemos as técnicas utilizadas para demonstrar como as informações coletadas pela FSM permitem melhorar o desempenho de aplicações quando submetidas a sistemas *software DSM*, se conhecermos os padrões de compartilhamento em tempo de execução. Todas as técnicas descritas a seguir, começam a interferir no TreadMarks original somente após o padrão de compartilhamento da página ter sido inferido.

Denominamos **SW-MW** a adaptação do protocolo entre um escritor e múltiplos escritores. Numa falha de página, seja de leitura ou de escrita, além das operações normais do Treadmarks, é verificada a classificação da página. Se a classificação for PAGE ou HYBRID e no conjunto de processadores dominantes só houver um processador, o processador que tem a página inválida pede a página ao invés de pedir *diffs*. Caso contrário, o protocolo não sofre nenhuma interferência.

Esta técnica de adaptação visa diminuir o *overhead* relativo aos mecanismos de *twining* e *diffing* utilizados para manter a consistência das páginas. Além disso, não aumenta o número de mensagens em relação ao TreadMarks original. Na falha de página ou é transferida a página ou os *diffs*. Há um aumento do tamanho médio das mensagens transferidas. Um problema com esta adaptação pode surgir se somente poucos dados forem atualizados dentro da página, pois informações desnecessárias são transferidas (página inteira ao invés de um *diff* de muito menor).

A técnica denominada **UP** habilita o protocolo a enviar ou não antecipadamente páginas. Esta técnica só considera as páginas classificadas como PAGE ou HYBRID. O envio antecipado das páginas pode ocorrer tanto nas barreira quanto nos *locks*.

Na saída da barreira, cada processador, para todas as páginas tocadas por ele, verifica a classificação da página. Se a página for do tipo PAGE ou HYBRID, e ele for o único escritor, então, se houver leitores com a página inválida, estas são enviadas antecipadamente. Se houver mais de uma página a ser enviada para um mesmo processador, estas são agrupadas em uma mesma mensagem. Da mesma forma, num pedido de acquisição de *lock*, o processador *owner* quando enviar a posse do *lock* ao processador *acquirer*, verificará as páginas associadas àquele *lock*, as quais ele atualizou, para enviá-las antecipadamente.

Caso uma página classificada como PAGE ou HYBRID, sofra uma falha de página e esta não tiver sido enviada antecipadamente, será enviado um pedido de página.

A técnica **UP** visa diminuir o número de falhas de página e o *overhead* dos mecanismos de *twining* e *diffing*. Além disso, pode aumentar o número de mensagens em relação ao TreadMarks original e aumentar o tamanho médio das mensagens transferidas. Com o envio prévio das páginas, o ganho pode não ser o esperado se as páginas forem invalidadas antes de serem utilizadas, ou porque não chegaram a tempo no seu destino. Neste último caso, o ganho será menor do que o esperado.

A técnica de *prefetching* (**PRF**) visa minimizar o *overhead* da busca de dados remotos, antecipando a busca destes dados. Neste trabalho, ela só é aplicada a páginas classificadas como DIFF. Os dados invalidados por outros processadores, são buscados antecipadamente para que no momento do acesso à pagina, se aumente a chance dos *diffs* já estarem disponíveis. Na nossa técnica de *prefetching* utilizamos uma nova heurística baseada no estado global da página para selecionar quais *diffs* são pedidos com antecedência.

Esta técnica também não visa diminuir o número de faltas de página. Além disso, pode aumentar o número de mensagens em relação ao TreadMarks original. Com a busca antecipada dos *diffs*, o ganho pode não ser o esperado se as páginas forem invalidadas antes de serem utilizadas, ou então porque os *diffs* não chegaram a tempo no seu destino.

A técnica de *pre-diffing* (**PD**) de tenta esconder o *overhead* da criação de *diffs* e, também só considera as páginas classificadas como DIFF. Ao chegar a uma barreira ou requisitar um *lock*, se o processador tiver que esperar que outros processadores cheguem a barreira ou que o *lock* seja liberado, ele então cria os *diffs* com antecedência, para as páginas que estão atualizadas.

Esta técnica pode aumentar o número de falhas de página e degradar o desempenho do sistema se os *diffs* criados não forem utilizados antes que ocorram novas atualizações. Neste caso, deve-se abandonar o *diff* criado e recuperar o *twin* anterior.

5 Metodologia Experimental

Ambiente de Simulação. O simulador utilizado neste trabalho consiste de duas partes, um *front end* e um *back end*. O *front end*, Mint[17], simula a execução de processadores e invoca o *back end* em todas as referências aos dados. Consideramos que a busca de instruções não causa falhas na memória *cache*. O *back end* simula o sistema de memória de forma bastante detalhada. Ele considera *write buffers* e memórias *cache* com tamanho finito, comportamento de *Table Lookaside Buffer* (TLB), emulação completa do protocolo, custos relativos a transferências na rede de interconexão e de acesso à memória, ambos incluindo os efeitos de contenção.

Simulamos uma rede de estações de trabalho com 16 nós. Cada nó consiste de um processador, um *write buffer*, uma *cache* de dados de primeiro nível mapeada diretamente (consideramos que todas as instruções são executadas em um ciclo), um módulo de memória local, e um roteador de rede em malha, utilizando o roteamento *wormhole*.Os parâmetros utilizados em nossas simulações correspondem aos de um sistema real de uma rede de estações de trabalho e podem ser encontrados em[15].

Características das Aplicações. Para se atingir um bom desempenho em sistemas *software DSM* é necessário conhecer o potencial de paralelismo de cada aplicação, suas características de comunicação e de compartilhamento de dados. Nesta seção destacamos em cada aplicação ou núcleo, o tamanho do problema, as operações de sincronização utilizadas, seus padrões de compartilhamento, e os possíveis *overheads* gerados quando os programas são executados em sistemas *software DSM*.

O conjunto de programas utilizado é representativos das áreas científica e de engenharia. Os programas FFT, Ocean, Water Spatial pertencem ao SPLASH-2 *suite*[18]. O programa Em3d foi desenvolvido na Universidade de UC Berkeley [7]. APPBT e IS são programas que pertencem ao NAS *parallel benchmark*[6]. Nas referências citadas podem ser encontradas descrições detalhadas sobre estes programas.

A tabela 1 destaca o tamanho dos problemas simulados para cada aplicação, além da utilização das operações de sincronização, barreira (BAR) e *lock* (L), seus padrões de compartilhamento e a presença de falso compartilhamento (FC) e fragmentação (FRG).

Aplicação	Tamanho	Sincronização	Padrão de Compartilhamento	*Overhead*
Appbt	12 × 12 × 12	BAR e L	MP-MC e migratório	FC e FRG
Em3d	40064 objetos	BAR	1P-MC	FRG
FFT	1M pontos	BAR e L	1P-MC	FRG
IS	1M pontos	BAR e L	migratório	
Ocean	258 × 258	BAR e L	1P-1C	FC e FRG
Water Nsquared	512 moléculas	BAR e L	MP-MC	FC e FRG

Tabela 1: Caracteristicas das Aplicações

6 Avaliação dos Resultados

Esta seção analisa o desempenho das aplicações executadas tanto no TreadMarks original quanto no Treadmarks modificado. A modificação no TreadMarks inclui a FSM juntamente com as

técnicas adaptativas. Os resultados mostrados são para as técnicas adaptativas que produziram os melhores resultados de acordo com o padrão de compartilhamento de dados de cada aplicação.

Na tabela 2 podemos observar que para todas as aplicações, não houve um aumento substancial no tempo de execução (média 2,5%). Comparamos a versão original de TreadMarks com a versão onde foram incluídos a FSM e a computação do padrão de compartilhamento das páginas. Os tempos são mostrados em ciclos de processador. A diferença no tempo de execução variou de -0,6% (Water Nsquared) a 8,8% (Ocean). A variação negativa em Water Nsquared é conseqüência da redução do tempo de espera na barreira. Ocean obteve o maior aumento no tempo de execução por dois motivos: possui uma grande quantidade de páginas compartilhadas e o padrão de compartilhamento de uma quantidade significativa delas gasta mais tempo para ser inferido. As colunas MSG mostram o tamanho médio das mensagens transferidas. Ocean e Em3d são as aplicações que apresentam maior aumento no tamanho médio porque possuem não só uma grande quantidade de páginas compartilhadas como possuem também uma grande quantidade de barreiras. Nas barreiras é computado o estado global de cada página.

Aplicação	TM Original		TM + FSM		Variação (%)	
	Tempo (10^6)	MSG	Tempo (10^6)	MSG	Tempo	MSG
Appbt	478,1	961	478,6	964	0,1	0,4
Em3d	196,4	739	207,0	858	5,1	13,8
FFT	1810,5	2930	1851,4	2977	2,2	1,5
IS	815,5	5464	813,8	5480	-0,2	0,2
Ocean	2098,4	1517	2300,9	1903	8,8	20,2
Water	268,2	1301	266,6	1310	-0,6	0,7

Tabela 2: Tempos de Execução para TreadMarks Original e TreadMarks com FSM

A figura 3 ilustra os resultados do tempo de execução em 16 processadores das aplicações utilizando diagramas de barras. Cada barra mostra o tempo de execução dividido em tempo de computação (*busy*), tempo de espera pelos dados (*data*), tempos dispendidos nas sincronizações de barreiras e *locks* (*synch*), IPC *overhead* (ipc), e outros *overheads* (*others*). Busy representa a quantidade de trabalho útil realizado. *Data* é uma combinação do tempo de processamento de operações de coerência e da latência da rede envolvidos na busca de páginas ou *diffs* como um resultado de violações de acesso às páginas. *Synch* representa os retardos envolvidos na espera em barreiras e *lock acquires/releases*. IPC contabiliza o tempo que o processador gasta servindo pedidos que vêm de processadores remotos. A última subdivisão é composta pelas latências de falhas na *cache* e de falhas na TLB, e pelos tempos de espera para o *write buffer* esvaziar e de interrupção. Todas as comparações são realizadas em relação ao TreadMarks original simulado.

A tabela 3 resume as combinações de técnicas adaptativas que apresentaram os melhores resultados para cada aplicação e a porcentagem de redução total de *overhead* para cada uma delas.

Appbt. Esta aplicação pode ser considerada como mista, pois há dois padrões de compartilhamento predominantes que são 1P-1C e MP-MC. Somente uma página tem padrão migratório. No grupo de páginas MP-MC, os processadores que atualizam a página são sempre os mesmos. Como os padrões são simples e repetitivos, as técnicas de UP e PRF se mostraram bastante eficientes. Os dois padrões predominantes correspondem a 71% de páginas 1P-1C (PAGE) e 29% de páginas MP-MC (DIFF). Em Appbt houve uma redução de 20,5% no *overhead* total utilizando as técnicas UP e PRF conjugadas. Foram reduzidos os tempos dispendidos na espera pelos dados e nas sincronizações.

Em3d. Esta é uma aplicação que pode ser considerada bastante regular. Seus padrões de compartilhamento predominates são 1P-1C e 1P-MC. Estes padrões correspondem a 11% (PAGE)

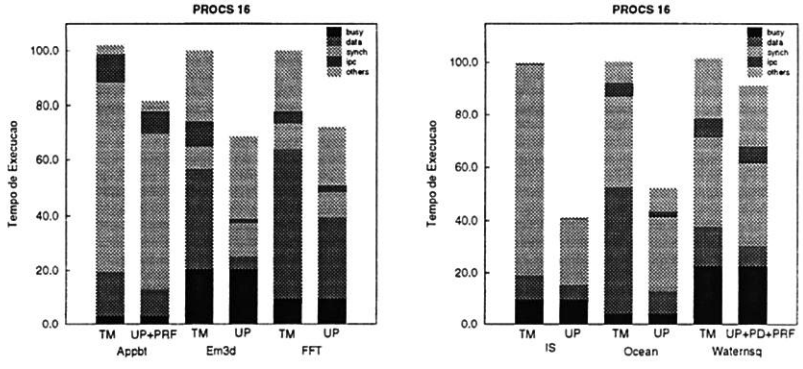

Figura 3: Tempo de execução das aplicações em 16 processadores

Aplicação	Técnica Adaptativa	Redução do *Overhead*
Appbt	UP+PRF	20,5%
Em3d	UP	31,4%
FFT	UP	34,5%
IS	UP	58,7%
Ocean	UP	46,8%
Water Nsquared	UP+PD+PRF	10,5%

Tabela 3: Combinação de Técnicas Adaptativas e Redução Total de *Overhead*

das páginas, e que são responsáveis pelos *overheads* das operações de coerência. Muitas páginas possuem o mesmo conjunto de processadores. Este fato proporciona o envio de várias páginas numa única mensagem, o que diminui o tempo relacionado a ativação e recebimento de mensagens. O restante das páginas 89% têm acesso somente de leitura e não necessitam de operações de coerência. Como os padrões de compartilhamento dominantes são um escritor e um ou vários leitores, a técnica UP é suficiente para se obter uma redução relevante que corresponde a 31% em relação ao TreadMarks original. Em Em3d, foram observadas reduções significativas no tempo de espera pelos dados e no tempo de ipc.

FFT. Esta é outra aplicação que podemos considerar regular, com padrão de compartilhamento predominante 1P-MC. Existe uma quantidade significativa de páginas (33%) onde só ocorrem acessos de leitura. Os 67% restante das páginas são responsáveis pelas operações de coerência. A técnica UP foi suficiente para atingir 34,5% de redução no *overhead* total. Nos tempos relativos a espera pelos dados e ipc é que podemos observar as maiores reduções.

IS. Esta é outra aplicação que podemos considerar regular, com padrão de compartilhamento totalmente migratório, e dentre todas a que apresentou maior redução no *overhead* total. Em 6% das páginas só ocorrem acessos de leitura. Os 94% restante das páginas são responsáveis pelas operações de coerência. Dentre as aplicações, esta é a que possui a maior quantidade de páginas classificadas como HYBRID. A adaptação UP proporcionou 58,7% de redução no *overhead* total. Podemos observar as maiores reduções nos tempos de espera pelos dados e nas sincronizações.

Ocean. Esta aplicação também pode ser considerada regular com relação aos seus padrões de

compartilhamento predominantes que são 1P-1C e 1P-MC. O conjunto de processadores que realizam acesso as páginas são muito coincidentes para grupos de páginas. Da mesma forma que em Em3d, FFT e IS, o envio das páginas pode ser feito numa mesma mensagem. Os padrões predominantes correspondem a 99% do total das páginas. Assim como nas demais aplicações consideradas regulares com padrão 1P-1C e 1P-MC, a adaptação UP foi adeqüada e atingiu 46,8% de redução total no *overhead*. Em Ocean, observamos reduções significativas no tempo de espera pelos dados e no tempo de ipc.

Water Nsquared. Comparando com as demais aplicações, o padrão de compartilhamento mais irregular é encontrado em Water Nsquared. Praticamente todas as suas páginas têm acessos dentro e fora de seção crítica. Das aplicações utilizadas, ela é a que possui uma quantidade significativa em todas as classes de páginas: PAGE 64%, HYBRID 26% e DIFF 10%. As páginas HYBRID e PAGE são responsáveis pela maior parte do *overhead* de coerência. As técnicas UP, PD e PRF proporcionaram uma redução de 10,5% no *overhead* total, sendo esta a menor redução obtida em relação as outras aplicações. No tempo de espera pelos dados observamos a maior porcentagem de redução.

7 Trabalhos Relacionados

Um protocolo que adapta-se ao padrão de compartilhamento um escritor e múltiplos escritores é o proposto por Amza *et al.*[2]. Sua forma de identificação de padrão de compartilhamento utiliza uma única variável de estado para identificar se o padrão é único escritor ou múltiplos escritores. Além disso, neste protocolo há um aumento no número de mensagens transmitidas relativo a posse das páginas. Somente o processador com a posse da página pode realizar atualizações.

O protocolo ADSM[13] desenvolvido simultaneamente a este trabalho, também utiliza um modelo de máquina de estados (SPC) para caracterizar o padrão de compartilhamento das aplicações. A principal diferença entre SPC e FSM é o nível de detalhes das informações obtidas dinamicamente. Em FSM as informações sobre os padrões de compartilhamento de cada página são mais completas e precisas, permitindo potencialmente maior e melhor adaptatividade do protocolo ao comportamento dinâmico das aplicações, já que ADSM implementa somente adaptações entre um e múltiplos escritores e entre atualizações e invalidações, utilizando a caracterização realizada por SPC considerando somente os três últimos acessos.

8 Conclusões

Neste artigo apresentamos o modelo FSM, utilizado para captar informações sobre os padrões de compartilhamento de aplicações executadas sobre sistemas *software DSMs*. Simulamos um sistema de 16 processadores baseado em TreadMarks e os resultados mostraram que com o uso da FSM, obtivemos informações relevantes para a identificação de tais padrões dinamicamente, sem acarretar num *overhead* significativo. Não são necessárias mensagens extras no protocolo.

Nossos resultados também mostraram que com a identificação dos padrões de compartilhamento simples e repetitivos, pudemos interferir no protocolo e utilizar diferentes técnicas adaptativas que melhor se adaptam ao comportamento dinâmico de cada aplicação. Com a adaptação do protocolo feita de forma dinâmica, sem a interferência do usuário e sem a necessidade de qualquer modificação nas aplicações, obtivemos ganhos de desempenho que variaram de 10,5% (Water Spatial) a 58,7% (IS).

Os resultados obtidos até o momento se mostram promissores ao desenvolvimento de protocolos adaptáveis e, sendo assim, estudos adicionais estão sendo realizados para desenvolver técnicas que possam cada vez mais adaptar eficazmente protocolos *software DSM* ao comporta-

mento dinâmico das aplicações.

Agradecimentos. Agradecemos a Raquel Coelho Pinto Gomes por suas explicações sobre a utilização do simulador.

Referências

[1] S. Adve and M. Hill. A Unified Formalization of Four Shared-Memory Models. *IEEE Trans. on Parallel and Distributed Systems*, 4(6), June 1993.

[2] C. Amza, A. Cox, S. Dwarkadas, and W. Zwaenepoel. Software DSM Protocols that Adapt Between Single Writer and Multiple Writer. In *Proc. of the 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA-3)*, pages 261 - 271, Febrary 1997.

[3] B.N. Bershad, M.J. Zekauskas, and W.A. Sawdon. The Midway Distributed Shared Memory System. In *Proc. of the 38th IEEE Int'l Computer Conference(COMPCON Spring'93)*, pages 528 - 537, Febrary 1993.

[4] J.B. Carter, J.K. Bennett, and W. Zwaenepoel. Implementation and Performance of Munin. In *Proc. of the 13th ACM Symp. on Operating Systems Principles*, pages 152–164, October 1991.

[5] S. Dwarkadas, P. Keleher, A.L. Cox, and W. Zwaenepoel. Evaluation of Release Consistent Software Distributed Shared Memory on Emerging Network Technology. In *Proc. of the 20th An. Int'l Symp. on Computer Architecture (ISCA'93)*, May 1993.

[6] D. Bailey et al. The nas Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, March 1994.

[7] D. Culler et al. Parallel Programing in Split-C. In *Proc. of Supercomputing (SC'93)*, pages 262 - 273, November 1993.

[8] L. Iftodé, J. P. Singh, and K. Li. Understanding Application Performance on Shared Virtual Memory Systems. In *Proc. of the 23th An. Int'l Symp. on Computer Architecture (ISCA'96)*, pages 122–133, May 1996.

[9] L. Iftode, J.P. Singh, and K. Li. Scope Consistency: A Bridge Between Release Consistency and Entry Consistency. In *Proc. of the 8th ACM Symp. on Parallel Algorithms and Architectures (SPAA'96)*, June 1996.

[10] D. Jiang, H. Shan, and J. Pal Singh. Application Restructuring and Performance Portability on Shared Virtual Memory and Hardware-Coherent Multiprocessors. In *Proc. of the Sixth ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP'97)*, pages 217 - 229, June 1997.

[11] P. Keleher, A.L. Cox, and W. Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory. In *Proc. of the 19th An. Int'l Symp. on Computer Architecture (ISCA'92)*, May 1992.

[12] P. Keleher, S. Dwarkadas, A.L. Cox, and W. Zwaenepoel. Treadmarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proc. of the 1994 Winter Usenix Conference*, January 1994.

[13] L. R. Monnerat and R. Bianchini. Efficiently Adapting to Sharing Patterns in Software DSMs. In *Proc. of the 4th IEEE Symp. on High-Performance Computer Architecture (HPCA-4)*, pages -, Febrary 1998.

[14] R. Pinto R. Bianchini and C. L. Amorim. Data Prefetching for Software DSMs. In *To appear* Proc. of the Int'l Conference on Supercomputing '98, July 1998.

[15] C. B. Seidel, R. Bianchini, and C. L. Amorim. The Affinity Entry Consistency Protocol. In *Proc. of the 1997 Int'l Conf. on Parallel Processing (ICPP'97)*, pages 208 - 217, August 1997.

[16] P. Trancoso and J. Torrellas. The Impact of Speeding up Critical Sections with Data Prefetching and Forwarding. In *Proc. of the 1996 Int'l Conf. on Parallel Processing (ICPP'96)*, pages -, August 1996.

[17] J. E. Veenstra and R. J. Fowler. MINT: A Front end for Efficient Simulation of Shared-Memory Multiprocessors. In *Proc. of the 2nd Int'l Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1994.

[18] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH2 Programs: Characterization and Methodological Considerations. In *Proc. of the 22th An. Int'l Symp. on Computer Architecture (ISCA'95)*, May 1995.