O Software do Computador Paralelo NCP_2 da COPPE/UFRJ *

L. Whately, R. Pinto, G. Silva, M. Hor-Meyll,
M. De Maria, J. Barros Jr., R. Bianchini e C. L. Amorim

COPPE Sistemas e Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil 21945-970
FAX/Tel.: (021)590-2552

Relatório Técnico ES-395/96, Junho 1996, COPPE/UFRJ

Resumo

O sistema paralelo NCP_2 em desenvolvimento na COPPE Sistemas/UFRJ introduz suporte de *hardware* simples e de baixo custo para implementação eficiente de protocolos de coerência de dados em *software*, os chamados *software DSMs*. Tal suporte de *hardware* consiste em controladores de protocolo programáveis que permitem a implementação de técnicas de tolerância à latência de comunicação e a *overheads* de processamento de coerência. Nesse artigo descrevemos os aspectos de *software* do NCP_2, dando ênfase a como os controladores de protocolo podem ser utilizados, ao código a ser executado nos controladores, e às estatísticas sobre sua utilização. Baseados nessas estatísticas, determinamos as transações de protocolo que necessitam de maiores otimizações e justificamos a implementação corrente do controlador de protocolos.

Abstract

The NCP_2 parallel system under development at COPPE Sistemas/UFRJ introduces the use of simple and low-cost hardware to support the implementation of efficient software DSMs. This hardware consists of a programmable protocol controller that allows for the implementation of overlapping techniques for minimizing the impact of comunication and coherence overheads. In this paper, we describe the NCP_2's software, concentrating on the functionality provided by the protocol controller, its software, and the statistics of its utilization. Based on these statistics, we determine the protocol transactions that need optimization and justify the current design of the protocol controller.

*Esta pesquisa foi financiada pela FINEP, FAPERJ e CNPq.

1 Introdução

Os sistemas de memória compartilhada distribuída (DSM) implementados exclusivamente em *software* (*software DSMs*) provêem aos programadores uma abstração de memória compartilhada sobre uma arquitetura que só permite a passagem de mensagens. Esses sistemas oferecem uma alternativa de baixo custo para a computação usando memória compartilhada, pois podem ser construídos com estações de trabalho e sistemas operacionais existentes no mercado. Entretanto, várias aplicações executando sobre *software DSMs* sofrem uma alta latência de comunição e *overheads* induzidos pelas tarefas relacionadas com a coerência de memória.

O projeto NCP_2 propõe o uso de um controlador de protocolos (CP) programável em cada nó de processamento para esconder ou tolerar o *overhead* dessas tarefas de comunicação e coerência [ABS+96, BKP+96]. Neste artigo descrevemos em detalhe os aspectos de *software* do sistema paralelo NCP_2, dando ênfase a como os CPs podem ser utilizados pelos protocolos de coerência, ao código a ser executado nos CPs, e às estatísticas sobre a utilização de seus comandos.

No sistema NCP_2, o processamento do protocolo de coerência de dados compartilhados fica dividido entre o processador principal e seu CP associado. O CP geralmente implementa as funções básicas necessárias a *software DSMs*, enquanto que o processador principal retém as tarefas mais complexas e de processamento rápido (tais como manipulação de listas encadeadas) associadas ao protocolo. O controle do protocolo fica então com o processador principal, o qual envia comandos ao seu CP.

Os *software DSMs* desenvolvidos para o sistema NCP_2 utilizam uma biblioteca de chamadas específicas para a comunicação com o CP. Estas chamadas constroem os comandos a serem enviados e acessam, quando necessário, estruturas de dados no CP. Não existe intervenção do sistema operacional na comunicação do protocolo com o CP. A memória do CP é mapeada diretamente na área da aplicação.

Nossa proposta inicial para o código do CP é baseada em dois tipos diferentes de *software DSMs*, ambos utilizando variações de *Release Consistency* [GLL+90]. A principal diferença entre esses dois tipos de sistema é o uso de *home nodes* para os dados compartilhados. As características desses *software DSMs* determinam então que os comandos do CP estejam nos seguintes grupos: a) transferência e manipulação de dados compartilhados; b) manipulação de diretórios; e c) troca de mensagens.

Nosso estudo da utilização desses comandos mostra que é importante a minimização de interferências entre processadores executando *software DSMs* através do uso de uma unidade de processamento adicional como o nosso CP. Além disso, concluímos ser importante a otimização das operações de manipulação de atualizações a páginas compartilhadas através de *hardware* dedicado.

O restante do artigo está organizado da seguinte forma. A seção 2 descreve o *hardware* do CP. A seção 3 apresenta as características principais dos protocolos que inspiraram a funcionalidade dos CPs. Os comandos, estruturas de dados e o código executado no CP são descritos em detalhe na seção 4. A seção 5 apresenta as estatísticas de utilização dos comandos, e fechando esse artigo,

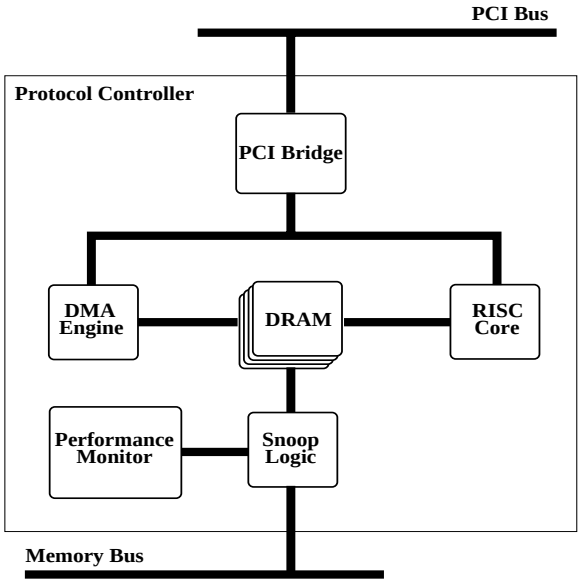

Figura 1: Controlador de Protocolo

a seção 6 conclui o nosso trabalho.

2 O Controlador de Protocolos

As funções do controlador de protocolo foram definidas, abstratamente, de modo a fornecer ao processador principal suporte para os mecanismos básicos usados pelos *software DSMs*, que são: pedido e resposta de página remota; pedido e resposta de modificações (também chamadas "diffs") a páginas remotas; criação e aplicação de diff; recepção e envio de mensagens; manipulação de diretórios. As tarefas restantes, usualmente mais complexas e de rápido processamento, continuam a ser executadas no processador principal.

Os diffs são retratos que refletem as modificações efetuadas nas páginas existentes nos diversos processadores do sistema. Normalmente um diff é obtido pela comparação palavra por palavra da página com uma cópia mais antiga salva anteriormente. A criação, transferência e aplicação de diffs são algumas das tarefas mais custosas em *software DSMs*. Essas operações possuem custo significativo, e a obtenção de um mecanismo simples e eficiente para processá-las foi um dos alvos principais de nossos estudos na elaboração do CP.

Procuramos elaborar um *hardware* para o CP que executasse as tarefas acima mencionadas de maneira mais eficiente e simples possível. Assim, projetamos o CP para incluir um microprocessador de propósito geral e *hardware* específico para a manipulação eficiente de diffs.

Conforme vemos na figura 1, o CP possui seis módulos: processador local, memória local, interface com o barramento PCI, lógica de *Snooping*, monitor de desempenho, e uma máquina *DMA*. O processador local é responsável pela interpretação e execução dos comandos que chegam ao CP e não utilizam *hardware* dedicado. A memória local se destina a armazenar estruturas de dados para comunicação com o processador principal. A *interface* com o barramento PCI, além

das funções de acesso ao PCI, faz o controle da memória local e realiza as operações de coerência da cache do processador local. A lógica de *Snooping* está conectada ao barramento do processador principal e monitora os acessos de escrita realizados às paginas compartilhadas. Os endereços das escritas que são realizadas pelo processador principal em páginas compartilhadas são anotados em uma memória de vetores de bits, onde cada bit representa 4 bytes da memória principal. O monitor de desempenho vai permitir a coleta de dados para avaliar o comportamento do CP. O *DMA* realiza transferências de páginas ou diffs. Os endereços a serem transferidos em um diff são determinados pelo conteúdo da memória de vetores ou por um vetor fornecido pelo *software DSM*. Maiores detalhes com relação ao *hardware* do CP podem ser encontrados em [SHMM+96].

3 Software DSMs

Vários sistemas *software DSM* usam as opções de proteção de páginas na memória virtual para implementar a consistência de memória, a nível de páginas, no protocolo de coerência. Os sistemas *software DSM* modernos buscam manter a consistência das páginas de memória apenas nos pontos de sincronismo e permitem que múltiplos processadores escrevam na mesma página concorrentemente.

Duas categorias de *software DSMs* são assumidas como básicas para a definição da funcionalidade do CP do NCP_2. Ambas implementam variações do protocolo *Release Consistency* [GLL+90]. A principal diferença entre as duas categorias é a existência de *home nodes* para as páginas. Os protocolos na primeira categoria podem ser relativamente simples, pois eles usam *home nodes* para manter informações de diretório e como fonte de versões atualizadas das páginas armazenadas. Os protocolos na segunda categoria não assumem *home nodes* e por isso devem ser capazes de reconstruir as páginas através do conjunto de modificações enviadas por vários processadores.

No protocolo baseado em diretórios, que chamaremos de protocolo Directory, na primeira escrita para uma página *read-only*, o processador cria uma cópia gêmea da página e envia um *write-notice* para todos os outros processadores compartilhando a página a ser modificada. No momento de uma liberação de *lock*, as modificações feitas pelo processador que libera o *lock*, são coletadas em forma de diffs e enviadas para as *home nodes* correspondentes das páginas. No próximo ponto de aquisição de *lock*, o processador que adquire o *lock*, invalida todas as páginas para as quais ele recebeu *write-notices*. No momento de um *page fault*, este mesmo processador pede uma versão atualizada da página para o respectivo *home node*. A informação de diretório para uma página inclui uma lista de processadores que compartilham a página, e uma indicação do estado de página, que pode ser a seguinte: a) Não-Cacheada - nenhum processador está usando a página; b) Compartilhada - um ou mais processadores estão usando a página em modo *read-only*; c) Suja - um processador está usando a página em modo *read-write*; d) Fraca - dois ou mais processadores estão usando a página, e pelo menos um está usando a no modo *read-write*.

TreadMarks é o sistema tomado como exemplo da segunda categoria de protocolos. Como na primeira categoria, em TreadMarks, a invalidação de páginas ocorre nos pontos de aquisição do

lock, enquanto que as modificações para as páginas invalidadas são coletadas no momento de um *page fault*. As modificações que o processador que adquire o *lock* deve coletar são determinadas através da divisão do tempo de execução em intervalos e criando-se um vetor de *timestamps* para cada um dos intervalos. O vetor de *timestamps* descreve uma ordem parcial entre os intervalos de diferentes processadores. Antes que o processador possa continuar a execução de seu programa, as atualizações de intervalos com vetor de *timestamp* menores que vetor de *timestamp* corrente do processador que adquire o *lock*, devem ser coletadas. O proprietário anterior do *lock* é responsável por comparar o vetor de *timestamp* corrente do processador que adquire o *lock*, com o seu próprio e enviar os *write-notices* necessários, indicando que uma página foi modificada num intervalo particular. Quando uma *page fault* ocorre o processador consulta sua lista de *write-notices* para descobrir que diffs ele precisa para atualizar a página. Uma descrição mais detalhada de TreadMarks pode ser encontrada em [KDCZ94].

No sistema NCP_2, dividimos o protocolo de coerência entre o processador principal e o CP. O CP realiza as tarefas básicas necessárias aos sistemas *software DSMs*. O processador principal possui o controle do protocolo e determina a execução de tarefas no CP através do envio de comandos. O CP que propomos oferece várias possibilidades para sobrepor *overheads* e computação útil. O CP concentra-se em três diferentes formas de realizar esta tarefa:

- As tarefas básicas de coerência podem ser executadas no CP em paralelo com computação útil no processador principal. Além disso, grande parte da comunicação remota e pedidos relativos a coerência podem ser realizados pelo CP sem sequer interromper seu processador associado.

- Buscas antecipadas de páginas, entradas de diretório, e diffs podem ser realizadas prevendo-se os acessos futuros para o dado compartilhado baseado na história passada.

- Diffs podem ser criados dinamicamente ao mesmo tempo que as páginas correspondentes são modificadas.

Na nossa versão modificada do protocolo Directory, o processador só é interrompido quando uma resposta para um pedido de entrada de diretório é retornada do *home node* ou quando um *write-notice* é recebido. A recepção de um *write-notice* requer a intervenção do processador principal, para que ele possa atualizar as estruturas de dados do protocolo. Um *write-notice* apresenta-se para os CPs como uma mensagem regular que requer uma interrupção no processador do nó destino e o envio de um *acknowledge* para o nó remetente.

Na versão modificada de TreadMarks, o processador principal deve ser interrompido em várias outras circunstâncias. Um pedido local de criação de diff deve interromper o processador para verificar se essas modificações ainda não foram processadas. Os pedidos remotos de diff também devem interromper o processador para que o protocolo crie um novo intervalo.

A busca antecipada de dados remotos é implementada nesses protocolos executando todas as transações que normalmente ocorrem nos *page faults*, nos pontos de aquisição de *lock*. Isto

é, logo após as páginas serem invalidadas. Assumimos que dados invalidados em um ponto de sincronização provavelmente serão acessados no trecho de código que segue a sincronização.

A criação de diffs ao mesmo tempo que os dados compartilhados são modificados é suportada diretamente pelo CP, e não apenas elimina a necessidade do uso de cópias gêmeas para as páginas modificadas, mas também reduz significativamente o tempo de geração de diff e o uso do barramento local.

4 O *Software* do Controlador de Protocolos

Essa seção descreve o *software* do CP, suas estruturas de dados, e apresenta uma descrição detalhada dos comandos executados pelo CP.

4.1 Estrutura de Dados

As principais estruturas de dados utilizadas pelo CP são as seguintes:

- Tabela de conversão de endereços
- Contador geral
- Tabela de diretórios
- Fila de comandos
- Buffer circular
- Variáveis de *lock*

A tabela de conversão de endereços é usada pelo CP para a conversão de endereço virtual para o endereço físico das páginas compartilhadas. Nessa tabela também encontram-se vários contadores e bits sinalizadores para cada página. Esses bits sinalizam os pedidos pendentes e os pedidos prontos sem interromper o processador.

Um pedido remoto pendente é armazenado na tabela de conversão de endereços até que a operação se complete. Nesse ponto, o CP ativa um bit na memória compartilhada para sinalizar o término da operação. Os pedidos pendentes e os bits sinalizadores são usados pelo processador nos *page faults*, para indicar que não é preciso repetir um pedido que está pendente ou já foi terminado. Os bits e contadores são:

- Contador diff/pag - usado para controlar o número de pedidos pendentes de diff ou sinalizar o pedido pendente de página.
- Contador write-notice - usado para controlar o número de *write-notices* pendentes.
- Bit diff/pag - sinaliza que todos os diffs já foram aplicados ou que uma página pedida está na memória.

- Bit write-notice - sinaliza que os *acknoledgements* para todos os *write-notices* enviados já foram recebidos.

O contador geral sinaliza quantos pedidos pendentes relativos a diffs, páginas ou *write-notices* existem. A tabela de diretórios armazena informações de compartilhamento das páginas em uso. Além disso, cada entrada nessa tabela contém um ponteiro para uma lista de processadores aguardando acesso exclusivo à entrada. O CP recebe os pedidos locais ou remotos através da fila de comandos. Os comandos são enfileirados para serem executados pelo CP. O buffer circular armazena temporariamente os diffs e páginas recebidos dos processadores remotos, antes de serem aplicados pelo DMA do CP. O sincronismo dos acessos concorrentes entre o processador e o CP às tabelas e à fila de comando é controlado pelas variáveis de *lock*.

4.2 Comandos

Como descrevemos anteriormente, para o CP suportar uma funcionalidade básica para o desenvolvimento de diferentes tipos de protocolos, esse possui um conjunto de comandos que podem ser separados nas seguintes categorias:

4.2.1 Comandos Relacionados a Diffs

- Geração de diff - O CP gera o diff correspondente a página e o armazena na memória principal no endereço indicado pelo comando. Esse comando é um pedido do processador do nó local.

- Pedido de diff para um nó remoto

- Pedido de diff vindo de um nó remoto - O CP recebe um pedido de diff vindo de um nó remoto. O comando pode sinalizar, através de bits de interrupção (ver descrição na seção 4.3), se o CP deve interromper o processador. Se a interrupção é necessária, o CP espera que o processador coloque na fila de comandos os pedidos de envio de diff para o nó remoto e o avise para prosseguir. Caso esse bit de interrupção não esteja ativado, o CP gera o diff e o envia para o nó remoto sem a intervenção do processador.

- Envio de diff - O CP recebe o pedido de envio de um diff, já gerado, do processador do nó local. O endereço da máscara de memória e dos dados é passado no comando, assim como o nó destino e a identificação da página. Esse pedido gera um comando para a aplicação do diff no nó remoto.

- Aplicação de diff vindo de um nó remoto - Ao executar esse comando o CP aplica o diff vindo do nó remoto na área de memória correspondente à página, de acordo com o vetor de bits incluído no comando.

4.2.2 Comandos Relacionados a Páginas

- Pedido de uma página remota - O CP gera um pedido de página para o nó remoto. O endereço físico correspondente à página no nó local é enviado junto ao pedido da página, possibilitando que a página, ao retornar, seja escrita sem haver a necessidade de consultas à tabela de conversão de endereços.
- Pedido de uma página local vindo de um nó remoto
- Envio de uma página para um nó remoto
- Recepção de uma página vinda de um nó remoto
- Envio e recebimento de *write-notice* - Ao receber do processador local o pedido de envio de *write notice*, o CP o envia para o nó destino com a identificação da página correspondente. Recebendo de um nó remoto, o CP interrompe o processador para processar o *write-notice*, e o processador avisa ao CP o término desse processamento.

4.2.3 Comandos para a Manipulação de Diretórios

- Pedido de entrada de diretório - O processador envia esse comando para ter acesso exclusivo à linha de diretório correspondente à página desejada. O CP compara a identificação do nó destino com a identificação do nó local, caso sejam iguais, o processamento é feito localmente. Caso contrário, é gerado o pedido de entrada de diretório remoto.
- Recepção de pedido de entrada de diretório vindo de um nó remoto - O CP após conseguir acesso exclusivo à linha de diretório desejada, a envia para o nó remoto que fez o pedido.
- Atualização de entrada de diretório vindo de um nó remoto - O CP escreve na tabela de diretórios a linha de diretório recebida do nó remoto. Logo após, o pedido de entrada de diretório é retirado da fila de pedidos pendentes. Se o CP verifica que ainda existe um pedido pendente, ele envia a entrada ao nó que estava esperando.

4.2.4 Comandos para a Transferência de Mensagens

- Envio de mensagem - O CP envia uma mensagem para um nó remoto.
- Recepção de mensagem - O CP recebe uma mensagem de um nó remoto. O CP sinaliza o processador sobre o recebimento da mensagem.

4.3 Bits de interrupção e acknowledgement nos comandos

Um dos objetivos do CP é interromper o processador o menos possível. Entretanto, como mencionado anteriormente, certas operações podem necessitar da assistência do processador. O CP oferece diferentes opções para a interrupção do processador em forma de bits sinalizadores associados aos comandos. O comando pode pedir a interrupção do processador quando chega ao topo

Aplicação	Tamanho da Entrada
TSP	18 cidades
Barnes	4K partículas
Water	512 moléculas
Em3d	40064 nós, 10% remoto

Tabela 1: Aplicações e Tamanho das Entradas.

da fila de comandos no CP ou logo depois de ser executado. Uma outra opção é a possibilidade de sinalizar, no comando, que o CP deve enviar um *acknowledgement* para o nó remoto, após esse comando terminar sua execução.

5 Estatísticas de Utilização de Comandos do CP

Através da execução de várias aplicações reais sobre os protocolos simulados (TreadMarks e Directory modificados), podemos determinar a utilização dos comandos executados no controlador de protocolo. Esses resultados nos permitem demonstrar a funcionalidade básica dos comandos e justificar o suporte de *hardware* a certas transações de *software DSMs*.

Utilizando um simulador baseado no interpretador Mint [VF94], modelamos 16 nós do NCP_2 em detalhe, assumindo um tamanho de página de 4 KBytes. Simulamos quatro aplicações: TSP, Barnes, Water e Em3d. TSP é da *Rice University* e faz parte do pacote do TreadMarks. As duas aplicações seguintes são do Splash-2 suite [WOT+95] e a última aplicação, Em3d [C*et al.*93], tem origem na *University of California at Berkeley*. A tabela 1 lista as aplicações e seus tamanhos de entrada.

TSP usa um algoritmo *branch-and-bound* para descobrir o custo mínimo de se percorrer 18 cidades. Barnes simula a interação de um sistema de 4K corpos sob a influência de forças gravitacionais para 4 passos, usando o método *Barnes-Hut hierarchical N-body.* Water é uma simulação dinâmica de moléculas, que calcula forças inter e intramoleculares em um conjunto de 512 moléculas de água. Essa simulação utiliza-se um algoritmo $\mathcal{O}(n^2)$ para a computação das interações. Em3d simula a propagação de ondas eletromagnéticas em objetos 3D. Simulamos 40064 objetos elétro-magnéticos conectados aleatoriamente, com uma probabilidade de 10% de que objetos vizinhos residam em nós distintos. As interações entre objetos são simuladas durante 6 iterações.

As figuras 2 a 5 mostram as aplicações executando sobre a nossa adaptação do protocolo TreadMarks. As figuras mostram o número total de operações de cada tipo executadas para cada aplicação. Os códigos das operações e suas descrições encontram-se na tabela 2.

Observamos que as aplicações sobre TreadMarks apresentam comportamentos comuns, como a concentração do número de operações nas transações de diff (códigos 7 a 10) e o pequeno número de transferências de páginas (códigos 1 a 3). Esses números são esperados para TreadMarks, pois

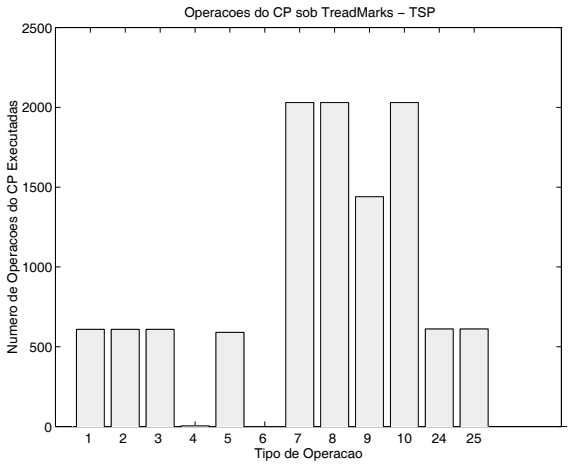

Figura 2: Operações do Controlador de Protocolo sob TreadMarks - TSP

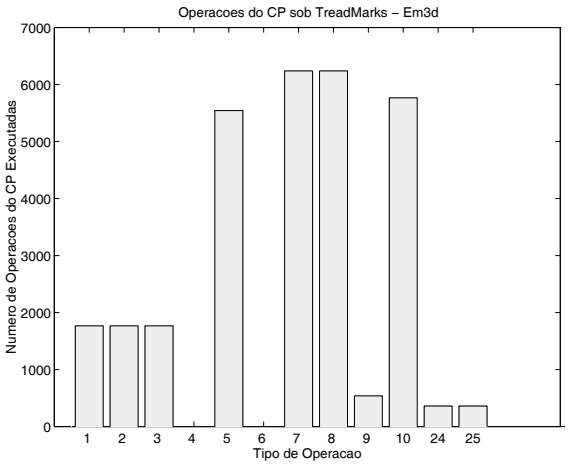

Figura 3: Operações do Controlador de Protocolo sob Treadmarks - Em3d

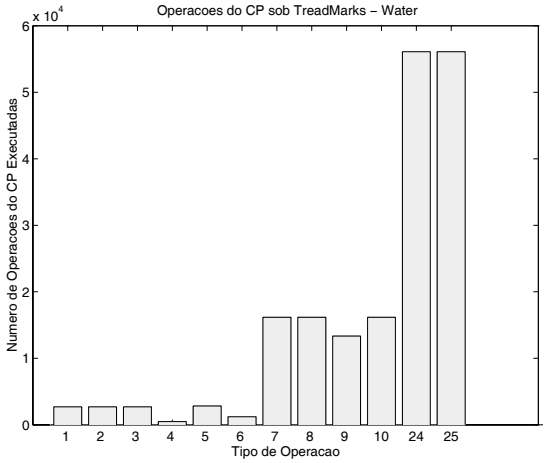

Figura 4: Operações do Controlador de Protocolo sob TreadMarks - Water

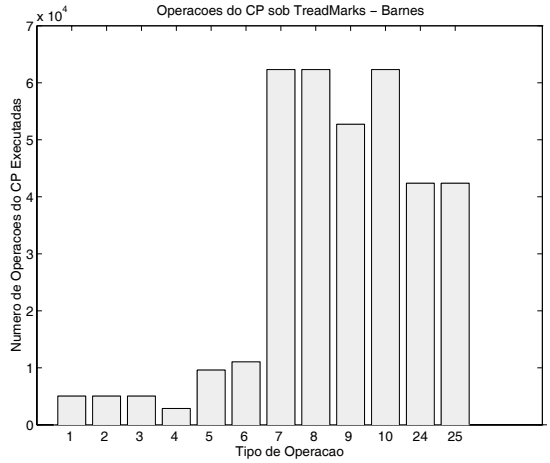

Figura 5: Operações do Controlador de Protocolo sob Treadmarks - Barnes

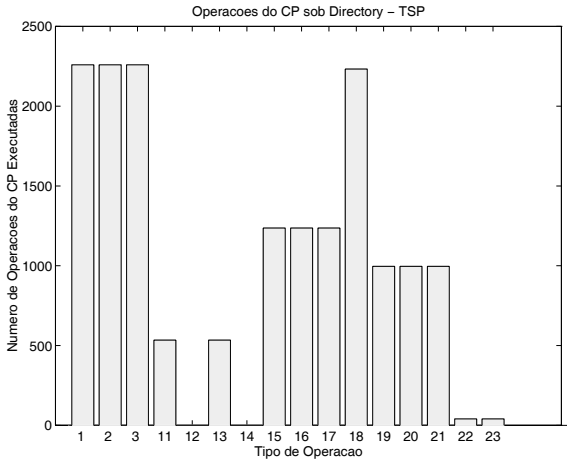

Figura 6: Operações do Controlador de Protocolo sob Directory - TSP

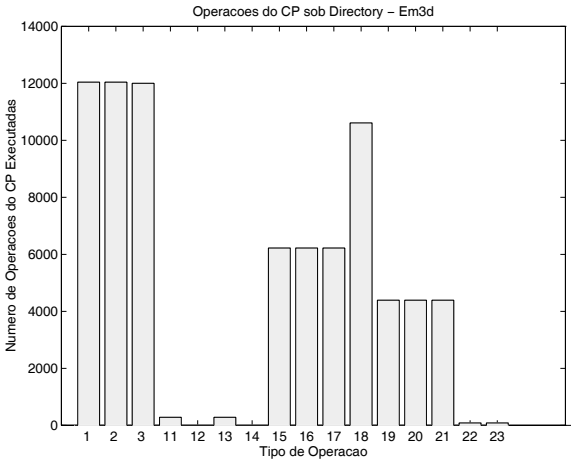

Figura 7: Operações do Controlador de Protocolo sob Directory - Em3d

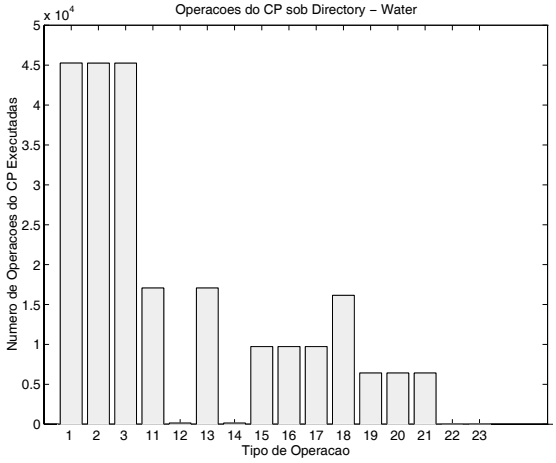

Figura 8: Operações do Controlador de Protocolo sob Directory - Water

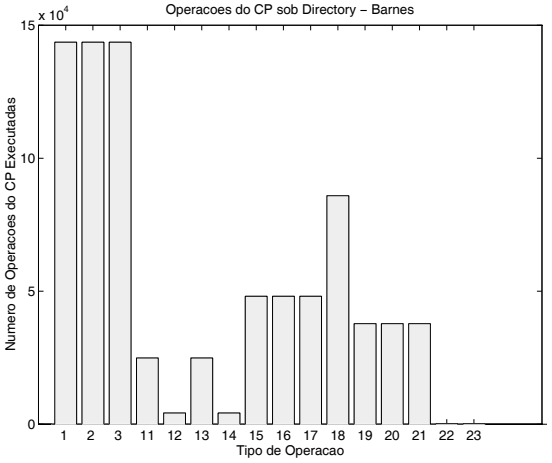

Figura 9: Operações do Controlador de Protocolo sob Directory - Barnes

Código	Operação
1	Pedido de página, processador p/ controlador
2	Pedido de página, controlador p/ controlador
3	Envio de página, controlador p/ controlador
4	Cria diff, processador p/ controlador, interrompe processador
5	Cria e envia diff, processador p/ controlador
6	Cria diff, processador p/ controlador, não interrompe processador
7	Pedido de diff, processador p/ controlador
8	Pedido de diff, controlador p/ controlador
9	Envia diff, processador p/ controlador
10	Envia diff, controlador p/ controlador
11	Envia e aplica diff, controlador p/ controlador, retorna *acknowledgment*
12	Envia e aplica diff, controlador p/ controlador, não retorna *acknowledgment*
13	Cria e envia diff, processador p/ controlador, espera *acknowledgment*
14	Cria e envia diff, processador p/ controlador, não espera *acknowledgment*
15	Pedido de linha diretório com interrupção na resposta, processador p/ controlador
16	Pedido de linha diretório com interrupção na resposta, controlador p/ controlador
17	Envio de linha diretório, controlador p/ controlador, interrompe processador
18	Atualiza linha de diretório, controlador p/ controlador
19	Pedido de linha diretório sem interrupção na resposta, processador p/ controlador
20	Pedido de linha diretório sem interrupção na resposta, controlador p/ controlador
21	Envio de linha diretório, controlador p/ controlador, não interrompe processador
22	Envia *write-notice*, processador p/ controlador
23	Recebe *write-notice*, controlador p/ controlador
24	Envia mensagem, processador p/ controlador
25	Recebe mensagem, controlador p/ controlador

Tabela 2: Operações do Controlador de Protocolo

sabemos (seção 3) que nesse protocolo as páginas compartilhadas são enviadas somente no ínicio do processamento da aplicação; todas as atualizações das páginas são feitas apenas através dos diffs. A única exceção para esse padrão é a aplicação Water, a qual é dominada pelo tráfego de mensagens de sincronização. O comportamento de Water é explicado pela grande quantidade de operações de sincronização realizadas na aplicação em comparação com os *page faults* ocorridos para cada página.

Os comandos de manipulação de diretórios (códigos 15 a 21) não aparecem nos gráficos de TreadMarks, pois esses são relacionados com a manipulação de diretórios. Como já foi visto na seção 3, os conceitos de *home node* e diretórios não se aplicam a TreadMarks.

O comportamento do protocolo Directory é bem diferente do comportamento de TreadMarks, como mostram as figuras 6 a 9. Em Directory, as atualizações através de diffs das páginas compartilhadas são realizadas apenas nos *home nodes*. Qualquer *page fault* em outro processador irá pedir uma página atualizada ao *home node* correspondente, o que justifica a grande concentração

de operações de transferência de páginas. O uso de *home nodes* e, consequentemente, a existência de entradas de diretório para registrar o compartilhamento de páginas nesses nós explica a grande quantidade de operações relacionadas com as linhas de diretórios (códigos 15 a 21).

Em resumo, podemos observar o uso intenso das funções básicas propostas para o CP. A manipulação de diffs é muito custosa e acontece em grande quantidade em TreadMarks. Tais resultados nos incentivaram a otimizar esta operação, justificando o uso de um *hardware* dedicado a geração e aplicação de diffs. O grande número de transações de páginas no protocolo Directory mostra a necessidade de evitar a interferência no processador remoto o máximo possível. A existência de uma entidade computacional adicional, o CP, no NCP_2 permite que esse tipo de interferência seja reduzido. O grande número de tranferências de mensagens pela rede indica a necessidade de implementação de mecanismos eficientes para esse tipo de operação; tais mecanismos devem evitar envolver o sistema operacional sempre que possível.

6 Conclusão

Nesse artigo descrevemos os aspectos de *software* do computador paralelo NCP_2, enfatizando a utilização das estruturas de dados e comandos do seu controlador de protocolos. Através da simulação de quatro aplicações reais sobre os protocolos TreadMarks e Directory modificados para execução no NCP_2, buscamos demonstrar a utilização dos comandos do controlador.

Em resumo, concluímos que é fundamental a minimização de interferências entre processadores executando *software DSMs* através do uso de uma unidade de processamento adicional como o nosso controlador de protocolos. Além disso, concluímos ser necessária a otimização das operações de manipulação de diffs através de *hardware* dedicado. Tais conclusões justificam a implementação corrente do nosso controlador de protocolos.

Referências

[ABS+96] C. L. Amorim, R. Bianchini, G. Silva, R. Pinto, M. Hor-Meyll, M. De Maria, L. Whately, and J. Barros Jr. A Segunda Geração de Computadores de Alto Desempenho da COPPE/UFRJ. In *Anais do Simósio Brasileiro de Arquitetura de Computadores*, Agosto 1996.

[BKP+96] R. Bianchini, L. Kontothanassis, R. Pinto, M. De Maria, M. Abud, and C. L. Amorim. Hiding Communication Latency and Coherence Overhead in Software DSMs. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct 1996.

[C*et al.*93] D. Culler *et al.* Parallel Programming in Split-C. In *Proceedings of Supercomputing '93*, pages 262–273, November 1993.

[GLL+90] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. L. Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiproces-

sors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26, May 1990.

[KDCZ94] P. Keleher, S. Dwarkadas, A. Cox, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proceedings of the USENIX Winter '94 Technical Conference*, pages 17–21, Jan 1994.

[SHMM+96] G. Silva, M. Hor-Meyll, M. De Maria, R. Pinto, L. Whately, J. Barros Jr., R. Bianchini, and C. L. Amorim. O Hardware do Computador Paralelo NCP_2 da COPPE/UFRJ. Technical Report ES-394/96, COPPE Sistemas e Computação, Universidade Federal do Rio de Janeiro, Junho 1996.

[VF94] J. E. Veenstra and R. J. Fowler. MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In *Proceedings of the 2nd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '94)*, 1994.

[WOT+95] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, May 1995.