O Hardware do Computador Paralelo NCP_2 da COPPE/UFRJ *

G. Silva, M. Hor-Meyll, M. de Maria, R. Pinto,
L. Whately, 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-394/96, 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. Neste artigo estaremos descrevendo os aspectos de *hardware* do NCP_2, dando grande ênfase à arquitetura dos controladores de protocolo e aos estudos que nos levaram ao seu dimensionamento atual. Baseados na nossa experiência no desenvolvimento do NCP_2 e nos nossos resultados preliminares de simulação, concluímos que o NCP_2 deverá alcançar uma boa razão custo–desempenho, representando assim uma atraente opção para a indústria nacional.

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. This paper describes the NCP_2 hardware, concentrating on the architecture of the protocol controllers, and on the research that led to its current design. Based on our current development experience and on our preliminary simulation results, we conclude that the NCP_2 will exhibit a good cost–performance ratio, and thus will represent an attractive system for our computer industry.

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

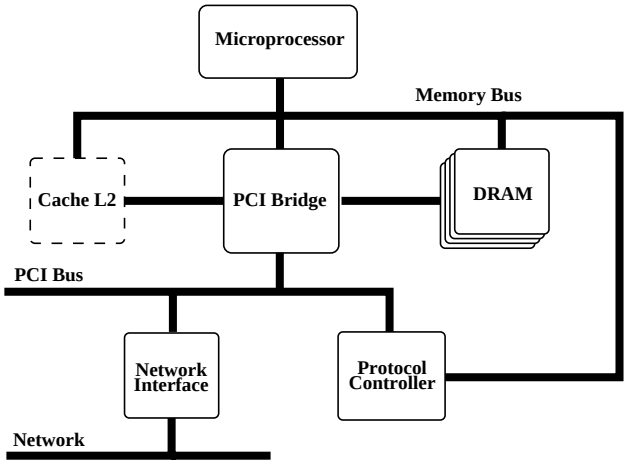

Figura 1: Diagrama em blocos de cada nó do NCP_2

1 Introdução

Os sistemas com memória compartilhada distribuída (DSM) implementados exclusivamente em *software* oferecem ao programador uma abstração de memória compartilhada sobre um *hardware* de troca de mensagens. Estes sistemas são uma alternativa de baixo custo para a computação paralela, já que podem ser construídos com estações de trabalhos e sistemas operacionais comuns. Contudo, diversas aplicações que executam nesse tipo de sistemas DSM perdem boa parte do tempo de processamento com a comunicação e manutenção da coerência dos dados, o que limita o seu desempenho.

O projeto NCP_2 está desenvolvendo um controlador de protocolos programável simples para esconder ou tolerar o *overhead* dessas tarefas de comunicação e coerência. No sistema NCP_2 cada controlador de protocolo está associado a um processador principal e provê três formas de diminuir as perdas de desempenho: a) o processamento de tarefas básicas de comunicação e coerência fora do processador principal; b) a busca antecipada de páginas e atualizações (também chamadas "*diffs*") das mesmas; e c) a geração e aplicação de diffs com suporte de *hardware*.

Neste artigo estaremos apresentando a arquitetura do primeiro protótipo do sistema paralelo NCP_2, situando-a em termos de capacidade de processamento e de comunicação, logo após esta introdução. Na seção 3, descrevemos as motivações para elaboração do controlador de protocolo e suas características principais. Na seção 4, descrevemos as estruturas de dados mais importantes do controlador, e analisamos os resultados de simulações realizadas para o dimensionamento dessas estruturas. A seção 5 faz uma comparação com trabalhos similares desenvolvidos recentemente e a seção seguinte apresenta nossas conclusões e perspectivas futuras.

2 Descrição Geral do Primeiro Protótipo do NCP_2

O NCP_2 [ABS+96] apresenta uma arquitetura básica do tipo memória distribuída, em que a comunicação entre processadores se dá por troca de mensagens através de uma rede de interconexão.

A rede de interconexão utilizada nesta fase do projeto é uma rede comercial, a Myrinet [Nan95], que possui uma latência mínima de cerca de 500 ns, banda passante máxima por canal de 80 Mbytes/s e excelente relação custo/desempenho.

As placas dos nós de processamento utilizadas na versão atual incluem microprocessadores RISC superescalares, PowerPC 604 [Mot94], comportam até 128 Mbytes de memória, cache de nível dois com até 1 Mbyte e interface para barramento PCI. Este conjunto confere a cada nó considerável capacidade de processamento e de armazenamento, a um custo relativamente baixo.

O barramento PCI é um barramento síncrono, que opera a uma frequência de 33 Mhz e permite taxas de transferência de até 132 Mbytes/s [PCI95]. O PCI tem surgido como um padrão para os computadores pessoais e estações de trabalho lançados recentemente no mercado. Neste barramento são conectadas as placas de interface com a rede de interconexão e o controlador de protocolo. A seguir faremos uma descrição mais detalhada deste controlador.

3 O Controlador de Protocolos

3.1 Motivação

As funções do controlador de protocolo foram definidas, abstratamente, de modo a fornecer ao processador principal suporte para os mecanismos básicos usados pelos DSMs baseados em *software*, quais sejam:

1. Pedido e resposta de página remota;
2. Pedido e resposta de *diff* remoto;
3. Criação e aplicação de *diff*;
4. Recepção e envio de mensagens;
5. Manipulação de diretórios.

A simples utilização de um segundo processador não é suficiente para a execução destas tarefas eficientemente. Um segundo processador teria que disputar o acesso à memória principal para o acesso às páginas compartilhadas, causando muita interferência com a computação útil. Uma interface de rede mais sofisticada, com DMA direto da área de usuário poderia ser solução para a transferência de páginas e mensagens, mas existem dificuldades para acesso direto a área de usuário e a manipulação dos *diffs* continuaria sem solução satisfatória.

Outro ponto importante é a necessidade da geração e aplicação de *diffs*. Normalmente os *diffs* são obtidos com a geração de cópias das páginas compartilhadas. Ao chegar um pedido de *diff*, o mesmo seria atendido pela comparação, palavra por palavra, entre a página atual (modificada) e a cópia realizada anteriormente (sem modificações). Esta operação possui custo significativo, e a obtenção de um mecanismo simples e eficiente foi um dos alvos principais dos nossos estudos na elaboração do controlador de protocolo.

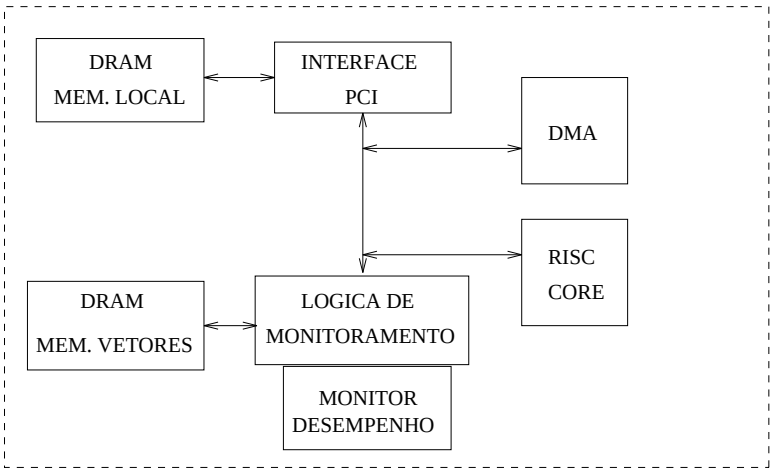

Figura 2: Módulos do Controlador de Protocolos

Procuramos então elaborar um *hardware* que executasse as tarefas acima mencionadas, da maneira mais eficente e simples possível. As tarefas restantes, usualmente mais complexas e específicas de cada tipo de protocolo, continuam a ser executadas no processador principal.

3.2 Módulos do Controlador

Conforme vemos na figura 2, o controlador de protocolos possui os seguintes módulos:

- Processador Local;
- Memória Local;
- Interface PCI;
- Lógica de *Snoopy;*
- Memória de Vetores;
- Monitor de Desempenho;
- DMA.

O processador local é responsável pela interpretação dos comandos que chegam ao controlador. Como a maior parte dos comandos é simples e contamos com um DMA para a realização das transferências de *diffs* e páginas, este processador não necessita possuir grande capacidade de processamento. Em uma primeira fase estaremos utilizando um processador comercial, o PowerPC 603, com forma de obtermos um protótipo em um curto espaço de tempo. Em uma etapa posterior, ele será substituido por um projeto próprio ou ainda um processador com capacidade mais adequada.

A memória local possui a princípio 4 Mbytes e se destina a armazenar algumas estruturas de dados (seção 4.2) para comunicação com o processador principal. O uso dessa memória local tem

como objetivo minimizar a interferência das operações realizadas pelo controlador de protocolo nos acessos à memória feitos pelo processador principal. O programa de controle do processador local também é carregado na memória local.

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, se estas forem necessárias. Os acessos provenientes do barramento PCI para a memória principal são vistos pelo processador local, o qual pode invalidar dados obsoletos na sua cache. O barramento PCI que estaremos utilizando possui largura de dados e endereços de 32 bits e frequência de operação de 33 Mhz. Isto é o suficiente para prover banda passante adequada para as transferências de páginas e *diffs* de/para a interface de rede.

Os outros módulos do controlador fazem parte do *hardware* específico que estamos construindo e implementando em FPGAs. A lógica de *Snoopy* está conectada ao barramento do processador principal e monitora os acessos de escrita realizados às paginas compartilhadas. Estes acessos são enfileirados em uma memória FIFO aguardando serem atualizados na memória de vetores. No caso de a FIFO encher, um sinal é ativado no barramento do processador principal, impedindo que novos acessos sejam realizados até que a FIFO seja esvaziada. Na mesma FPGA do Snoopy colocamos um controlador de DRAM, que faz o controle dos acessos à memória de vetores; uma lógica *slave* que permite o acesso externo ao conteúdo da memória de vetores (descrita a seguir); e o monitor de desempenho.

O módulo da memória de vetores anota todos os endereços das escritas que são realizadas pelo processador principal em páginas compartilhadas. Só precisamos anotar os endereços que são modificados, pois o dado modificado está disponível na memória principal e será utilizado quando precisarmos enviar um *diff* para a interface de rede. Esses endereços são anotados de uma forma comprimida, ou seja, um bit para cada 4 bytes na memória principal. Deste modo, com apenas 1 Mbyte de capacidade de armazenamento, a memória de vetores pode controlar as escritas realizadas em até 32 Mbytes de memória física. Como os endereços armazenados são físicos, isto é, os observados no barramento após a translação realizada pela gerência de memória, existe a necessidade de uma tabela de conversão para endereços virtuais, que são aqueles conhecidos pelas aplicações. Esta tabela pode ser localizada tanto na memória local quanto na memória principal.

O monitor de desempenho vai permitir o levantamento de dados para avaliar o comportamento do controlador de protocolos. Podem ser monitorados o tamanho médio da FIFO, o total de escritas realizadas pelo processador principal e o total de ciclos em que o processador principal esteve bloqueado, entre outros. Esta lógica é capaz também de fornecer um relógio global para todos os nós processadores, de forma a permitir a monitoração de desempenho em mais alto nível.

O DMA realiza transferências de áreas de memórias contíguas (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* de controle de coerência. Estaremos implementando este módulo também utilizando FPGAs de grande capacidade e linguagem de descrição VHDL. O DMA é um componete crítico ao NCP_2 e por isso esperamos conseguir uma frequência de operação compatível com as taxas de transferência de dados do PCI.

Constante do Sistema	Valor *Default*
Número de processadores	16
Todas as interrupções	400 ciclos
Tamanho da página	4K bytes
Tamanho da cache por processador	128K bytes
Tamanho do *write buffer*	4 entradas
Tamanho da linha de cache	32 bytes

Tabela 1: Valores *Default* dos Parâmetros do Sistema. 1 ciclo = 10 ns.

4 Dimensionamento do NCP_2

Para dimensionar adequadamente as estruturas existentes no controlador de protocolos e para avaliar o impacto das soluções adotadas, realizamos um conjunto de simulações descrito nesta seção.

4.1 Metodologia

O simulador utilizado é constituído de duas partes: *front end,* [VF94], que simula a execução dos processadores, e *back end,* que simula o sistema de memória em detalhes (*write buffers* e caches com tamanhos finitos, TLB, emulação completa do protocolo, custo de transferência na rede de interconexão incluindo efeitos de contenção, e custos de acesso à memória incluindo efeitos de contenção). O *front end* chama o *back end* em todas as referências a dados (assumimos que a busca de instrução sempre resulta em *hit* na cache). O *back end* decide quais processadores ficam bloqueados esperando por memória (ou outros eventos) e quais continuam sua execução. Visto que esta decisão é realizada *on line*, o *back end* altera a temporização do *front end.* Sendo assim, a forma como as instruções são intercaladas pelos processadores depende do comportamento do sistema de memória, e o fluxo de controle em um processador pode mudar como consequência da temporização das referências à memória.

Simulamos em detalhe uma rede de estações de trabalho com 16 nós. Cada nó consiste de um processador de computação, um *write buffer,* uma cache de dados de primeiro nível diretamente mapeada (assumimos que todas as instruções são executadas em um ciclo), memória local e um roteador de rede em malha (utilizando o roteamento *wormhole*). A tabela 1 resume os parâmetros *default* usados nas simulações. Todos os tempos são dados em ciclos de processador de 10-ns. *Note que o sistema computacional que simulamos não modela exatamente o primeiro protótipo do NCP_2; nossas simulações visam simplesmente aproximar o desempenho do protótipo e, principalmente, avaliar diferentes opções de projeto.*

Foram utilizados cinco programas na simulação: Barnes, Radix, Water, Ocean e Em3d. As quatro primeiras aplicações são do Splash-2 suite [WOT+95]. Estas aplicações foram executadas com os tamanhos de entrada *default* para 32 processadores, como sugerido pelos pesquisadores de

Aplicação	Tamanho da Entrada
Em3d	40064 nós, 10% remoto
Barnes	4K partículas
Radix	1M inteiros, radix 1024
Ocean	258×258 oceano
Water	512 moléculas

Tabela 2: Aplicações e Tamanho das Entradas.

Stanford, com exceção de Barnes. A tabela 2 lista as aplicações e seus tamanhos de entrada.

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.* Radix é um *kernel* que ordena números inteiros. O algoritmo é iterativo, executando uma iteração por dígito de 1M chaves. Water é uma simulação dinâmica de moléculas, que calcula forças inter- e intramoleculares em um conjunto de 512 moléculas de água. Utiliza-se um algoritmo $\mathcal{O}(n^2)$ para a computação das interações. Ocean estuda movimentos em grande escala de oceanos baseado nas suas correntes. Simulamos uma grade oceânica de dimensão 258×258. Em3d [C*et al.*93] simula a propagação de ondas eletromagnéticas em objetos 3D. Simulamos 40064 objetos elétricos e 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.

4.2 Estruturas de Dados

Nesta seção descrevemos algumas estruturas de dados que são necessárias para o funcionamento do controlador de protocolos e sua localização [WPS+96]. O dimensionamento destas estruturas determinou a quantidade de memória local disponível no controlador de protocolos.

Ao receber uma mensagem, ou aplicar um *diff*, o controlador de protocolos deve identificar a que página de usuário se destina e transferir os dados diretamente para esta área. Isto obriga a existência de uma estrutura na memória do controlador de protocolos, que mapeia os endereços físicos para endereços lógicos, para as páginas que estão sob compartilhamento. Chamamos essa estrutura de tabela de conversão de endereços. No NCP_2 essa estrutura deve ter cerca de 8K entradas, uma para cada página de memória física que pode ser compartilhada. Cada entrada possui um campo com o número da página virtual, outro com número da página física e um indicando se a entrada é válida ou não.

A tabela de diretórios faz o controle de acesso dos diversos nós processadores às páginas compartilhadas. Esta tabela é utilizada pelo protocolo de coerência para determinar que nós de processamento devem ser notificados das modificações realizadas em uma página, por exemplo. No caso de um processador remoto desejar atualizar esta tabela, um pedido é enviado através da rede ao controlador de protocolos responsável pela página correspondente, que disputa o acesso

Aplicação	Tot Escritas	Tam Máx	Tam Médio	*Hit Page*	*Hit Buffer*
Em3d	480.768	1	0.02	99%	13%
Barnes	1.038.203	19	0.14	77%	16%
Radix	7.718.904	62	41.58	40%	22 %
Ocean	52.150.842	50	0.12	67%	19 %
Water	2.818.579	5	0.09	88%	25 %

Tabela 3: Estatísticas sobre a FIFO

exclusivo a esta tabela, atualizando-a de acordo com a requisição do processador remoto. No caso de mais de um processador remoto solicitar acesso a esta tabela ao mesmo tempo, os pedidos são enfileirados. Esta tabela também possui 8K entradas e cada entrada possui um bit de controle por nó de processamento e um ponteiro para a fila de pedidos pendentes correspondente.

Uma outra estrutura existente na memória do controlador é uma fila de comandos, onde são enfileirados os pedidos realizados pelo processador principal ao controlador de protocolos. Os pedidos podem ter prioridades distintas, de modo que pedidos de maior prioridade avançam mais rápido que os de menor. Este esquema é utilizado, por exemplo, para evitar que buscas antecipadas de dados atrasem outros pedidos dos quais o processador principal depende para prosseguir sua execução. O tamanho desta estrutura é variável e dependente da aplicação, devendo ser determinado pelas simulações.

O processador principal pode atualizar a sua cache de nível de dois a uma taxa maior que a taxa de atualização da memória de amostragem, então consideramos o uso de uma pequena memória FIFO, para suportar os picos de escrita, sem que a execução da aplicação sofra qualquer atraso no processador principal. No caso desta FIFO encher, o processador principal é impedido de utilizar o barramento, até que esta FIFO seja novamente esvaziada. Esta FIFO está situada na FPGA do monitor de desempenho e seu tamanho inicial foi definido em 64 posições. As simulações, que são analisadas na próxima seção, procuram avaliar a adequação deste e de outros valores.

4.3 Análise dos Resultados

Foram vários os resultados obtidos a partir das nossas simulações. Alguns deles, de impacto sobre a arquitetura do controlador de protocolos, são analisados a seguir.

A tabela 3 apresenta informações sobre o comportamento da FIFO que armazena os endereços monitorados no barramento do processador principal. Os valores na coluna *Tot Escritas* mostram qual foi o total de escritas compartilhadas realizadas por cada aplicação. São valores que acreditamos significativos para validar o experimento realizado. A coluna *Tam Máx* diz qual foi o maior tamanho que a FIFO atingiu durante a simulação das aplicações. A coluna *Tam Médio* mostra o valor médio do tamanho da FIFO durante todos os ciclos da simulação. O máximo utilizado para o tamanho da fila na simulação foi de 62 posições. Este limite é atingido pelo programa

Aplicação	Tam Médio	Tam Máx
Em3d	0.5	4
Barnes	1.6	107
Radix	60.3	1024
Ocean	1.8	28
Water	0.8	14

Tabela 4: Estatísticas sobre a fila de comandos

Aplicação	Tam Médio *Diff* Bytes	Tam Médio Msg Bytes
Em3d	626	460
Barnes	89	1001
Radix	524	1686
Ocean	1655	1080
Water	608	1235

Tabela 5: Estatísticas sobre *diffs* e mensagens

radix, que é um programa de ordenação com *bursts* de acessos às páginas compartilhadas com pouca ou nenhuma localidade. Neste caso entra em atuação o mecanismo descrito anteriormente para impedir que o processador principal prossiga com as escritas, preservando desta maneira a memória de vetores consistente com a memória principal. Na maior parte dos casos esse tamanho máximo não é atingido e os valores médios mostram que a FIFO cumpre sua função, ou seja, permitir que a amostragem prossiga sem interferências significativas com a computação.

Outros dados significativos na tabela 3 são aqueles de *Hit Page* e *Hit Buffer*. Os valores em *Hit Page* indicam o percentual de vezes que um acesso de escrita é para a mesma página que o acesso anterior. Os valores em *Hit Buffer* indicam o percentual de escritas que são realizadas em um mesmo *buffer*. Cada *buffer* possui 32 bits e corresponde a uma palavra da memória local. Um valor alto para *Hit Buffer* significa que necessitamos realizar um número pequeno de acessos à memória local. As estatísticas de *Hit Buffer* nos mostram valores entre 8:1 e 4:1 para a relação entre acessos monitorados e acessos a memória local. Isto é o bastante para compensar a relação entre os tempos de acesso a memória local e a memória cache do processador principal. Os valores de *Hit Page* nos indicam que a lógica de controle deve utilizar um esquema de acesso a memória de vetores em bloco, eventualmente com o uso de uma pequena cache, de modo a abranger uma página inteira da memória física.

A tabela 4 mostra os tamanhos máximo e médio que a fila de comandos apresenta na simulação das aplicações. O pequeno valor médio mostra que na maior parte das vezes não existe

Aplicação	Redução de TE
Em3d	43%
Barnes	33%
Radix	29%
Ocean	51%
Water	11%

Tabela 6: Redução de tempo de execução com controladores.

pedido pendente, indicando um controlador de protocolos bem dimensionado, capaz de atender aos pedidos sem que ocorram atrasos importantes. O pior caso não é preocupante em termos de espaço de memória utilizado e ocorre justamente no programa **radix**, pelo que já foi descrito anteriormente.

A tabela 5 se refere aos tamanhos médios dos *diffs* e mensagens que circulam na rede de interconexão. Os tamanhos de mensagem são exatamente dentro do esperado e confirmam as premissas assumidas quando da elaboração da arquitetura do NCP_2: mensagens grandes que tornam adequado o uso de redes com latência relativamente alta e, consequentemente, de baixo custo.

A última tabela (tabela 6) comprova a eficiência de nossa proposta. Nela estão expressos os ganhos que podem ser obtidos com a execução das aplicações sobre Treadmarks com o suporte dos nossos controladores de protocolo versus Treadmarks exclusivamente por software. O programa **radix**, apesar de exigir bastante do controlador de protocolo, possui tempo de execução que é 43% do tempo necessário para a implementação exclusivamente por software, mostrando a eficiência do controlador de protocolo mesmo nesses casos mais difíceis. Uma análise mais detalhada desses resultados pode ser encontrada em [BKP+96]. De uma maneira geral, os ganhos obtidos pelo uso dos nossos controladores de protocolo são significativos e nos encorajam para que em breve possamos apresentar os resultados da execução destes programas no primeiro protótipo do NCP_2.

5 Trabalhos Correlatos

Diversos multiprocessadores com coerência de cache foram construídos recentemente [LLJ+93, ABC+95, KSR92]. Esses multiprocessadores conseguem obter um alto desempenho a custa de projetos complexos com uso intensivo de lógica dedicada. Tais sistemas possuem custos elevados, limitando significativamente a difusão do seu uso. Devido a sua simplicidade e utilização de componentes comerciais, nosso projeto consegue ter um custo muito menor e também um menor ciclo de desenvolvimento, o que nos coloca em uma classe diferente de sistemas de memória compartilhada distribuída.

Nosso trabalho apresenta algumas idéias utilizadas nos projetos pioneiros do computador

FLASH de Stanford [K*et al.*94] e de Typhoon de Wisconsin [RLW94]. Estes sistemas buscam prover um compartilhamento eficiente de dados ao nível de linhas de cache. O uso de transferência de pequenos blocos de dados exige redes de interconexão com baixa latência para atingir-se este objetivo, o que torna esses projetos mais complexos e caros. Ao contrário destas abordagens, nosso protocolo de coerência baseado em páginas permite o uso de um processador de protocolo mais simples e uma rede de interconexão comercial de baixo custo conectada a um barramento PCI [Nan95].

Do ponto de vista dos algoritmos, nossa pesquisa parte do trabalho realizado por vários sistemas que provêem memória compartilhada e coerência em software, utilizando variantes de consistência relaxada. Tanto Munin [CBZ91] como TreadMarks [KDCZ94] foram projetados para execução em redes com estações de trabalho, sem nenhum hardware de suporte. Nosso trabalho [BKP+96] mostra que um aumento de desempenho significativo pode ser obtido com uso de um hardware simples para esconder latência de comunicação e as perdas com a manutenção de coerência em sistemas deste tipo.

O trabalho de Iftode *et al.* [IDFL96] propõe AURC, um software DSM que usa um hardware específico para atualização automática de dados compartilhados. No artigo, AURC é comparado a TreadMarks padrão, usando algumas das mesmas aplicações por nós utilizadas. Os autores mostram que o AURC supera Treadmarks em todas as aplicações. O desempenho das aplicações quando controladores de protocolo são utilizados é igual ou melhor que o de AURC na maior parte dos casos [BKP+96].

6 Conclusões

Apresentamos neste artigo o hardware do controlador de protocolo do NCP_2. A nossa proposta é prover suporte para os diversos protocolos de coerência com baixo custo e simplicidade, mas também com ganhos significativos de desempenho na aplicações paralelas em *software DSMs*. Como é um trabalho pioneiro, muito tempo foi gasto na elaboração de uma especificação acurada, envolvendo aspectos de *hardware* e *software*, que cobrisse todas as situações existentes em um ambiente complexo como esse. Os resultados até agora obtidos são bastante promissores e esperamos em breve apresentar os resultados da avaliação final de desempenho do NCP_2.

Referências

[ABC+95] A. Agarwal, R. Bianchini, D. Chaiken, K.L. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung. The MIT Alewife Machine: Architecture and Performance. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA)*. ACM, June 1995.

[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 De-

sempenho 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.

[CBZ91] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Implementation and Performance of Munin. In *Proceedings of the 13th Symposium on Operating Systems Principles*, October 1991.

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

[IDFL96] L. Iftode, C. Dubnicki, E. Felten, and K. Li. Improving Release-Consistent Shared Virtual Memory using Automatic Update. In *Proceedings of the 2nd IEEE Symposium on High-Performance Computer Architecture*, February 1996.

[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.

[K*et al.*94] J. Kuskin *et al.* The Stanford FLASH Multiprocessor. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, Chicago, IL, April 1994. IEEE.

[KSR92] Kendall Square Research. *KSR1 Principles of Operation*, 1992.

[LLJ+93] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The DASH Prototype: Logic Overhead and Performance. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):41–61, Jan 1993.

[Mot94] Motorola. *PowerPC 604 RISC Microprocessor User's Manual*, 1994.

[Nan95] Nanette J. Boden and Danny Cohen and alli. Myrinet - A Gigabit-per-second Local Area Network. *IEEE Micro*, February 1995.

[PCI95] PCI Special Interest Group. *PCI Local Bus Specification - Rev. 2.1*, 1995.

[RLW94] Steven K. Reinhardt, James R. Larus, and David A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, Chicago, IL, April 1994. IEEE.

[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.

[WPS+96] L. Whately, R. Pinto, G. Silva, M. Hor-Meyll, M. De Maria, J. Barros Jr., R. Bianchini, and C. L. Amorim. O Software do Computador Paralelo NCP_2 da COPPE/UFRJ. Technical Report ES-395/96, COPPE Sistemas e Computação, Universidade Federal do Rio de Janeiro, Junho 1996.